UCUENCA

Facultad de Ingeniería

Carrera de Ingeniería de Sistemas

Desarrollo de un prototipo informático de soporte para gestionar conjuntos de datos de inventarios de ciclo de vida representativos del contexto ecuatoriano

Trabajo de titulación previo a la obtención del título de Ingeniero de Sistemas.

Autor: Kamila Nicole Molina Orellana.

CI: 0105072797

Correo: kamila.molinaorellana@gmail.com

Autor: Estuardo Mateo Quizhpi Peralta.

CI: 0106039332

Correo: mateoquizhpi96@gmail.com

Directora: Ing. Lorena Catalina Sigüenza Guzmán .

CI: 0102659687

Cuenca - Ecuador

02-mayo-2022

Resumen:

En la actualidad, los proyectos de investigación y monitoreo de impacto ambiental generan un gran volumen de datos con diferentes formatos de almacenamiento. Las iniciativas enfocadas en organizar y evaluar dichos datos necesitan una organización dentro de lo que se conoce como una base de datos ambiental. Es por esto que, gracias a los avances tecnológicos en hardware y software para el almacenamiento y visualización de información, también han aumentado las expectativas de consultores y clientes con respecto a la gestión de los datos obtenidos a través del ciclo de vida del proyecto. En vista de estas expectativas, se ve la necesidad de crear un prototipo informático enfocado al almacenamiento y gestión de metainformación ambiental, datos importantes para conocer el recorrido de un producto. El prototipo fue construido con la metodología ágil llamada SCRUM que se basa en el trabajo en equipo generando resultados en un lapso determinado de tiempo. Adicionalmente, para el desarrollo de la aplicación se usaron diversas tecnologías como el lenguaje de programación Python, el sistema gestor de base de datos PostgreSQL, y el Framework de desarrollo Django Rest Framework.

El aporte práctico de la investigación es el desarrollo de un prototipo informático de implementación web, que gestiona información de archivos de análisis de ciclo de vida (LCA por sus siglas en inglés). Donde uno o varios usuarios tienen acceso a la metainformación de los archivos LCA obtenidos por el inventario de ciclo de vida (LCI por sus siglas en inglés) de un producto o servicio. Estos datos son publicados en la base de datos ambiental implementada como parte del prototipo. Los usuarios pueden realizar consultas personalizadas sobre la metainformación y con esto tener datos más concretos y relevantes para sus propios intereses, así como pueden analizar el recorrido de la información a través de diferentes versiones.

De esta manera se concluye que, con el desarrollo del prototipo informático web, su implementación y prueba, este trabajo de titulación apoya en el registro y análisis de metainformación ambiental para la gestión de inventarios de ciclo de vida.

Palabras claves: UNEP: Programa de las Naciones Unidas para el Medio Ambiente. **LCA:** Evaluación del Ciclo de Vida. **LCI:** Inventario del Ciclo de Vida. **LCC:** Costeo del ciclo de vida. **API:** Interfaz de Programación de Aplicaciones.

Abstract:

Environmental impact research and monitoring projects currently generate a large volume of data with different storage formats. Initiatives focused on organizing and evaluating such data need an organization within what is known as an environmental database. Therefore, thanks to technological advances in hardware and software for the storage and visualization of information, the expectations of consultants and clients regarding the data management obtained throughout the project's life cycle have also increased. Given these expectations, there is a need to create a prototype focused on storing and managing environmental meta-information essential data to know the path of a product. To this end, the prototype was built with the agile methodology called SCRUM, based on teamwork generating results in a given period. Additionally, various technologies were used to develop the application, such as the Python programming language, the PostgreSQL database management system, and the Django Rest Framework development framework.

The practical contribution of the research is the development of a prototype for web implementation, which manages information from life cycle analysis (LCA) files. One or several users have access to the meta-information of the LCA files obtained by the life cycle inventory (LCI). These data are published in the environmental database implemented as the prototype. Users can make personalized queries on the meta-information and have more specific and relevant data for their interests. They can also analyze the path of the information through different versions.

In this manner, it is concluded that, by developing the web prototype, its implementation, and testing, this titling work supports the registration and analysis of environmental meta-information to manage life cycle inventories.

Keywords: UNEP: United Nation Environment Programme. **LCA:** Life Cycle Assessments. **LCI:** Life Cycle Inventories. **LCC:** Life Cycle Costing. **API:** Application Programming Interface.

Índice del Trabajo

UCUENCA

Cláusula de licencia y autorización para publicación en el Repositorio Institucional

Kamila Nicole Molina Orellana, en calidad de autora y titular de los derechos morales y patrimoniales del trabajo de titulación "Desarrollo de un prototipo informático de soporte para gestionar conjuntos de datos de inventarios de ciclo de vida representativos del contexto ecuatoriano", de conformidad con el Art. 114 del CÓDIGO ORGÁNICO DE LA ECONOMÍA SOCIAL DE LOS CONOCIMIENTOS, CREATIVIDAD E INNOVACIÓN reconozco a favor de la Universidad de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos.

Asimismo, autorizo a la Universidad de Cuenca para que realice la publicación de este trabajo de titulación en el repositorio institucional, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Cuenca, 2 de mayo de 2022

Kamila Nicole Molina Orellana

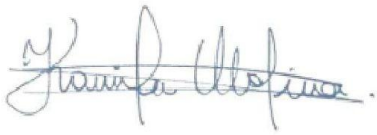
C.I: 0105072797

UCUENCA

Cláusula de Propiedad Intelectual

Kamila Nicole Molina Orellana, autora del trabajo de titulación "Desarrollo de un prototipo informático de soporte para gestionar conjuntos de datos de inventarios de ciclo de vida representativos del contexto ecuatoriano", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autora.

Cuenca, 2 de mayo de 2022



Kamila Nicole Molina Orellana

C.I: 0105072797

UCUENCA

Cláusula de licencia y autorización para publicación en el Repositorio Institucional

Estuardo Mateo Quizhpi Peralta, en calidad de autor y titular de los derechos morales y patrimoniales del trabajo de titulación "Desarrollo de un prototipo informático de soporte para gestionar conjuntos de datos de inventarios de ciclo de vida representativos del contexto ecuatoriano", de conformidad con el Art. 114 del CÓDIGO ORGÁNICO DE LA ECONOMÍA SOCIAL DE LOS CONOCIMIENTOS, CREATIVIDAD E INNOVACIÓN reconozco a favor de la Universidad de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos.

Asimismo, autorizo a la Universidad de Cuenca para que realice la publicación de este trabajo de titulación en el repositorio institucional, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Cuenca, 2 de mayo de 2022

_____

Estuardo Mateo Quizhpi Peralta

C.I: 0106039332

Cláusula de Propiedad Intelectual

Estuardo Mateo Quizhpi Peralta, autor del trabajo de titulación "Desarrollo de un prototipo informático de soporte para gestionar conjuntos de datos de inventarios de ciclo de vida representativos del contexto ecuatoriano", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autora.

Cuenca, 2 de mayo de 2022

_____

Estuardo Mateo Quizhpi Peralta

C.I: 0106039332

Dedicatoria.-

Dedico de manera especial a mi mami Mirian, a mi papi Edgar, a mi ñaño Sebas y a mi enamorado Juanjo. Ustedes han sido un pilar fundamental en este proceso, por ser un gran ejemplo de fuerza y dedicación. Me han motivado diariamente para culminar con éxito mis metas.

“Nadie te atacará tanto como tú misma en tu cabeza”. A todas esas personas que a pesar de las circunstancias siguen con nosotros. No se rindan. Sean intrépidos e indefensos, mientras derriban los muros a su alrededor. Mantengan la esperanza, y aunque no es un momento perfecto, valdrá la pena. Nunca renuncien ni se den por vencidos aunque sientan que la vida los está destrozando. Son valientes.

Kamila Molina Orellana.

Dedicatoria.-

Mi sentida dedicatoria para las mujeres más importantes de mi vida, mi madre Ruth y mi hermana Mariuxi, sin ellas jamás hubiera podido potenciar ningún valor en cualquier aspecto individual. De igual manera, dedico a toda mi familia y amigos que son un pilar imprescindible en apoyo, soporte y cariño debido a que siento que tengo un entorno enriquecido por personas maravillosas.

Mateo Quizhpi Peralta.

Agradecimiento.-

A mis padres Edgar y Mirian por haberme formado con valores y principios que han hecho de mí una mujer fuerte, con convicciones, perseverante y valiente, por el apoyo que me han brindado y el amor con el que me han cobijado. Por las ganas de salir adelante. Agradezco a mi querido hermano Sebastián por estar siempre presente y ser esa felicidad diaria. A mi enamorado Juan José por todo el cariño y alegría que me ha brindado durante estos años. De manera especial a *Dios,* por la dicha de la vida, por la fuerza ante cada barrera, por la valentía que me da, por permitirme estar aquí el día de hoy. Gracias a ustedes por todo el apoyo, soporte y amor, por ser constantes en los momentos difíciles y por ser incondicionales en los momentos más importantes de mi vida.

A la Universidad de Cuenca por las herramientas brindadas para crecer profesionalmente. Tanto a los docentes que me acompañaron desde mis inicios en la carrera, hasta el fin de la misma. Finalmente, de manera especial a mis tutores de tesis, Brandon Kuczenski, Erik Sigcha, Lorena Sigüenza, Paul Vanegas; a ustedes por brindarme su tiempo y paciencia para culminar este trabajo.

Concluyo agradeciendo a todos esos amigos que nunca me fallaron, esos amigos que sí saben el valor de la amistad, esos amigos que solamente cuento de verdad, ustedes saben quienes son. Gracias por siempre estar conmigo en los buenos y malos momentos y gracias infinitas por pertenecer a mi pequeño círculo.

Kamila Molina Orellana.

Agradecimiento.-

Agradezco infinitamente a mis seres queridos, de manera especial a mis padres Ruth y Estuardo, quienes me han inculcado con un espíritu de superación, constancia y valentía validos para conseguir mis objetivos y metas personales. En este camino el apoyo de mi hermana Mariuxi ha sido fundamental para enfrentar con ímpetu y convicción cada uno de los límites que me han hecho lograr una mejor versión personal y profesional cada día.

De igual manera a la Universidad de Cuenca por ser una plataforma impulsora de desarrollo académico con ética y valores humanos. En particular a Brandon, Lorena, Paúl y Erik por el tiempo dedicado a impulsarnos a realizar este trabajo de titulación de manera óptima materializando todo nuestro potencial.

Finalmente agradezco también a mis queridos compañeros y amigos Luis, David y Kamila por transformar esta experiencia única y enriquecedora en una etapa entrañable de mi vida estudiantil.

Mateo Quizhpi Peralta.

1. Introducción:

El análisis de ciclo de vida (Life Cycle Assessment, LCA) es un estándar metodológico para valorar el potencial impacto ambiental que ocasiona la fabricación de productos o entrega de servicios. Esta metodología permite modelar los procesos industriales y los intercambios que tienen con el entorno, los cuales deben ocurrir para que los productos puedan ser entregados a un consumidor. Como menciona Kuczynski (2018), la técnica sobre el análisis del impacto ambiental está bien establecida, ampliamente practicada, y se basa en un extenso cuerpo de estandarización donde incluso destacan herramientas de desarrollo y también de análisis político. La importancia de esta experiencia va mucho más allá de la aplicación de la LCA porque, en términos más generales, ayuda a lograr los objetivos e iniciativas de desarrollo sostenible al sensibilizar a todos los niveles de gobierno y otras partes interesadas sobre estos beneficios del análisis ambiental (Vadenbo et al., 2020).

Muchos de los desafíos más comunes que se dan dentro del LCA están relacionados con que los principales contribuyentes a este método de análisis no han acordado un formato interoperable de la estructura y contenido del LCA. Sin embargo, una de las ventajas principales de este desarrollo, es que desde un enfoque sustentable, se requiere la aplicación de diversas acciones estructurales y no estructurales para solucionar, o al menos remediar los efectos negativos en el impacto ambiental. Estas acciones pueden ser consideradas como la creación de políticas ambientales que sean exigentes y tengan el fin de disminuir impactos actuales y futuros en el medio ambiente y, por consiguiente, proteger la salud humana.

El presente trabajo de titulación se refiere al estudio previo, análisis, diseño e implementación de bases de datos para un inventario ambiental. En este caso específico, una base de datos ecuatoriana (nacional) que sea interoperable con un formato de LCA ya establecido y con impacto entre los expertos del tema. El objetivo es apoyar el desarrollo de las bases de datos a nivel nacional de LCA, garantizando de esta forma la interoperabilidad con otras fuentes de datos. Uno de los principales desafíos de esta iniciativa de creación de base de datos de LCA es recopilar datos secundarios o conocidos como metadata. La metadata es esencial en los productos, procesos o industrias para la economía y el medio ambiente, al mismo tiempo que se garantiza una representación regional adecuada. Por lo tanto, como mencionan Vadenbo et al. (2020), es importante comprender el grado y la relevancia de ciertos datos como la representación geográfica requerida por los usuarios y las partes interesadas, pues recopilar datos de LCA regionalizados también ayuda a mejorar la experiencia local en métodos de ciclo de vida.

Actualmente, los proyectos de investigación y monitoreo de impacto ambiental, acostumbran a generar un gran volumen de datos con diferentes formatos (Ihobe, 2017). Por lo tanto, para organizar y evaluar estos datos eficientemente, es necesaria una cierta gestión de los mismos, enmarcada en una organización dentro de lo que se conoce como una base de datos ambiental. Cabe mencionar que, en el país, la mayoría de actividades de gestión de información se centraron en la recolección de datos para su almacenamiento futuro y no se le daba seguimiento a las estrategias para un análisis eficiente (Ramirez et al., 2019). Es por esto que, gracias a los avances tecnológicos en hardware y software para el almacenamiento y visualización de datos, se han transformado los procesos tradicionales de trabajo y se ha aumentado las expectativas de consultores y clientes con respecto a la gestión de los datos a través del ciclo de vida del proyecto. Sin duda alguna, aplicando estas técnicas al diseño e implementación de una base de datos ambiental ecuatoriana, se estaría dando un máximo provecho a la información referente al impacto ambiental.

1.1. Motivación y Contexto.

A nivel mundial, la creación de políticas públicas y la toma de decisiones enmarcadas dentro de los principios de sostenibilidad representan un reto de índole académica y política. Considerando que este principio aborda aspectos de carácter social, económico y ambiental (Fauzi et al., 2019), algunas alternativas prometedoras para hacer frente a esta problemática son las evaluaciones desde un enfoque de pensamiento de ciclo de vida de los productos y servicios (Fauzi et al., 2019; Klöpffer, 2008). Desde una perspectiva ambiental, un LCA es esencial dentro de un proceso de política legislativa (Reed, 2012; Seidel, 2016); sin embargo, para llevarla a cabo es indispensable contar con datos accesibles, confiables y transparentes, que en la mayoría de los casos son limitados, descontinuados o inexistentes si se los va a aplicar a nivel nacional (Wolf, 2014).

En efecto, una base datos nacional proporciona un número relevante de conjuntos de datos consistentes metodológica y técnicamente, que se mantienen y se actualizan incluso después de su inicio. Por ende, es importante realizar una revisión exhaustiva de las características y requerimientos que presentan diversas bases de datos mundiales, tales como ecoinvent, idea, GaBi o exiobase (OPENLCA Nexus, 2019), con la finalidad de determinar la estructura que debe seguir una base de datos nacional. Así mismo, estas bases de datos representan la situación de producción nacional, por lo que deben tener una suficiente calidad para ser usadas en una o más aplicaciones de LCA relevantes en la industria (Wolf, 2014).

En el Ecuador, los datos con los que se cuentan corresponden a inventarios de ciclo de vida (Life Cycle Inventories, LCI) de energía hidroeléctrica y termoeléctrica (Ramirez et al., 2019). Sin embargo, en temas nacionales prioritarios como: i) eficiencia energética, ii) producción y consumo sostenible, y iii) gestión integral de residuos (SENPLADES, 2017; Secretaría Técnica Planifica Ecuador, 2019), no se cuenta con datos en el formato adecuado. Tampoco con los requerimientos para estructurar una base de datos ambiental nacional que satisfaga simultáneamente los aspectos económicos, ambientales y sociales del desarrollo sostenible (Azapagic, 2017). Es por ello que, como proponen Vadenbo et al. (2020), se debe crear una hoja de ruta para la creación de bases de datos nacionales de LCA con el objetivo de mejorar la disponibilidad de datos en países con economías emergentes. Con esta afirmación, se lleva a cabo este proyecto de titulación para construir un sistema destinado a extraer, transformar, organizar y almacenar conjuntos de datos digitales obtenidos a través de métodos de evaluación del impacto del ciclo de vida. Para llevar a cabo esto, el Programa de Naciones Unidas para el Medio Ambiente, PNUMA (2011) presenta una lista de principios generales para las bases de datos de LCA con la siguiente estructura: accesibilidad, responsabilidad, exactitud, exhaustividad, coherencia, intercambiabilidad, materialidad, practicidad, garantía de calidad, pertinencia, reproducibilidad y transparencia. Estos aspectos ofrecen una guía hacia una iniciativa de base de datos nacional de LCA funcional (Vadenbo et al., 2020).

1.2. Planteamiento del problema.

Tomando en cuenta que, la falta de integración de la información requerida para llevar a cabo análisis ambientales y una toma de decisión sostenible, impide la gestión responsable de las organizaciones, resulta notoria la necesidad de aprovechar los sistemas gestores de datos. De esta manera, este tipo de herramientas brindan capacidades de almacenamiento, gestión y acceso a los datos, permitiendo que los usuarios del sistema puedan visualizar e interactuar con ellos de una forma más simple. Sin embargo, a causa de la falta de este tipo de herramientas orientadas al contexto ecuatoriano, surge la necesidad de la generación e implementación de una base de datos ambiental que permita gestionar la información obtenida por los diferentes proveedores de LCA.

En este sentido, es importante mencionar que la solución tecnológica debe cubrir un enfoque específico referente a la agrupación en cuatro grupos de información, tales como metainformación, intercambios (que son datos calculados), cantidades y flujos, de cada producto o servicio. Por lo tanto, se busca seleccionar y centralizar un grupo determinado de datos ambientales, con el

fin de lograr una organización de procesos sustentables en cuanto al manejo de inventarios ambientales en nuestra nación.

1.3. Solución propuesta.

Inicialmente, como mencionan Ramírez et al. (2019), es importante tener la iniciativa de construir una base de datos ambiental en un contexto nacional. Para el caso de Ecuador, la carencia de una tecnología de almacenamiento de datos específica que haga posible una evaluación confiable, motiva el diseño e implementación de un prototipo de una base de datos nacional ambiental que proporcione metadatos bien documentados. Este prototipo, se propone conjuntamente con el proyecto VLIR-TEAM “Mejorando el valor social de la economía circular en Latinoamérica” y con el apoyo de un desarrollador y experto en el análisis ambiental, e investigador asociado Brandon Kuczenski, Ph.D., como soporte de la Universidad de California, para trabajar en conjunto. Con esta iniciativa en marcha, se sigue el concepto propuesto en el informe de Orientación Shonan PNUMA (2011) que define su importancia dentro del análisis de ciclo de vida y, por lo tanto, los vuelve esenciales para comprender y evaluar la calidad de los datos. Un tema muy amplio es la calidad de los datos, por lo tanto, es importante encontrar un enfoque adecuado y abordar aquellos aspectos que son más importantes para el conjunto de datos (datasets), generando consultas personalizadas sobre los metadatos para crear conexiones entre los datasets.

El prototipo propuesto debe considerar criterios de calidad, accesibilidad y formatos de intercambio de datos que garanticen la interoperabilidad (Wolf, 2014). Entre los formatos de intercambio pueden ser: ILCD (sistema internacional de datos del ciclo de vida); en formato JSON; o finalmente, el tipo de datos .spold. que se implementa como formato de importación o exportación de datos LCA. Las guías internacionales de LCA que cumplen dichos criterios de interoperabilidad, requieren la acción de todos los actores sociales, incluidos los gobiernos, la industria y los consumidores (Azapagic, 2017), brindando el apoyo necesario para estructurar los datos de LCA.

Teniendo en cuenta los requerimientos de una base de datos ambiental mencionados en la Sección 1.1, este trabajo de titulación está enmarcado en una metodología de desarrollo de software incremental para el desarrollo de un prototipo de base de datos ambiental. En particular, a través de este proceso de desarrollo es que se asegurará el modelado de banco datos para los LCI (Aurea, 2017), teniendo acceso a administrar la base de datos. El desarrollo deberá contemplar el uso de sistemas informáticos para almacenar el contenido, que

permita la creación y el mantenimiento de los datos, así como la búsqueda y formas de acceso pertinente. Esta propuesta de prototipo pone a disposición de los usuarios los conjuntos de datos directamente en línea a través de un sistema basado en la web, como un navegador o, si el usuario lo prefiere, a través de una interfaz de programación de aplicaciones (API). Independientemente, el sistema de alojamiento de la base de datos va a ser fiable, seguro y garantizará la integridad del contenido del mismo, a la vez que se le brindará un fácil acceso y utilización a los usuarios. Dentro de la infraestructura informática se coloca una plataforma de intercambio. Esta actúa en base a un archivo comprimido con todos los conjuntos de datos en un formato determinado, para su posterior proceso de ETL (extracción, transformación y carga). Además, la infraestructura ofrece varias características, como la gestión de usuarios, un motor de búsqueda, la visualización de los datos, y la descarga selectiva.

1.4. Objetivos.

Objetivo General

Desarrollar un prototipo informático de soporte para gestionar conjuntos de datos de inventarios de ciclo de vida (LCI) representativos del contexto ecuatoriano.

Objetivos específicos

- Realizar el levantamiento de la información necesaria para la identificación de la problemática actual en cuanto a la generación de datos de inventario de ciclo de vida (LCI) en el Ecuador.

- Analizar la información recopilada, para la definición de los requerimientos del sistema que se desea implementar.

- Diseñar e implementar el prototipo informático de gestión de conjuntos de datos de LCI teniendo en cuenta los requerimientos obtenidos.

- Probar el prototipo informático con un caso de estudio práctico.

1.5. Metodología de la investigación.

Para el desarrollo de este análisis de prototipo informático, se ha seguido con el uso de la metodología estructurada conforme al modelo de transferencia tecnológica (Gorschek et al., 2006). Con esta metodología, el autor plantea ocho actividades, con las cuales se encuentra una solución realista por medio de un proceso iterativo de validación empírica sobre soluciones candidatas. El proceso se detallan a continuación:

1. Análisis del problema: Comprender el problema que da origen y propósito a la investigación, para lo cual se observa el enfoque del prototipo y se identifican las necesidades de los interesados.

2. Formulación del problema: Luego de analizar el problema, se lo formula de forma clara y precisa, incluyendo factores de contexto, objetivos que la investigación persigue, planteamiento de preguntas de investigación y justificación del estudio.

3. Revisión del estado del arte: Revisión sistemática de literatura que esclarece el estado del arte, revisando soluciones existentes e identificando brechas que la investigación desea abordar.

4. Solución candidata: Proponer una solución al problema establecido, por medio de un método definido.

5. Entrenamiento: Proporcionar el conocimiento necesario a los profesionales del área para generar una vista general de la aplicación de la solución propuesta.

6. Validación inicial: Se utiliza un caso de estudio práctico.

7. Validación realista: Se aplica un entorno real para llevar a cabo casos de estudio.

8. Liberación de la solución: Valoración de los resultados obtenidos y se preparan las herramientas y material requerido para su uso.

1.6. Estructura del trabajo.

A continuación, se presenta una breve descripción del contenido de cada capítulo que está presente en el trabajo de titulación.

- Capítulo 1: Introducción.

Acerca de este capítulo, que está relacionado con el análisis y la formulación del problema, se detalla la motivación y el contexto, la problemática actual, la

solución propuesta, y los objetivos tanto generales como específicos que se buscan alcanzar con el desarrollo del trabajo de titulación.

- Capítulo 2: Marco conceptual.

Expone los términos y conceptos correspondientes a los temas principales sobre las diferentes métricas y procesos que hacen posible la estructuración de datos ambientales.

- Capítulo 3: Marco tecnológico.

Este capítulo se enfoca en temas técnicos relacionados con el análisis, compilación y estructuración del trabajo de titulación. Los requerimientos de software que son necesarios para su construcción y el planteamiento de la metodología de desarrollo que se emplea durante el desarrollo del trabajo de titulación.

- Capítulo 4: Desarrollo de la propuesta.

El capítulo describe la puesta en práctica de la teoría establecida en el Capítulo 2, empleando las técnicas y métodos de investigación definidos previamente en el Capítulo 3. Para esto, se aplica una enfoque centrado en el proceso de modelado óptimo con referencia al desarrollo del prototipo e implementando la metodología de desarrollo más adecuada según los datos con los que se va a trabajar.

- Capítulo 5: Resultados y discusión.

Este capítulo presenta las pruebas realizadas con el prototipo informático desarrollado, dando lugar a una evaluación sistemática. Además, se comprueba que se cumpla el alcance y el propósito del trabajo de titulación en base a la interacción del usuario con el prototipo. Cabe señalar que en este capítulo se proponen posibles líneas de trabajo futuro.

- Capítulo 6: Conclusiones.

El último capítulo expone la información más relevante obtenida referente al cumplimiento de los objetivos planteados en el Capítulo 1 y en base al desarrollo del trabajo de titulación se describen los principales hallazgos y resultados obtenidos para futuras referencias.

- Anexos.

Este apartado presenta la documentación complementaria generada durante el desarrollo del trabajo de titulación, tales como, el manual de usuario y el manual técnico.

2. Marco Conceptual:

El análisis de ciclo de vida (LCA) es un área de estudio que requiere de conocimientos diversos para ser reunidos e interpretados juntos debido a que se desenvuelve en un campo altamente interdisciplinario (Kuczenski et al., 2016). Considerando esto, en los siguientes apartados se describen conceptos y métodos fundamentales para el desarrollo teórico del presente proyecto.

2.1. Iniciativa ambiental.

Originalmente, el Programa de las Naciones Unidas para el Medio Ambiente (PNUMA) y la Sociedad de Toxicología y Química Ambiental (Society of Environmental Toxicology and Chemistry, SETAC) estuvieron colaborando dentro de la iniciativa para el ciclo de vida PNUMA (2011). Esto ha llevado a que 23 países tengan un encuentro para desarrollar una guía global sobre los principios del PNUMA para establecer, dirigir y publicar conjuntos de información a través de las bases de datos. La finalidad de esta reunión fue apoyar a la gestión de los productos y servicios elaborados a nivel mundial (Kusche et al., 2015) en términos de LCA. Al trabajar con productos y servicios brindados por diferentes compañías geográficamente diversas, quiere decir que, las fuentes de recursos, las operaciones de fabricación y montaje, el uso de productos y/o servicios y la disposición final, brindan información que cuantifica de manera eficaz y constante el análisis del consumo de recursos y las emisiones de esas actividades.

El objetivo de proporcionar una base científica que sea sólida para la administración de los múltiples productos empresariales y de industria ayuda a avanzar en la sostenibilidad de los productos y las actividades económicas de la sociedad. Además, se presentan políticas consistentes basadas en el ciclo de vida de los gobiernos y así abordar la necesidad de orientación global sobre la recopilación y el procesamiento de datos LCI en bases de datos, enfocadas a un uso generalizado (Sonnemann et al., 2011).

En el marco del avance propuesto por la PNUMA, conjuntamente con la SETAC, se definieron seis ejes temáticos para el desarrollo y acceso al conjunto de datos generado sobre la metodología y conducta de la evaluación del ciclo de vida en términos de sostenibilidad ambiental. Dichos tópicos se fundamentan en la gestión de LCI a través de un enfoque adaptativo en la unificación, agregación, revisión y validación del conjunto de datos en beneficio de obtener de manera eficiente un mayor volúmen de información ambiental, mientras se mantiene la calidad y la supervisión (Sonnemann et al., 2011).

Por otro lado, se tiene constancia que, en el año 2015, los jefes de gobiernos y los altos representantes toman la decisión histórica ante el mundo, de presentar una agenda internacional que abarque los nuevos objetivos mundiales de desarrollo sostenible (United Nations, 2015). Dicha agenda se compromete en ser plenamente implementada hasta el año 2030, comprendiendo que, para la creación de políticas sobre sostenibilidad, se presentan retos de carácter ambiental, económico y social, que deben ser analizados a partir de los impactos directos de la sostenibilidad por medio de LCA y el Costeo del ciclo de vida o Life Cycle Costing (LCC) (Fauzi et al., 2019).

2.2. Análisis de ciclo de vida.

El análisis del ciclo de vida (LCA) o también conocido como *"análisis de la cuna a la tumba"* es un método científico, organizado, completo y estandarizado en varios países, que tiene la finalidad de examinar los impactos del ciclo de vida desde la extracción de la materia prima (cuna) hasta la eliminación (tumba) (Wolf & Chomkhamsri, 2012). La *Fig. 1* muestra las cuatro etapas que conforman un estudio de LCA según las directrices de la Norma ISO 14040. Primero, durante la etapa de *Definición de objetivo y alcance* se describen los parámetros LCA de manera cualitativa y cuantitativa, declarando el objetivo, estableciendo el contexto del estudio, y explicando cómo y a quién se comunicarán los resultados. Luego, la etapa de *Análisis de inventario (LCI)* implica la creación de un inventario de los flujos desde y hacia la naturaleza. La tercera etapa, *Evaluación de impacto*, es el resultado de los posibles impactos ambientales y en la salud humana provenientes de la segunda etapa. Finalmente, en la etapa de *Interpretación* se efectúa la identificación, cuantificación, verificación y evaluación de la información de los resultados del LCI y la evaluación del impacto del ciclo de vida. El resultado de la fase de interpretación es un conjunto de conclusiones y recomendaciones del estudio. Las fases son independientes por lo que no se considera finalizada cada etapa sino hasta que culmine todo el estudio.

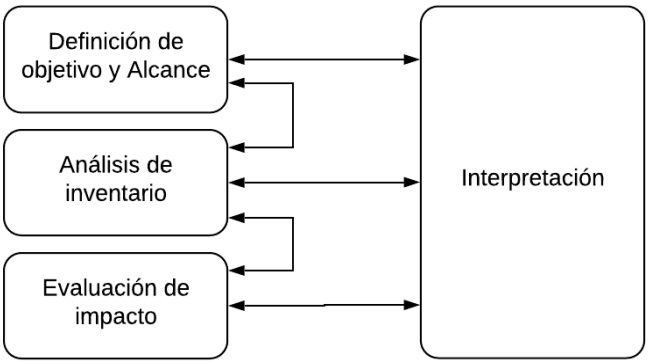


Fig. 1: Ilustración de las fases generales de una evaluación del ciclo de vida, según lo descrito por la Norma ISO 14040 (ISO 14044, 2006).

Concretamente, se conoce que LCA cubre diferentes temas relacionados con el cambio climático, el smog de verano, la ecotoxicidad, los efectos del cáncer humano y el agotamiento de los recursos materiales y energéticos. Dicho de otra manera, este método permite realizar comparaciones directas entre productos, tecnologías y servicios, basado en el rendimiento funcional cuantitativo de las alternativas analizadas (Ciroth et al., 2015). LCA captura el ciclo de vida completo del sistema que se analiza. La *Fig. 2* describe las fases involucradas en el ciclo de vida de un producto iniciando con la fase de extracción de recursos primarios, pasando por las fases de producción, uso y reciclaje, hasta llegar a la eliminación de los residuos restantes (Wolf, 2014). Indiscutiblemente, LCA es una técnica que permite evaluar los impactos ambientales asociados con todas las etapas de la vida de un producto pero, una vez que se recopilan las diferentes entradas y salidas, la información generada debe ser transformada en un modelo matemático para su análisis (tercera etapa) (Muthu, 2020). Estos efectos luego se cuantifican en el LCA y se relacionan con los resultados. La información recopilada y mencionada se utiliza para ayudar a los responsables de la toma de decisiones a seleccionar productos o procesos que tengan el menor impacto en el medio ambiente, al considerar un sistema de producto completo y así evitar la suboptimización que podría ocurrir si solo se usara un solo proceso (Corporación Internacional de Aplicaciones Científicas, 2006).

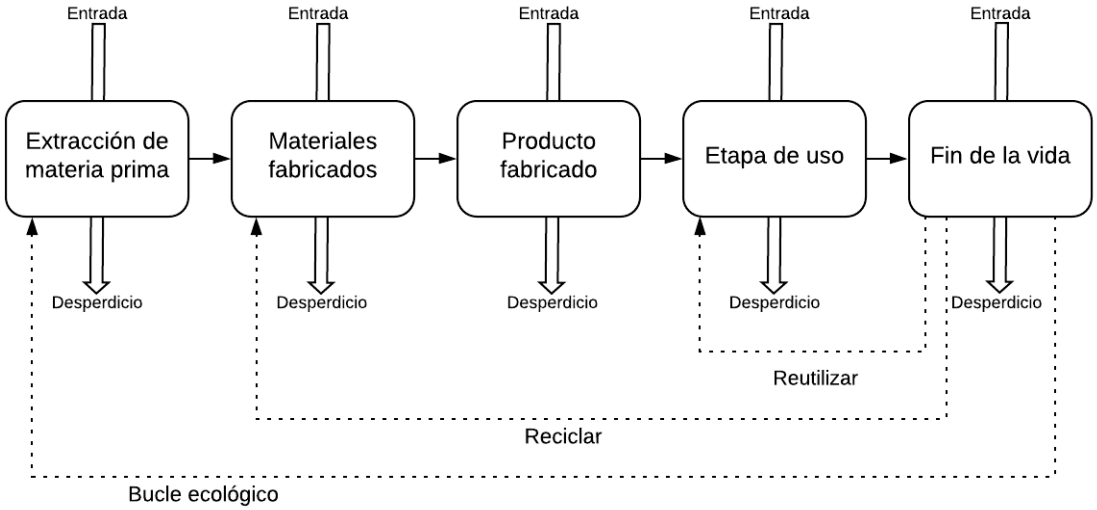


Fig. 2: Diagrama ejemplo que describe las fases más importantes de un LCA. (Muthu, 2020).

En particular, tener al alcance datos que sean interoperables y de buena calidad es una necesidad para cualquier instrumento de análisis. Sin embargo, en la mayoría de casos, estos datos son limitados, descontinuados o inexistentes (Miah et al., 2018). Indiscutiblemente, LCA tiene una curva de crecimiento muy notable dentro de los gobiernos de todo el mundo. Los gobiernos que ya la aplican activamente se encuentran buscando y actualizando información sobre el ciclo de vida para estructurar sus políticas de manera que estén encaminados a las diferentes problemáticas ambientales (Reed, 2012). Es evidente que la conciencia hacia la importancia de la protección ambiental, y los posibles impactos asociados con los productos, tanto manufacturados como extendidos, ha incrementado el interés por el desarrollo de métodos para comprender y tratar de manera más eficiente esos impactos (ISO 14044, 2006).

2.3. Inventarios de ciclo de vida.

El Inventario del Ciclo de Vida (LCI), también denominado como “conjunto de datos de proceso" es a menudo utilizado como sinónimo de "LCA data". LCI es información cuantitativa orientada a almacenar flujos elementales relacionados con cada proceso de un producto. Es decir, es el proceso de cuantificar las materias primas y los requisitos energéticos, las emisiones atmosféricas, las emisiones a la tierra, las emisiones al agua, el uso de recursos y otras liberaciones durante el ciclo de vida de un producto o proceso. LCI permite la creación, adición, mantenimiento y búsqueda sobre los recursos de datos individuales consumidos y las emisiones que se generan para un proceso o producto definido (Klöpffer,

2008). Con esto puede cubrir un solo paso del proceso, partes de un ciclo de vida o incluso todo el ciclo de vida.

En concreto, una base de datos de LCA es una colección organizada de conjuntos de datos de LCI que cumplen con las normas ISO 14040 y 14044 que, de manera similar a LCA, se ajustan suficientemente a un conjunto de criterios. Estos criterios incluyen una metodología consistente, validación o revisión, formato intercambiable, la documentación y la nomenclatura, que permiten la interconexión de conjuntos de datos individuales (Sonnemann et al., 2011). De tal que manera que, las bases de datos LCA al estar organizadas por una administración responsable, permiten identificar y rastrear las responsabilidades de la creación, contenido, mantenimiento y actualización de los datos. A los datos se los debe someter a un proceso de validación definido de forma interna para garantizar que cumplan con el protocolo de la base de datos (Vigon et al., 2017).

Para desarrollar el inventario, se comienza con un modelo de flujo del sistema técnico, recopilando datos sobre entradas y salidas del sistema de productos, como se muestra en el ejemplo de la *Fig. 3*. El diagrama de flujo incluye las actividades que se evaluarán en la cadena de suministro relevante y brinda una imagen clara de los límites técnicos del sistema. Cuanto más detallado es el diagrama, más preciso es el estudio y por lo tanto sus resultados (Corporación Internacional de Aplicaciones Científicas, 2006). De acuerdo con la ISO 14044, un LCI debe documentarse mediante los siguientes pasos: i) Preparación de la recopilación de datos en función de la meta y el alcance. ii) Recopilación de datos iii) Validación de datos. iv) Asignación de datos. v) Relacionamiento de datos con el proceso unitario. vi) Relacionamiento de datos con la unidad funcional. Y, finalmente vii) Agregación de datos (Matthews et al., 2014).

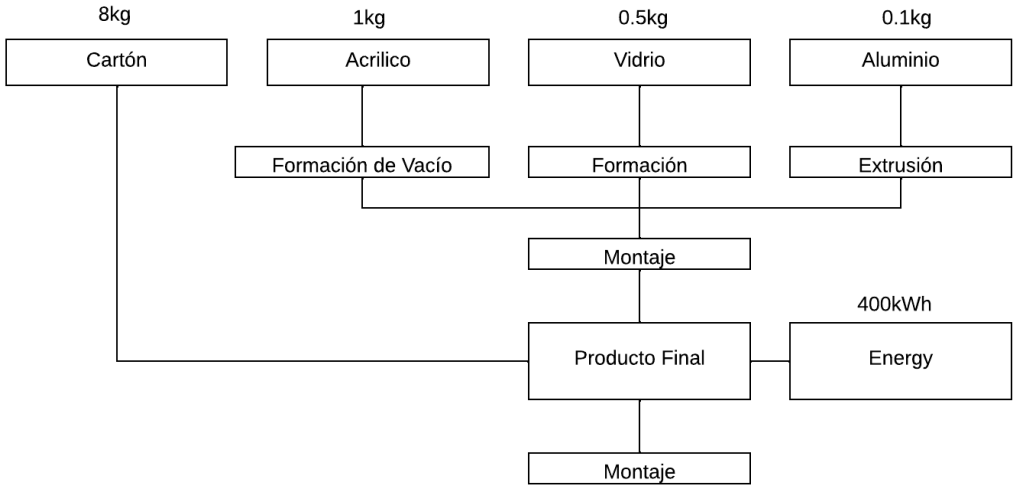


Fig. 3: Ejemplo de diagrama de Inventario del ciclo de vida (Corporación Internacional de Aplicaciones Científicas, 2006).

En cuanto a la gestión que se brinda a las bases de datos LCI, al recopilar la información para cada proceso dentro de los límites del sistema, LCA requiere que el estudio obtenga mediciones o estimaciones para representar cuantitativamente el desarrollo en el sistema del producto. Al identificar las entradas y salidas para documentar el procesamiento unitario dentro del sistema de productos de un LCI, un profesional puede encontrarse con la instancia en la que un proceso tiene múltiples flujos de entrada o generar múltiples flujos de salida. Por esta razón, durante las últimas dos décadas, las bases de datos para LCI han sido desarrolladas, mantenidas y actualizadas por profesionales en el área, tales como: i) proveedores de bases de datos generales, ii) proveedores de bases de datos del sector industrial, y iii) grupos internos de la industria con un alto nivel de profesionalismo. Dichos profesionales, según el documento de la PNUMA (2011), han seguido ciertas directrices para respaldar los datos en las bases de datos de LCI, entre ellos:

- Responsabilidad.

Implica una responsabilidad continua por las referencias proporcionadas. Se recomienda que se designe a un individuo para abordar la información en la base de datos en caso de cualquier problema con la comunicación hacia y desde el cliente, de lo contrario, no se respaldará el uso responsable de los datos de LCI.

- Soporte técnico y metodológico.

Las consultas de los clientes sobre los datos requieren respuestas rápidas. Por lo tanto, se recomienda que en un tiempo razonable se brinde asistencia a los usuarios al tener algún problema mientras se utiliza el servicio.

- Rutinas para un mantenimiento y una actualización consistente.

Rutinas estándar para brindar mantenimiento y extender actualizaciones de forma constante. Este proceso tiene la finalidad de guiar al usuario a través de una actualización y así evitar errores en un procedimiento de actualización.

- Documentación conforme.

Tener al alcance documentación adecuada que se ajuste a los principios globales de las LCA para el uso de las bases de datos, ocasionando responsabilidad entre el proveedor y el cliente para evitar que se sienta engañado.

- Equilibrio entre continuidad e innovación.

Actualizaciones frecuentes pero de contenido que sea relevante, debido a que no existe una posición exacta sobre cuál debería ser la frecuencia de las actualizaciones. La frecuencia de las actualizaciones debe juzgarse en función de la relevancia y la importancia de los cambios.

- Nivel de transparencia.

Los proveedores deben ofrecer conjuntos de datos con un nivel de transparencia apropiado para el usuario, respetando los requisitos de confidencialidad de los conjuntos de datos que contienen contenido clasificado y que deben estar de acuerdo con los principios globales de LCA.

- Almacenamiento seguro.

Tomar las medidas adecuadas para evitar la pérdida involuntaria o la distribución accidental del contenido interno de los datos.

- Armonización entre nuevos enfoques y conjuntos de datos, con los enfoques y conjuntos de datos ya existentes.

Al momento que se amplían las bases de datos, se recomienda que la base de datos existente se sincronice con el contenido recién proporcionado. Se debe tomar en cuenta que los administradores de bases de datos deben tomar todas las medidas necesarias para facilitar la armonización.

- Interfaces comúnmente disponibles para el intercambio de datos.

Para una correcta comunicación e intercambio de datos, el contenido de la base de datos debe estar construida de forma que sea adecuada para el intercambio, a través de interfaces estándar en otro software o sistemas LCA. Dicho eso, es necesario tener en cuenta que, primero se necesita armonizar los contenidos de la base de datos para evitar malentendidos, malas interpretaciones e inconsistencias no intencionadas.

Entre los formatos de datos más empleados de actualidad, se tienen los siguientes: ILCD, proporciona orientación, coherencia y garantía en la aplicación de LCA, el formato JSON que adapta los datos que contiene a una estructura simple y con todo el análisis en un mismo archivo; o finalmente, ecoSpold 2 o .spold. Es capaz de sobrescribir al formato .xml, por lo tanto, los procesos unitarios, el LCI y los resultados de la evaluación de impacto de cualquier actividad se pueden ver en cualquier editor xml. Así mismo, los datos de ecoSpold 2 se resumen en diferentes archivos para facilitar la legibilidad y el entendimiento .

2.4. Estudios previos.

En los últimos 15 años, un número considerable de países en todo el mundo ha desarrollado bases de datos nacionales basadas en LCA (Wolf & Chomkhamsri, 2012), haciendo énfasis en el análisis de la eficiencia, el consumo de la energía y sus fuentes, el consumo de materias primas y, en la disposición final de los residuos generados. De manera puntual, las bases de datos LCA deben demostrar el enfoque de los datos adoptados, los mecanismos de financiación y la gestión de mediante qué métodos de LCI fueron obtenidos. Esto con la finalidad de plantear un correcto camino hacia la interoperabilidad global de los datos, sin importar que se encuentre en una etapa temprana de desarrollo. Otro rasgo relevante en el auge de la compilación de datos es la generación de un esquema acorde con la certificación de los productos o servicios PNUMA (2011).

En particular, el análisis de modelos de gestión para unificar procesos de sostenibilidad ambiental es relevante para lograr minimizar el impacto regional a través de un sistema de evaluación estandarizado. En el marco de este proceso, Tailandia es un ejemplo adecuado para analizar, debido a la longevidad del desarrollo de su proyecto (desde 2006). Su ejecución estuvo restringida por i) la falta de conciencia de las partes interesadas en su financiación, utilización y gestión de políticas públicas y ii) la ausencia de experiencia de los especialistas locales en LCA (Chomkhamsri et al., 2017). En definitiva, la base de datos nacional tailandesa de LCI se ha utilizado como una de las infraestructuras clave de la nación para apoyar la gestión pública y las aplicaciones relacionadas con el crecimiento verde.

A nivel regional se han implementado soluciones enfocadas en el desarrollo sostenible de la industria y en el posicionamiento de los productos latinoamericanos. De esta manera, en Brasil, un equipo de trabajo formado por profesionales cualificados e investigadores de alto nivel académico científico desarrollaron ACV Brasil (2020). Esta solución ha ayudado a las industrias, empresas e individuos a visualizar, evaluar y medir los impactos ambientales y socioeconómicos a lo largo de su cadena de producción. De esta forma, se afirma que fue posible fomentar y promover la integración de herramientas de gestión a través de iniciativas públicas y privadas. Por otra parte, la base de datos del ciclo de vida nacional peruana surge como un recurso vigente para generar beneficios políticos y corporativos. La aplicación web peruana, PERULCA (2019), organiza el ingreso de datos brutos y conjuntos de datos, brindando información y análisis para la optimización de recursos. Esta estructura de información contiene cuatro sectores estratégicos referentes con: i) vertederos, ii) centrales hidroeléctricas, iii) productos de refinería, y iv) sectores de la pesca y la acuicultura (Vázquez-Rowe et al., 2019). En definitiva, se puede evidenciar una relevante vinculación a

diferentes sectores de la economía peruana que introduce los puntos a tener en cuenta para una propuesta nacional.

Considerando el caso particular de Ecuador, y tomando en cuenta que en el año 2018 se ratificó la asimilación de la Agenda 2030 como parte de la política pública, se espera que el país analice los diferentes Objetivos de Desarrollo Sostenible (ODS) en la planificación estadística nacional para que la vincule con la agenda global. Sin embargo, los datos con los que se cuentan a partir de este proceso, correspondientes a LCI, son de energía hidroeléctrica y termoeléctrica. Por lo tanto, se puede decir que, para la eficiencia energética, producción y consumo sostenible, así como para la gestión integral de residuos (Arias et al., 2019), no se cuenta con los datos en el formato, ni con los requerimientos adecuados para estructurar una base de datos nacional.

De este modo, para la definición de una estructura de información en el contexto ecuatoriano es necesario tomar en cuenta que los especialistas internacionales en sostenibilidad ambiental realizan los estudios de LCA en base a la información provista por sistemas informáticos que leen y evalúan críticamente información ambiental. Estos sistemas están diseñados e implementados por desarrolladores de software internacionales, que a su vez no han determinado un estándar para publicar los resultados de LCA, lo cual conlleva una falencia en común definida como con una baja confidencialidad de interpretación de los datos (Hauschild et al., 2018). La comunidad de LCA proporciona una variedad de herramientas de software que se pueden utilizar para diseñar modelos LCI, además de realizar los cálculos de LCA PNUMA (2011). Entre los sistemas de código abierto disponibles en la Web se encuentran OpenLCA, Brightway2, Antelope, Amaru, entre otros (Liebsch, 2020), los cuales no han implementado una gestión consistente para el almacenamiento de la información debido a que los sistemas mencionados utilizan o Lenguaje de Marcado Extensible (XML) (Ciroth, 2007), Notación de Objetos JavaScript (JSON), o incluso ambos formatos en el proceso de extracción de los datos pertinentes de LCI (Mutel, 2017). La problemática relacionada con la falta de estandarización en cuanto a un formato único de datos, limita la capacidad de los profesionales para difundir sus resultados con un alto impacto en la comunidad (Ciroth, 2019).

En efecto, para el trabajo de titulación, y con fines de guía e implementación, se analizan los tres sistemas mencionados anteriormente. OpenLCA que es un software para evaluación de ciclo de vida y sostenibilidad ambiental, tiene características como información detallada sobre los resultados o capacidades de importación y exportación (*OpenLCA*, 2016). Al mismo tiempo Brightway2 que al ser un marco de código abierto para LCA. No intenta reemplazar otros software como SimaPro u OpenLCA, sino ofrecer nuevas posibilidades a aquellos que

quieren romper los límites del LCA convencional construido en base al lenguaje de programación Python (Mutel, 2016). También conociendo Antelope que es una interfaz estándar y un marco de referencia para LCA construída en el lenguaje Python, brinda una variedad de información, y casos de uso en el análisis de LCA (Kuczenski, 2020). Aquí se menciona que el propósito de Antelope es identificar datos de acuerdo con una especificación consistente y desarrollar herramientas que hagan uso de esa especificación para simplificar las tareas de publicación y acceso a los datos. Así mismo es importante mencionar a AMARU, que textualmente es: “una iniciativa de conocimiento abierto impulsada por la Red Ecuatoriana de Ciclo de Vida y Economía Circular que tiene como objetivo difundir el pensamiento y los enfoques de ciclo de vida” (*Amaru*, 2022). Estos se convierten en herramientas fundamentales para comprender el alcance del proyecto de titulación y lo que se puede lograr con la implementación de la base de datos para LCA. Tomando en cuenta el hecho que para analizar estos software se debe adquirir una base en Python para usarlos. Por lo tanto, y gracias a la comunicación que se pudo generar, se trabaja en conjunto con Antilope y su desarrollador para generar un prototipo informático para LCA.

2.5. Conclusiones del análisis conceptual.

En este capítulo se presentaron políticas consistentes basadas en el ciclo de vida de los productos y servicios. Se abordó la necesidad de orientación global sobre la recopilación y el procesamiento de información en bases de datos, enfocadas a un uso generalizado. Se detalla que el LCA es un método científico, regulado, completo y estandarizado en muchos países, destinado a examinar los efectos del ciclo de vida desde la extracción de la materia prima hasta su eliminación.

Existen software específicos, tanto comerciales como de código abierto que cumplen con la finalidad de una base de datos LCA, que es generar una agrupación estructurada de conjuntos de datos LCI. Con esto, el LCI es información cuantitativa que tiene como propósito almacenar flujos básicos, relacionados con el proceso de cada producto, es decir, el proceso de determinar la cantidad de materias primas y requisitos de energía, emisiones a la atmósfera, emisiones al suelo, emisiones al agua, uso de recursos y otras emisiones a lo largo de la vida del producto o proceso. PNUMA sugiere directrices para respaldar los datos en las bases de datos de LCI, que son: responsabilidad, soporte técnico y metodológico, rutinas para un mantenimiento y una actualización consistente, documentación conforme, equilibrio entre continuidad e innovación, nivel de transparencia, almacenamiento seguro, armonización entre nuevos enfoques y

conjuntos de datos ya existentes, y finalmente interfaces disponibles para el intercambio de datos.

Básicamente, con esto y con fundamentos obtenidos a partir del análisis de demás software, como los mencionados que permiten la interacción de las bases de datos LCA y brindan funcionalidad, se puede construir, implementar y desplegar este proyecto de código abierto que busca ser un agregado a los demás software y no reemplazarlos, debido a que se encuentra en la intersección de LCA y el querer brindar información personalizada a los usuarios con referencia a la metainformación.

3. Marco Tecnológico:

El presente capítulo está dedicado a mostrar algunos aspectos generales, conceptos concretos e ideas importantes para comprender el dominio de la aplicación y principalmente las tecnologías que se emplean para el desarrollo de los objetivos del trabajo de titulación. En los siguientes apartados se describen atributos fundamentales para la comprensión e implementación tecnológica del proyecto, tales como la arquitectura de software a seguir, la base de datos de preferencia, las herramientas de desarrollo y la metodología empleada para el desarrollo de la propuesta.

3.1. Bases de datos.

Las bases de datos son el método preferido para almacenar datos (Camps Paré et al., 2005), éste término como tal, apareció en 1963. Dentro de la informática, una base de datos consiste en un conjunto de datos interrelacionados y un conjunto de programas que acceden a dicha información. Según Gómez Fuentes (2013), no es más que un informe relacionado agrupado que se utiliza para asegurar la integridad de la información y a su vez facilitar el acceso a los datos.

Las bases de datos se clasifican en diferentes grupos según los tipos de datos que quiera almacenar y la tecnología empleada en su funcionamiento. Las más antiguas son las jerárquicas y en red, las más avanzadas son las orientadas a objetos, pero las más utilizadas son las bases de datos relacionales y no relacionales (Galvez, 2017). Las bases de datos relacionales son ampliamente utilizadas en software profesional, educativo y comercial. Éstas permiten establecer relaciones entre los datos que están almacenados en sus tablas, y conectar los datos de ambas tablas, donde cada elemento puede ser identificado de forma inequívoca. Tras ser postulada en 1970, no tardó en consolidarse como un nuevo paradigma. Tomando en cuenta distintos factores, como el hecho que las bases de datos relacionales son fiables debido a que si un proceso tiene algún error, no se lleva a cabo, o que su capacidad de garantizar la integridad de los datos es muy buena, la convierten en una buena opción para elegir este tipo de base en un proyecto.

Existe un sistema dedicado exclusivamente a manejar bases de datos relacionales, para mantener control centralizado tanto de los datos como de los programas que tienen acceso a dichos datos (Gómez Fuentes, 2013). Este conjunto de programas se conoce como SGBD (Sistema Gestor de Base de Datos). Entre los gestores o manejadores actuales más populares se encuentran: Oracle Database, MySQL, PostgreSQL, entre otros. Oracle Database es una

tecnología privada que ofrece a los clientes versiones de alto rendimiento y costo optimizado, a más de un buen rendimiento, confiabilidad, escalabilidad, y seguridad (Oracle, 2021). Se lo considera líder en el mercado. Por otra parte, MySQL es un sistema de administración de bases de datos relacionales con todas las funcionalidades de Oracle DB, debido a que es propiedad de Oracle. MySQL difiere de Oracle DB porque es de dominio público. Los desarrolladores pueden usar este SGBD con una licencia pública (Moore, 2018). Muchas de las organizaciones con rápido crecimiento del mundo confían en MySQL para economizar tiempo y dinero al momento de alimentar sus sistemas de gran volumen de datos (MySQL, 2021). Finalmente, se tiene a PostgreSQL que es un potente sistema de base de datos relacional con un desarrollo activo que le ha valido una sólida reputación de confiabilidad, rendimiento y robustez de características (PostgreSQL, 2021a). PostgreSQL tiene varias propiedades enfocadas a ayudar a los desarrolladores a crear aplicaciones para proteger la integridad de los datos y crear entornos tolerantes a fallas, ayudando a administrar dichos datos sin importar cuán grande o pequeño sea el conjunto de datos. PostgreSQL es altamente extensible además de ser de código abierto y libre pues no es dependiente de ninguna empresa privada. Por lo tanto, este gestor (PostgreSQL, 2021b) se vuelve una buena opción al momento de decidir qué SGBD escoger para emplear en un proyecto.

3.2. Arquitectura de Software.

Durante el desarrollo de un sistema informático es imprescindible el uso de algún patrón arquitectónico, que pueda ser reutilizable ante problemas similares y que permita crear soluciones abstractas. Actualmente, existen diferentes tipos de arquitecturas. El más utilizado es el *Modelo Vista Controlador*, el cual es muy utilizado en APIs (Application Programming Interfaces, es decir, Interfaz de Programación de Aplicaciones) dentro del entorno de internet (Ccori, 2018). Este modelo, como se puede ver en la *Fig. 4,* es un patrón de desarrollo de software que divide los programas aplicativos en tres capas simples: i) Modelo: encargado de manejar los datos y gestionar las conexiones entre la base de datos y el interior de la aplicación. ii) Vista: encargado de manejar cómo se mostrará la información de la aplicación. Y, iii) Controlador: encargado de manejar la entrada del usuario y de dirigir sus acciones (Bodega Fernanz, 2019).

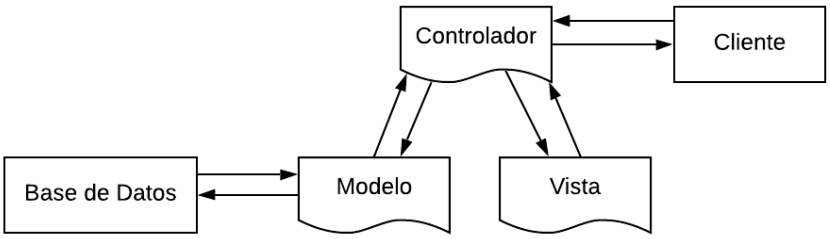


Fig. 4: Esquema de flujo del patrón de desarrollo Modelo Vista Controlador (Bodega Fernanz, 2019).

Al desarrollar aplicaciones separadas en diferentes capas que están encargadas cada una de una parte específica de la lógica de negocio de aplicación, se obtiene el siguiente flujo de despliegue: i) el usuario puede realizar una petición a la aplicación, que es recibida por el controlador, luego ii) el controlador se comunica con las capas de modelo y vista conjuntamente, pues se debe obtener toda la información necesaria y, finalmente, iii) el controlador devuelve la información seteada a como la vista lo requiera para mostrarlo al usuario.

3.3. Desarrollo web.

Dentro del desarrollo de una aplicación, y siguiendo un patrón de diseño como el mencionado anteriormente, es preferible implementar frameworks de trabajo para eliminar código redundante, aumentar la velocidad de desarrollo y aumentar las buenas prácticas de programación (Orix Systems, 2015). La utilización de un framework implica un costo en el aprendizaje, pero a largo plazo facilita tanto el desarrollo como el mantenimiento. Existen frameworks orientados a diferentes lenguajes o funcionalidades, entre los más utilizados están Symfony 4, Laravel, Django Rest Framework, entre otros. Symfony 4 es uno de los más utilizados y recomendados, ya que es una de las primeras opciones de muchos desarrolladores web. Este framework consta de una gran cantidad de componentes reutilizables, y también existe una comunidad activa que siempre lanza código nuevo para desarrollar posibles actualizaciones y mejoras (Tébar, 2020). Está basado en el lenguaje de programación de php. Por su parte, Laravel es uno de los frameworks más jóvenes que utiliza algunos componentes de Symfony (Tébar, 2020). Finalmente, Django Rest Framework está basado en el lenguaje de programación Python y permite manejar de una forma más fácil e intuitiva la información del aplicativo, además que es capaz de conectar con cualquier base de datos relacional (Bodega Fernanz, 2019). Este framework sigue el patrón de Modelo Vista Controlador de una forma bien establecida (Slant, 2021),

con lo cual proporciona características para construir una API web y no una aplicación web. Por lo tanto, el último mencionado, Django Rest Framework se vuelve una la mejor opción al momento de decidir qué framework escoger para implementar en este proyecto de titulación, además que su desarrollo con el lenguaje Python genera la conexión con los software de LCA analizados en los estudios previos de la sección 2.4, brindando la utilidad del mismo.

3.4. Interfaz de Programación de Aplicaciones.

La interfaz de programación de aplicaciones API es el medio por el cual un programa aplicativo accede a servicios de otros componentes de software, es decir, es un código que permite que dos programas de software se comuniquen (*Interfaces para los programas de aplicación (APIs)*, 2015). Cuando se usa en el contexto del desarrollo web, se define típicamente como un conjunto de mensajes de solicitud del Protocolo de transferencia de hipertexto (hypertext transfer protocol, HTTP) debido a que una API puede incluir especificaciones para rutinas, estructuras de datos, clases de objetos y variables (Creative Commons, 2021). Las APIs web permiten la combinación de múltiples servicios en nuevas aplicaciones basados en protocolos de comunicación más directa al estilo REST (REpresentational State Transfer, en español, transferencia de estado representacional), obteniendo una estructura de los mensajes de respuesta, que generalmente se encuentra en lenguaje de marcado (extensible markup language, XML) o en formato de notación de objeto de JavaScript (JavaScript object notation, JSON) (Creative Commons, 2021).

3.5. Arquitectura REST.

La transferencia de estado representacional, REST, es una de las arquitecturas más conocidas basadas en cliente-servidor que fue originada por Roy Fielding en el año 2000 como parte de su disertación doctoral (Jakl, 2005). Este estilo de arquitectura de software fue diseñado para sistemas distribuidos y para la internet porque sigue un lenguaje básico de HTTP, convirtiéndose en el estilo arquitectónico de software predominante y aceptado para la internet (Bodega Fernanz, 2019). Esta arquitectura sigue ciertas pautas sobre cómo se comporta una aplicación web bien diseñada, que involucra transiciones de estado, dando como resultado la página siguiente, que representa la siguiente etapa de la aplicación para el usuario, y se logra con los siguientes verbos: i) POST: agregar datos. ii) GET: recuperar datos sin alterarlos de una Uniform Resource Locator

(URL) en particular. iii) PUT: guardar una actualización. Y, finalmente, iv) DELETE: eliminar un recurso específico (REST, 2016).

3.6. Metodología de Desarrollo Incremental de Software.

La notación de modelado y las diferentes herramientas se han diseñado para ser la clave dentro del desarrollo de un software exitoso, pero si no se proporcionan instrucciones efectivas para su aplicación, entonces las buenas notaciones y herramientas son inútiles. Sería complicado modificar las reglas con respecto al desarrollo de software, por lo que se procede a adaptar una metodología de especificación y desarrollo que pueda obtener un resultado rápido, algo que se pueda ver, mostrar y sobre todo utilizar. Esta metodología ha de ser ágil, por lo que debe contar con ciclos cortos de desarrollo y debe incrementar las funcionalidades en cada iteración del mismo preservando las existentes, ayudando al desarrollo en lugar de darle la espalda. Es para este entorno que como consecuencia se creó el Manifiesto Ágil (Beck et al., 2001).

El Manifiesto Ágil es un modelo de mejora continua en el que el resultado debe ser planificado, creado, comprobado y mejorado cada vez. Algo que es constante y rápido, con plazos de entregas reducidos que busca evitar la dispersión y centrar toda la atención en una tarea encomendada. Tiene varias ventajas como i) mejorar la calidad, pues minimiza los errores en los entregables y mejora la experiencia y las funcionalidad para el cliente; ii) mayor compromiso, debido a que mejora la satisfacción del empleado y genera conciencia de equipo; iii) rapidez, gracias a que acorta los ciclos de producción y minimiza los tiempos de reacción y toma de decisiones; y, finalmente, iv) aumenta la productividad, ya que asigna mejor los recursos y de forma más dinámica (Tena, 2018).

3.6.1. Metodología del Desarrollo Scrum:

Aunque los creadores e impulsores de las metodologías ágiles más populares apoyan el manifiesto ágil y coinciden con los principios mencionados, cada metodología tiene características propias. Algunas de las metodologías ágiles más utilizadas en la actualidad son: Extreme Programming (XP), Test Drive Development (TDD), Agile Project Management (Scrum). XP es una metodología de desarrollo de software destinada a mejorar la calidad del software y la capacidad de respuesta a los requisitos cambiantes del cliente manteniendo comunicación constante con ellos. Opta por lanzamientos frecuentes en ciclos de desarrollo cortos, destinados a mejorar la productividad e introducir puntos de

control en los que se pueden adoptar nuevos requisitos del cliente (Wells, 2013). Por su parte TDD tiene un enfoque en el que se desarrollan casos de prueba para especificar y validar lo que hará el código, implementado una funcionalidad y probando primero y, si la prueba falla, se escribe el nuevo código para pasar la prueba y hacer que el código sea simple y sin errores (Hamilton, 2021). Finalmente Scrum es un método de gestión de proyectos que no solo es adecuado para el desarrollo de software, sino también para cualquier tipo de proyecto. Según Mariño & Alfonzo (2014), se define como una colección de procesos de gestión de proyectos, que permite concentrarse en el plan de mejora continua, es decir, brindar valor a los clientes y empoderar al equipo para lograr la máxima eficiencia. En este último se centra el trabajo de titulación y se hace hincapié en algunos aspectos más específicos.

Al ser una metodología de desarrollo ágil tiene como base la idea de creación de ciclos breves para el desarrollo, que comúnmente se llaman Sprints (iteraciones). Para comprender el ciclo de desarrollo de Scrum es necesario conocer las cinco fases que definen su ciclo de vida:

- Concepto: Se define de forma general las características del producto y se asigna el equipo que se encargará de su desarrollo.
- Especulación: Se hacen disposiciones con la información obtenida y se establecen los límites que marcarán el desarrollo del producto. Esta fase se repite en cada iteración y consiste en: i) desarrollar y revisar los requerimientos generales, ii) mantener la lista de funcionalidades que se esperan, y iii) establecer las fechas de entrega de las versiones.
- Exploración: Se incrementa el producto en el que se ha construido.
- Revisión: Se revisa todo lo construído.
- Cierre: Se entregará en la fecha acordada una versión del producto deseado.

Scrum gestiona estas iteraciones a través de reuniones, que es uno de los elementos fundamentales de esta metodología. Como se puede ver en la *Fig. 5*, Scrum es un marco iterativo e incremental que se enfoca en iteraciones de 1 a 4 semanas, una tras otra. Al comienzo de cada Sprint, el equipo multifuncional selecciona los requisitos del cliente de la lista de prioridades. Prometen completar el proyecto al final del Sprint. Durante el Sprint, el elemento seleccionado no se puede cambiar. Al final del Sprint, el equipo lo revisa con las personas interesadas en el proyecto y les muestra lo que han construido para su retroalimentación.

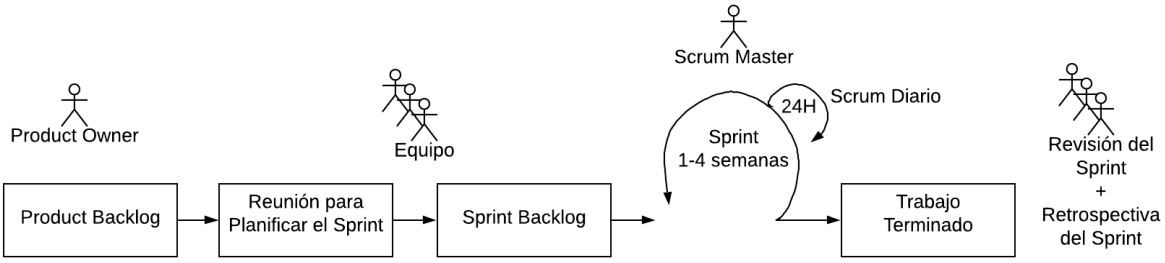


Fig. 5: Representación de procesos Scrum. (Cohn, 2009).

3.6.1.1. Componentes de Scrum:

Para entender el proceso de desarrollo Scrum, se describen de forma general los roles y las fases.

- Los Roles:

Los roles se subdividen en dos grupos. Las personas que están comprometidas con el proyecto como i) product Owner, ii) scrum Master y iii) equipo de Desarrollo; y el otro grupo son los que, aunque no son parte del proceso de desarrollo de Scrum, son parte del equipo de la retroalimentación para poder revisar cada Sprint, como i) usuarios, ii) stakeholders, y iii) managers.

  - Product Owner: Es la persona que toma las decisiones, y es la que realmente conoce el negocio del cliente y su visión del producto.
  - Scrum Master: Encargado de comprobar que el modelo y la metodología funcionan correctamente.
  - Equipo de Desarrollo: Equipo de personas para organizar y tomar decisiones para conseguir el objetivo.
  - Usuarios: Destinatario final del producto.
  - Stakeholders: Personas a las que el proyecto les producirá un beneficio y participan durante las revisiones del Sprint.
  - Managers: participan en las decisiones finales para comprobar los objetivos y los requerimientos.

- Las Fases:

Scrum se puede dividir en tres fases: i) planificación del Backlog, ii) seguimiento del Sprint, y iii) revisión del Sprint.

- ❖ Planificación del Backlog: Se definen los requisitos del sistema por prioridades. Se define el Sprint 0, en el cual se deciden cuáles serán los objetivos para la iteración. Además se define el Sprint Backlog que es una lista de tareas.
- ❖ Seguimiento del Sprint: Se llevan a cabo reuniones diarias para evaluar el avance de las tareas.
- ❖ Revisión del Sprint: Al finalizar el Sprint se realiza una revisión del incremento que se ha generado.

3.6.1.2. Elementos del Scrum:

Los elementos que forman parte del Scrum son: i) product Backlog, ii) sprint Backlog, e iii) incremento.

- Product Backlog: Lista de necesidades del cliente de forma priorizada. Estos requerimientos serán los que tendrá el producto. Esta lista será gestionada y creada por el cliente con la ayuda del Scrum Master. Una vez definidos los requerimientos se acuerda cuando se tiene que entender su objetivo como terminado o completado. Este elemento siempre evoluciona y tiene valor frente al cliente.
- Sprint Backlog: Lista de tareas que elabora el equipo durante la planificación de un Sprint, asignando las tareas a cada persona del equipo y el tiempo que queda en terminarlas.
- Incremento: Se presentan los requisitos que se han completado en una iteración y que son perfectamente operativos. Según los resultados obtenidos, el cliente puede hacer cambios y replantear el proyecto.

3.7. Conclusiones del análisis tecnológico.

En este capítulo se indica que las tecnologías a emplear en el trabajo de titulación. Efectivamente, las bases de datos son un apoyo importante en el método de almacenamiento de datos. Las bases de datos relacionales permiten establecer una asociación entre los datos almacenados en sus tablas y vincular los datos de las dos tablas, ya que cada elemento se puede identificar sin ambigüedades. Así mismo, la API web permite integrar una serie de servicios REST y obtener mensajes de respuesta estructurados en formato xml. Finalmente, el uso de la metodología Scrum para el desarrollo del prototipo informático, lleva a

conseguir un resultado que se pueda ver, mostrar y sobre todo utilizar. Contando con ciclos cortos de desarrollo e incrementando las funcionalidades en cada iteración del mismo preservando las implementaciones existentes, y ayudando al desarrollo en lugar de darle la espalda.

4. Desarrollo de la Propuesta:

El presente trabajo se centra en el desarrollo de un prototipo de una base de datos ambiental nacional de LCI, la cual sea accesible, confiable, de calidad y utilice los formatos de intercambio de datos que garanticen la interoperabilidad entre las distintas bases de datos a nivel mundial. Esta investigación se ha planteado con el fin de obtener un resultado sólido que pueda ser empleado a futuro para realizar evaluaciones ambientales bajo un enfoque de pensamiento de ciclo de vida. Esto con el propósito de facilitar la toma de decisiones y la creación de políticas públicas basadas en evidencia.

En este capítulo se describe la construcción del proyecto de titulación mediante la metodología de desarrollo Scrum, explicando las etapas que se siguieron antes y durante la implementación del proyecto. La finalidad de esta sección es presentar el marco de trabajo, así como los requerimientos y seguimiento de los avances del proyecto.

4.1. Planificación del Proyecto:

La ejecución de este proyecto se basó en la metodología de desarrollo Scrum, la cual comprende un proceso iterativo. Los equipos de personas se involucran sobre una secuencia de pasos a seguir que permite el desarrollo eficiente de software, y por ende entregar un producto de calidad. Para llevar a cabo una correcta planificación, se inició con la identificación de actores y se continuó con la planificación del product backlog.

4.1.1. Identificación de Actores:

En este paso, se identificaron los actores involucrados en el desarrollo del prototipo informático para comprender el caso de negocio y el valor del proyecto. Como se mencionó en la sección *3.6.1.1*, se tienen los siguientes roles, los cuales están cubiertos gracias al apoyo de diferentes actores que forman parte del equipo de investigación VLIR-TEAM de la Universidad de Cuenca. Además de Brandon Kuczenski de la Universidad de California, que actuó como soporte y guía durante el trabajo de titulación, brindando ideas y sugerencias sobre LCA y lo que un software LCA debe contener, gracias a su conocimiento sobre LCI.

- Product Owner: Brandon Kuczenski.
- Scrum Master: Brandon Kuczenski, Erik Sigcha.

- Equipo de Desarrollo: Kamila Molina, Mateo Quizhpi.
- Usuarios: Brandon Kuczenski & Comunidad científica con interés en información LCA.
- Stakeholders: Brandon Kuczenski.
- Managers: Lorena Sigüenza, Erik Sigcha.

4.1.2. Backlog del producto

Una vez definidos los actores, se procedió con la definición del caso de negocio y la visión del proyecto. El equipo del proyecto en conjunto creó el caso, en base al análisis de la documentación y de las necesidades encontradas. El proceso estuvo organizado en cuatro fases con la finalidad de enumerar y priorizar las tareas en el Backlog del Producto.

- Creación de la lista priorizada:

A continuación se detallan las tareas que se desarrollaron en el equipo técnico para las iteraciones. En la Tabla 1 se pueden observar las tareas definidas junto con sus prioridades de desarrollo, tiempo estimado de desarrollo en semanas y su respectivo actor responsable.

Tabla 1: Lista Priorizada.

ID	Prioridad	Descripción	Estimación (semanas)	Actor
1	Muy alta	Análisis de los formatos de LCI	2	Equipo de Desarrollo
2	Muy alta	Definición del tipo de datos a almacenar (metadata, flujos, intercambios)	2	Equipo de Desarrollo
3	Muy alta	Revisión de la estructura del formato y el tipo definido de LCI	2	Equipo de Desarrollo
4	Muy alta	Definición de elementos prioritarios para la estructura de la información a almacenar	1	Equipo de Desarrollo
5	Alta	Definición del modelo de administración de datos	1	Equipo de Desarrollo
6	Muy alta	Desarrollar el modelado de la base de datos	2	Equipo de Desarrollo
7	Alta	Definición de sistema gestor para la base de datos	1	Equipo de Desarrollo

ID	Prioridad	Descripción	Estimación (semanas)	Actor
8	Muy alta	Definición de las entidades y dependencias de la base datos	1	Equipo de Desarrollo
9	Muy alta	Proceso de creación de la base de datos en el sistema gestor	1	Kamila Molina
10	Muy alta	Desarrollo del método de lectura de los archivos LCA	1	Kamila Molina
11	Muy alta	Desarrollo del método de carga de archivos a la base de datos	2	Kamila Molina
12	Alta	Pruebas de funcionalidad e integridad sobre la base de datos	1	Kamila Molina
13	Alta	Definición de la tecnología y ambiente de desarrollo para la API	1	Equipo de Desarrollo
14	Muy alta	Definición del modelo de objetos de la base de datos	1	Mateo Quizhpi
15	Muy alta	Pruebas de funcionalidad, integridad y libre acceso a la información referente a la base de datos	1	Mateo Quizhpi
16	Muy alta	Desarrollo de consultas personalizadas para la API	1	Kamila Molina
17	Muy alta	Análisis de la estructura de los archivos de correspondencia de las versiones Ecospold	1	Equipo de Desarrollo
18	Muy alta	Definición de las variables de los archivos de correspondencia a ser almacenadas	1	Equipo de Desarrollo
19	Alta	Desarrollar el modelo de la base de datos	1	Kamila Molina
20	Alta	Definición de las entidades y dependencias de la base de datos	1	Kamila Molina
21	Alta	Proceso de creación de nuevas tablas en la base de datos	1	Kamila Molina
22	Muy alta	Desarrollo del método de lectura de los archivos de correspondencia	2	Kamila Molina
23	Muy alta	Desarrollo del método de carga de archivos de correspondencia a la base de datos	2	Kamila Molina
24	Muy alta	Desarrollo de las vistas para la API	2	Mateo Quizhpi
25	Media	Generación de documentación y mensajes personalizados de la API	1	Mateo Quizhpi

ID	Prioridad	Descripción	Estimación (semanas)	Actor
26	Muy alta	Administración del acceso a la información mediante la API	1	Mateo Quizhpi
27	Alta	Generación de reportes de la información mediante la API	1	Mateo Quizhpi
28	Alta	Desarrollo del inicio de sesión en el sistema.	1	Mateo Quizhpi
29	Alta	Creación de menú ítems en la interfaz de usuario.	1	Mateo Quizhpi
30	Alta	Creación de un menú ítem para las consultas personalizadas de la metadata.	1	Mateo Quizhpi
31	Alta	Creación de un menú ítem para el intercambio de versiones.	1	Mateo Quizhpi
32	Alta	Creación de un menú ítem para subir un archivo ecoinvent en formato .zip	1	Mateo Quizhpi
33	Alta	Creación de un menú ítem para subir un archivo de correspondencia	1	Mateo Quizhpi
34	Alta	Generación de documentación para la usabilidad del prototipo informático	2	Equipo de Desarrollo

- Creación de las historias de usuario con los criterios de aceptación

En este apartado se establecieron los requerimientos por parte del usuario, los cuales permitieron identificar las tareas de desarrollo para el prototipo informático. Las historias de usuario son la unidad funcional del Product Backlog. Existen varias estructuras de historias de usuario, pero para fines del proyecto se usa la siguiente estructura.

Número: Representado por un número o un valor alfanumérico único.

Nombre de la historia: Título descriptivo de la historia de usuario.

Prioridad: En la implementación de la historia de usuario, ésta es la prioridad asignada respecto al resto de las historias de usuario. Está considerado en tres rangos: Alta, media y baja, siempre y cuando las historias de usuario tengan registrados los criterios de aceptación (que se indicarán a continuación).

Descripción: Le dice al desarrollador lo que el cliente espera de ese requerimiento. Descripción sintetizada de la historia de usuario siguiendo la siguiente secuencia:

- Quién – Rol
- Qué – Algo
- Para qué – Beneficio

Validación: Le indica al desarrollador las validaciones internas que debe tener el sistema.

Criterios de aceptación: Lista detallada de requerimientos funcionales; pueden ser varios criterios por historia de usuario. Los criterios de aceptación se traducen en pruebas detalladas en pocas palabras. Utiliza un lenguaje entendible para que el equipo de desarrollo pueda especificar requerimientos funcionales, y es por ello que toman tanta importancia en las historias de usuario.

Los requisitos por parte del usuario identificados fueron los siguientes:

1) Poder consumir la API, ver Tabla 2.
2) Poder consumir la base de datos, ver Tabla 3.
3) Poder registrar archivos LCA, ver Tabla 4.
4) Poder ver la información LCA, ver Tabla 5.
5) Poder filtrar la información LCA, ver Tabla 6.
6) Poder agregar usuarios, ver Tabla 7.

Tabla 2: Historia de Usuario - Poder consumir la API.

Historia de Usuario			
Número	1	**Usuario**	Usuario
Nombre Historia		Poder consumir la API	
Prioridad	ALTA	**Riesgo en Desarrollo**	BAJO
Programador Responsable		Kamila Molina - Mateo Quizhpi	
Descripción			
El Product Owner desea que la API sea publicada en un servidor de la Universidad de Cuenca y que sea accesible mediante métodos HTTP.			
Validación			
1. El usuario debe iniciar sesión. 2. El usuario registrado puede acceder a la información. 3. Los usuarios tienen diferentes privilegios.			
Criterios de Aceptación			

1. No permitir usuarios sin privilegios asignados.

Tabla 3: Historia de Usuario - Poder acceder a la base de datos.

Historia de Usuario			
Número	2	**Usuario**	Usuario
Nombre Historia		Poder consumir la base de datos	
Prioridad	ALTA	**Riesgo en Desarrollo**	BAJO
Programador Responsable		Kamila Molina - Mateo Quizhpi	
Descripción			
El Product Owner desea que la base de datos esté publicada en un servidor de la Universidad de Cuenca y que sea accesible remotamente para poder consumirla.			
Validación			
1. Acceso mediante el puerto 3306. 2. Acceso restringido.			
Criterios de Aceptación			
1. No permitir usuarios sin privilegios asignados. 2. No permitir eliminar o editar los atributos y/o tablas. 3. Permitir visualizar la información.			

Tabla 4: Historia de Usuario - Poder registrar archivos LCA.

Historia de Usuario			
Número	3	**Usuario**	Usuario
Nombre Historia		Poder registrar archivos LCA	
Prioridad	ALTA	**Riesgo en Desarrollo**	ALTO
Programador Responsable		Kamila Molina	
Descripción			
El usuario desea agregar nuevos archivos LCA dentro del sistema y posteriormente sea el caso poderlos visualizar.			
Validación			
1. Verificar que sea una versión aceptada por el prototipo. 2. Verificar que no esté duplicada la versión ni el dataset.			
Criterios de Aceptación			
1. No permitir a usuarios sin privilegios subir un archivo LCA. 2. No permitir introducir un archivo LCA repetido. 3. No permitir no introducir la versión que se va a subir.			

Tabla 5: Historia de Usuario - Poder ver la información LCA.

Historia de Usuario			
Número	4	**Usuario**	Usuario
Nombre Historia		Poder ver la información LCA	
Prioridad	ALTA	**Riesgo en Desarrollo**	ALTO
Programador Responsable		Mateo Quizhpi	
Descripción			
El usuario desea ver la información referente al LCA dentro del sistema, de una manera estructurada, clara y personalizada conforme a lo deseado.			
Validación			
1. Verificar que se consume la base de datos al momento que el usuario realiza una consulta personalizada.			
Criterios de Aceptación			
1. Permitir a cualquier usuario ver la información. 2. Permitir ver toda la información. 3. No permitir modificar o eliminar datos.			

Tabla 6: Historia de Usuario - Poder filtrar la información LCA.

Historia de Usuario			
Número	5	**Usuario**	Usuario
Nombre Historia		Poder filtrar la información LCA	
Prioridad	ALTA	**Riesgo en Desarrollo**	ALTO
Programador Responsable		Kamila Molina	
Descripción			
El usuario desea filtrar la información referente al LCA dentro del sistema, y posteriormente sea el caso, poder visualizarlo. La información se filtra por varios campos, por ejemplo según los identificadores únicos de cada tupla, o colocando variables específicas que cumplan el propósito de filtrar los datos.			
Validación			
1. Verificar que se ingresen correctamente los parámetros de búsqueda y filtración de datos.			
Criterios de Aceptación			
1. Permitir a cualquier usuario ver la información. 2. No permitir ingresar otros valores de busqueda a mas de los establecidos			

3. Permitir ver la información filtrada. 4. No permitir modificar o eliminar datos.

Tabla 7: Historia de Usuario - Poder agregar usuarios.

Historia de Usuario			
Número	6	**Usuario**	Usuario
Nombre Historia		Poder agregar usuarios	
Prioridad	ALTA	**Riesgo en Desarrollo**	ALTO
Programador Responsable		Kamila Molina	
Descripción			
El Product Owner desea agregar nuevos usuarios y darle los privilegios de lectura a los nuevos usuarios.			
Validación			
1. Verificar que el usuario sea agregado.			
Criterios de Aceptación			
1. No permitir usuarios sin privilegios. 2. Permitir asignar privilegios a los nuevos usuarios.			

4.1.3. Planificación de Sprints

El propósito del Sprint es transformar un conjunto de elementos del Backlog del Producto en incrementos de funcionalidades potencialmente productivas. Además de dividir las historias de usuario de acuerdo con la prioridad asignada por el Product Owner para ser desarrolladas por parte del equipo de desarrollo. Las historias de usuario se ordenan por prioridad, desde las historias de usuario con prioridad ALTA, pasando por la prioridad MEDIA y culminando con las historias de usuario de prioridad BAJA hasta que se termine el producto completo de software.

En esta sección se explica el avance del proyecto mediante el uso de Sprints, En las tablas desde la 8 a la 14 se muestra las tareas realizadas en cada sprint, el tipo de actividad que representa la tarea (Análisis, Desarrollo o Actualización). Así como el estado (Finalizado), debido a que se terminó con la implementación. También en estas tablas se tiene el parámetro del esfuerzo, y está estimado en el número de semanas que tomaría cada tarea y cada Sprint.

Tabla 8: Sprint 1.

	Fecha inicio	Fecha entrega	Esfuerzo
Sprint 1	12/03/2021	30/04/2021	7

ID	Tarea	Tipo	Estado	Responsable	Esfuerzo
1	Análisis de los formatos de LCI	Análisis	Finalizado	Kamila Molina - Mateo Quizhpi	2
2	Definición del tipo de datos a almacenar (metadata, flujos, intercambios)	Análisis	Finalizado	Kamila Molina - Mateo Quizhpi	2
3	Revisión de la estructura del formato y el tipo definido de LCI	Análisis	Finalizado	Kamila Molina - Mateo Quizhpi	2
4	Definición de elementos prioritarios para la estructura de la información a almacenar	Análisis	Finalizado	Kamila Molina - Mateo Quizhpi	1

Tabla 9: Sprint 2.

	Fecha inicio	Fecha entrega	Esfuerzo
Sprint 2	30/04/2021	11/06/2021	6

ID	Tarea	Tipo	Estado	Responsable	Esfuerzo
5	Definición del modelo de administración de datos	Análisis	Finalizado	Kamila Molina - Mateo Quizhpi	1
6	Desarrollar el modelado de la base de datos	Desarrollo	Finalizado	Kamila Molina - Mateo Quizhpi	2
7	Definición de sistema gestor para la base de datos	Análisis	Finalizado	Kamila Molina - Mateo Quizhpi	1
8	Definición de las entidades y dependencias de la base datos	Análisis	Finalizado	Kamila Molina - Mateo Quizhpi	1
9	Proceso de creación de la base de datos en el sistema gestor	Desarrollo	Finalizado	Kamila Molina	1

Tabla 10: Sprint 3.

	Fecha inicio	Fecha entrega	Esfuerzo
Sprint 3	11/06/2021	27/08/2021	6

ID	Tarea	Tipo	Estado	Responsable	Esfuerzo
10	Desarrollo del método de lectura de los	Desarrollo	Finalizado	Kamila Molina	1

	archivos LCA				
11	Desarrollo del método de carga de archivos a la base de datos	Desarrollo	Finalizado	Kamila Molina	2
12	Pruebas de funcionalidad e integridad sobre la base de datos	Desarrollo / Análisis	Finalizado	Kamila Molina	1
13	Definición de la tecnología y ambiente de desarrollo para la API	Análisis	Finalizado	Kamila Molina - Mateo Quizhpi	1
14	Definición del modelo de objetos de la base de datos	Desarrollo / Análisis	Finalizado	Mateo Quizhpi	1
24	Desarrollo de las vistas para la API	Desarrollo	Finalizado	Mateo Quizhpi	1

Tabla 11: Sprint 4.

	Fecha inicio	Fecha entrega	Esfuerzo
Sprint 4	27/08/2021	22/10/2021	8

ID	Tarea	Tipo	Estado	Responsable	Esfuerzo
15	Pruebas de funcionalidad, integridad y libre acceso a la información referente a la base de datos	Análisis	Finalizado	Mateo Quizhpi	1
16	Desarrollo de consultas personalizadas para la API	Desarrollo	Finalizado	Kamila Molina	3
24	Desarrollo de las vistas para la API	Desarrollo / Análisis	Finalizado	Kamila Molina	1
25	Generación de documentación y mensajes personalizados de la API	Desarrollo	Finalizado	Mateo Quizhpi	1

Tabla 12: Sprint 5.

	Fecha inicio	Fecha entrega	Esfuerzo
Sprint 5	08/11/2021	22/11/2021	5

ID	Tarea	Tipo	Estado	Responsable	Esfuerzo
17	Análisis de la estructura de los archivos de correspondencia de las versiones EcoSpold.	Análisis	Finalizado	Kamila Molina - Mateo Quizhpi	1
18	Definición de las variables de los archivos de correspondencia a ser almacenadas	Diseño	Finalizado	Kamila Molina - Mateo Quizhpi	1

19	Desarrollar el modelo de la base de datos	Diseño	Finalizado	Kamila Molina	1
20	Definición de las entidades y dependencias de la base de datos	Diseño	Finalizado	Kamila Molina	1
21	Proceso de creación de nuevas tablas en la base de datos	Desarrollo	Finalizado	Kamila Molina	1

Tabla 13: Sprint 6.

	Fecha inicio	Fecha entrega	Esfuerzo
Sprint 6	22/11/2021	08/12/2021	5

ID	Tarea	Tipo	Estado	Responsable	Esfuerzo
22	Desarrollo del método de lectura de los archivos de correspondencia	Desarrollo	Finalizado	Kamila Molina	2
12	Pruebas de funcionalidad e integridad sobre la base de datos	Análisis	Finalizado	Kamila Molina - Mateo Quizhpi	1
19	Desarrollar el modelo de la base de datos	Desarrollo / Actualización	Finalizado	Mateo Quizhpi	1
26	Administración del acceso a la información mediante la API	Desarrollo	Finalizado	Mateo Quizhpi	1
28	Desarrollo del inicio de sesión en el sistema.	Desarrollo	Finalizado	Mateo Quizhpi	1

Tabla 14: Sprint 7.

	Start date	Delivery date	Effort
Sprint 7	08/12/2021	01/03/2022	8

ID	Task	Type	Estado	Responsible	Esfuerzo
23	Desarrollo de las vistas para la API.	Desarrollo / Análisis	Finalizado	Kamila Molina	1
27	Generación de reportes de la información mediante la API.	Desarrollo	Finalizado	Mateo Quizhpi	1
29	Creación de menú ítems para la interfaz de usuario.	Desarrollo	Finalizado	Mateo Quizhpi	1
30	Creación de un menú ítem para las consultas personalizadas de la metadata.	Desarrollo	Finalizado	Mateo Quizhpi	1
31	Creación de un menú ítem para el	Desarrollo	Finalizado	Mateo Quizhpi	1

	intercambio de versiones.				
32	Creación de un menú ítem para subir un archivo ecoinvent en formato .zip	Desarrollo	Finalizado	Mateo Quizhpi	1
33	Creación de un menú ítem para subir un archivo de correspondencia	Desarrollo	Finalizado	Mateo Quizhpi	1

4.2. Definición de Especificaciones.

Para construir el prototipo informático se comenzó con la recolección y refinamiento de requerimientos. El proceso se organizó en cuatro fases, 1) estudiar el porqué de la necesidad de ese proyecto, 2) analizar el formato de la fuente de datos LCA, 3) definir el origen de la fuente de datos que se tienen como entrada; y, finalmente, 4) definir la salida que genera el prototipo.

En la primera fase, se llevaron a cabo varias discusiones con el equipo de trabajo para llegar a un consenso sobre lo que es necesario abordar y lo que se quería lograr con este proyecto. En síntesis, según el análisis realizado a los LCI, estos no muestran la metainformación de sus LCA, por más que tienen acceso a ella. Este hecho fue el que fundamentó la construcción de un prototipo de software que maneje dicha metainformación. Con este trasfondo, la motivación principal de este proyecto es brindar una mejora en la calidad de información que se comparte con cada tupla de datos y el cómo, cuándo y dónde son generados los mismos. Actualmente se tiene acceso a la metainformación de los LCA leyendo directamente los archivos adecuados. No obstante, se busca responder ciertas preguntas sobre la metainformación, por eso es importante crear conexiones entre las diferentes versiones de LCI. Estas conexiones o relaciones se dan entre las tuplas de datos de LCA para agregar valor a la información que se obtiene al leer los diferentes archivos. Un punto importante dentro de esta discusión fue la confidencialidad sobre las tuplas de datos, ya que existe información tanto privada como pública. Por lo tanto, la finalidad de este proyecto es brindar reconocimiento a la información pública. Con esto se garantiza la privacidad de la información, al no compartir datos delicados libremente con los usuarios.

Continuando con la segunda fase, y después de haber determinado la importancia del proyecto, se estudiaron los diferentes formatos de LCA para posteriormente enfocar el proyecto de titulación en uno de ellos. El formato de la fuente de datos LCA que se analizó fue ecoinvent en la versión EcoSpold 2 (ecoinvent, 2021), pues es el formato con mayor detalle a nivel de atributos entre ellos. Esta versión contiene archivos en formato XML que cuentan con una estructura de información con diferentes variables como se puede ver en la *Fig. 6.*

Cada archivo en formato EcoSpold 2 presenta una estructura que agrupa de manera general la jerarquía de los 233 tipos de datos con los que cuenta la versión.

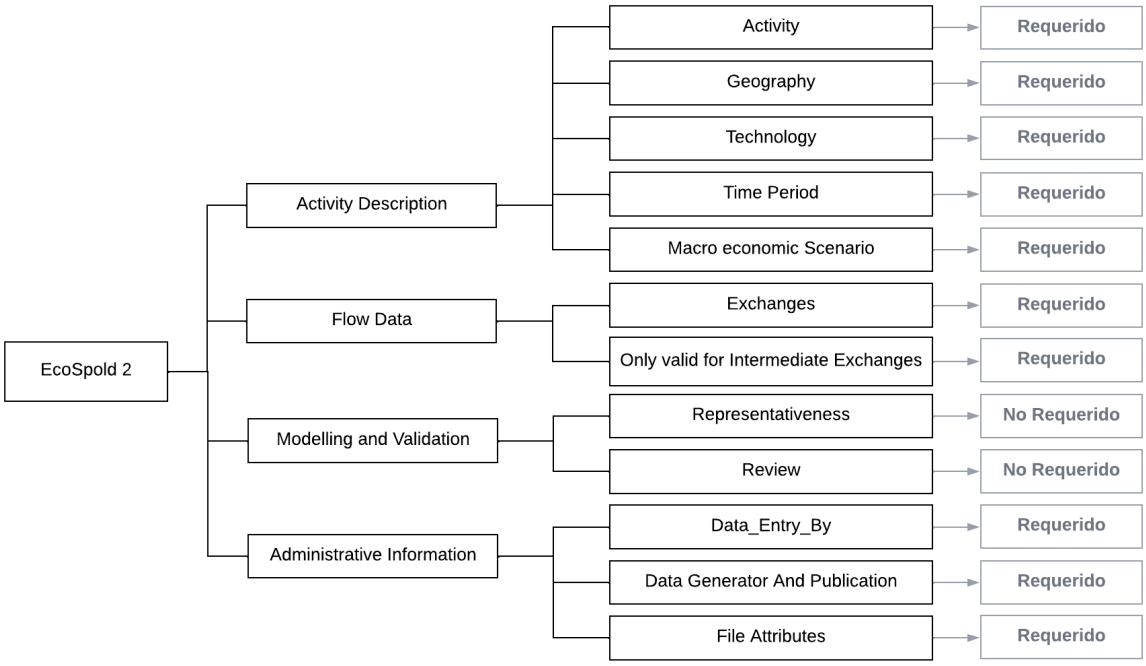


Fig. 6: Estructura de alto nivel de las secciones de datos de los archivos de formato EcoSpold 2, obtenida por la documentación, indicando las secciones requeridas o no.

Los archivos y documentos de respaldo provenientes de la documentación de Ecoinvent, ayudaron a formular el análisis que se le hizo al formato para examinar los requisitos de cada sección de datos. Cada archivo contiene diferentes variables que van a ser detalladas a continuación.

- ***Activity*** *(requerido)*, contiene la información de un conjunto de datos de actividad, incluido el nombre y la clasificación;
- ***Geography*** *(requerido)*, contiene información sobre la validez geográfica del proceso;
- ***Technology*** *(requerido)*, contiene una descripción de la tecnología para la que se han recopilado los datos;
- ***Time period*** *(requerido)*, describe la Fecha de inicio y finalización del período de tiempo para el que el conjunto de datos es válido;
- ***Macro economic Scenario*** *(requerido)*, hace referencia al escenario macroeconómico utilizado en este conjunto de datos;
- ***Exchanges*** *(requerido)*, contiene campos utilizados en varios tipos de intercambio;

- ***Only valid for Intermediate Exchanges*** *(requerido)*, comprende campos (adicionales a la sección de *exchanges*) que solo son válidos para intercambios intermedios;
- ***Representativeness*** *(no requerido)*, contiene información sobre la representatividad de la información en el conjunto de datos y sobre cómo se han muestreado y / o calculado los datos;
- ***Review*** *(no requerido)*, contiene información sobre quién llevó a cabo la revisión crítica y sobre los principales resultados y conclusiones de la revisión y las recomendaciones formuladas;
- ***Data entry by*** (requerido), contiene información sobre el autor del conjunto de datos;
- ***Data generator and publication*** (requerido), contiene información sobre quién recopiló, compiló o publicó los datos originales;
- ***File attributes*** (requerido), contiene metainformación sobre el archivo en su conjunto.

Según la documentación disponible en la página oficial de Ecoinvent, no todas las secciones mencionadas son de carácter obligatorio para construir un LCA, refiérase a la *Fig. 6.* Además, no todas las secciones son necesarias para este proyecto. Por lo tanto, se realizó un proceso de selección sobre las variables contenidas en cada sección, manteniendo las variables públicas y necesarias. Con esto se cumple con la confidencialidad de datos y el manejo de Metainformación de manera pertinente y acorde a lo requerido.

La tercera y cuarta fase estuvieron enfocadas en el origen y destino de los datos involucrados en el desarrollo del trabajo de titulación. Como parte de los estudios previos se analizaron las diferentes herramientas de software libre para LCA, como se mencionó en la sección 2.4, ya sea OpenLCA, Brightway2 o Antelope. Con este contexto, se pudo generar una comunicación consistente con el desarrollador de Antelope, quien brindó su apoyo desde el punto de vista LCI para construir, no solo una herramienta que ofrezca mayor relevancia a la metainformación, sino que también sea de utilidad para la comunidad científica con interés en información LCA. Para el proyecto de titulación, los archivos fueron provistos por el desarrollador de Antelope, y corresponden a diferentes versiones en el formato EcoSpold 2. Estos archivos fueron de completa utilidad para realizar el análisis e implementación del proyecto pues, como se mencionó con anterioridad, se deben extraer los atributos necesarios para el desarrollo del prototipo informático

4.3. Descripción de Requerimientos de Software.

Los requerimientos son un conjunto de procedimientos y técnicas que permiten definir los elementos necesarios para desarrollar un proyecto de software. El objetivo principal de capturar los requerimientos es comprender lo que clientes y usuarios esperan del sistema. Como parte del proyecto de titulación, se desarrolló un prototipo informático que da soporte a la metainformación referente a LCA, al cual se lo ha denominado INTI, en honor a su origen del quechua que significa sol.

El prototipo que se desarrolló presta diversas funcionalidades tales como obtener, controlar y analizar la meta información directamente de archivos LCA, con la finalidad de ofrecer una vista clara de la información a quien busque y controle el servicio. Como se puede apreciar en la *Fig. 7*, la estructura del modelo del proyecto de titulación se enfocó en cuatro puntos principales. Así, la *Fig. 7.a* contempló la construcción del modelo de la base de datos, la decisión de su estructura así como la definición del sistema gestor. La *Fig. 7.b* abarcó la implementación del prototipo de la API, la cual permitió establecer la conexión con la base de datos, que fue previamente implementada. La *Fig. 7.c* incluyó varios factores que se mencionan a continuación. El apartado de la *Fig. 7.c.1* incorporó la obtención de los archivos originales de LCA, ya sean los archivos en formato Ecospold 2 y los archivos de correspondencia. Seguido el apartado de la *Fig. 7.c.2* implicó el análisis ETL de los archivos del apartado *c.1* para su publicación en la base de datos. El apartado de la *Fig. 7.c.3* incluyó el proceso de visualización de la información (interfaz de usuario) de manera personalizada conforme a lo que un usuario desee consultar, obteniéndose al consumir la base de datos. Así como también en este apartado está presente el proceso de creación de reportes en base a la información previamente consultada. Y, finalmente, la *Fig. 7.d* engloba la implementación del proyecto de titulación, pues el prototipo (Base de datos, API, y UI) está albergado y publicado en un servidor de la Universidad de Cuenca.

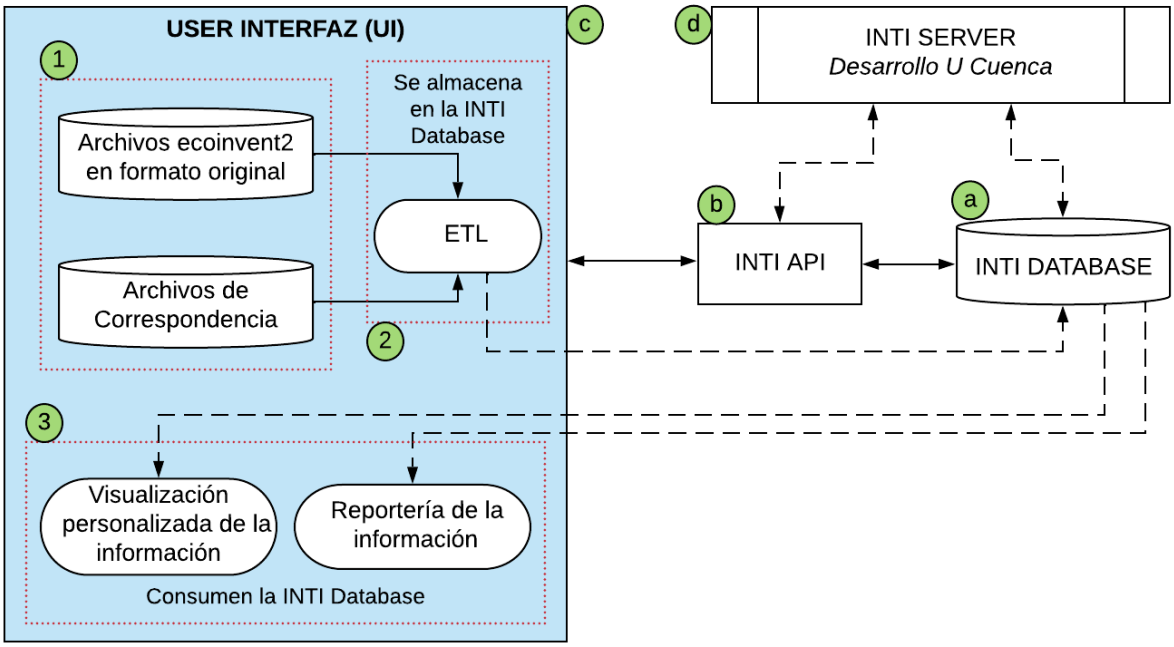


Fig. 7: Modelo arquitectónico del trabajo de titulación.

La captura y el análisis de los requisitos del sistema es una de las etapas más importantes para el éxito del proyecto, y al desarrollar un prototipo, se conduce hacia una especificación ejecutable. Sommerville (2005) plantea que una buena especificación de requerimientos debe dividir la funcionalidad de la implementación, para garantizar que se cumpla cada una de las especificaciones. A continuación se darán a detalle los requerimientos funcionales que describen las actividades del sistema y los no funcionales que especifican criterios para las operaciones del desarrollo del prototipo de software.

4.3.1. Requerimientos funcionales.

A continuación, se procede a dar una breve descripción de los requerimientos funcionales del prototipo informático utilizando lenguaje natural, de modo que sea de fácil entendimiento para el usuario final del sistema. Para cada requerimiento funcional, se indica un código, nombre que identifica la funcionalidad y una descripción breve que explica su función. Los requerimientos funcionales para la construcción de la aplicación web son presentados en la Tabla 15.

Tabla 15: Tabla resumen de requerimientos funcionales.

Código	Nombre	Descripción
R-001	Inicio de Sesión	El usuario debe iniciar sesión para tener acceso a las funcionalidades del sistema, asignadas según sus privilegios.

R-002	Cuenta de Administrador	El administrador podrá tener acceso a las siguientes interfaces: cargar archivos, ver información, generar reportes, gestión de roles, gestión de usuarios.
R-003	Cuenta de Usuario Estándar	El usuario podrá tener acceso a las siguientes interfaces: ver información, generar reportes.
R-004	Cargar Archivos	El usuario con privilegios podrá cargar archivos LCI en formato Ecospold dentro del sistema.
R-005	Ver Información	Los usuarios podrán consultar la información referente al LCA deseado.
R-006	Ver Información Personalizada	Los usuarios podrán realizar consultas personalizadas en la vista de la información para obtener resultados más específicos.
R-007	Generar Reportes	Los usuarios podrán generar reportes según la consulta que estén realizando referente al LCA.
R-008	Gestión de Roles	El administrador podrá crear, editar y eliminar privilegios de usuarios.
R-009	Gestión de Usuarios	El administrador podrá crear, editar y eliminar usuarios.
R-010	Gestión de la Base de Datos	La base de datos permite centralizar la información del sistema informático.
R-011	Publicar en el Servidor	Tanto la base de datos como la API estarán publicadas en el servidor de la Universidad de Cuenca.

4.3.2. Requerimientos no funcionales.

A continuación, se procede a especificar los requerimientos no funcionales del sistema usando lenguaje natural, de modo que sea de fácil entendimiento para el usuario final del sistema. Los requerimientos no funcionales son los presentados en la Tabla 16.:

Tabla 16: Tabla resumen de requerimientos no funcionales.

Código	**Nombre**	**Descripción**
R-1	La aplicación debe ser desarrollada para cualquier dispositivo con acceso a internet.	El usuario puede ingresar en la plataforma desde cualquier dispositivo con una conexión a internet.
R-2	La aplicación debe ser desarrollada con el IDE de Django Rest Framework y en lenguaje de programación Python.	Los desarrolladores emplean este framework de desarrollo ya que se complementa con el lenguaje de programación y este tiene módulos que facilitan el tratamiento de los archivos.
R-3	La aplicación debe tener conexión a internet.	El usuario debe tener acceso a internet desde el dispositivo por el cual desee

		ingresar.
R-4	La aplicación debe mantener conexión constante con una web service de la Universidad de Cuenca que proveerá de todos los datos.	Los programadores deben garantizar que la aplicación web publicada tenga una conexión estable.
R-5	El sistema debe ser capaz de operar adecuadamente con sesiones concurrentes.	Varios usuarios pueden iniciar sesión en el sistema y obtener respuestas eficientes a sus consultas.
R-6	Los permisos de acceso al sistema podrán ser cambiados solamente por el administrador de acceso a datos.	El administrador del sistema es el encargado de dar o retirar privilegios a los demás usuarios.
R-7	El tiempo de aprendizaje del sistema por un usuario deberá ser menor a cuatro horas.	Se desarrolla la plataforma con un ambiente amigable para garantizar una buena usabilidad por parte de los usuarios.
R-8	En términos de interacción humano-máquina, la tasa de errores cometidos por el usuario deberá ser menor del 2% de las acciones totales ejecutadas en el sistema.	El usuario comprende lo que desea lograr al interactuar con el prototipo de software web.
R-9	El sistema debe poseer interfaces gráficas bien estructuradas y con estilo y formato común con los respectivos manuales de usuario.	Los programadores diseñan una interfaz amigable para el usuario, con un diseño que ayude al usuario a navegar de manera intuitiva a través del sistema.

4.4. Implementación.

Luego del levantamiento de requerimientos del proyecto, y siguiendo la metodología de desarrollo Scrum, se inició el diseño y la construcción de los diferentes módulos que conforman el prototipo. Los módulos están divididos entre la base de datos, la API y la interfaz de usuario. A continuación se detalla la implementación de cada uno de ellos.

4.4.1. Diseño y despliegue de la Base de Datos Relacional.

La elaboración del diagrama ER (Entidad-Relación) para el proyecto se dió a lo largo de las primeras reuniones con el equipo de trabajo. Aquí se expusieron las ideas iniciales acerca de la naturaleza del trabajo, con lo cual al ya tener una base más clara se procede a realizar el diagrama ER final para la base de datos.

- Recopilación y análisis de requisitos

Esta primera fase fue un paso previo para asegurar que el sistema cumpliera con los objetivos. Se analizaron distintos factores, tales como los datos que se van a analizar, filtrando la información que los datos detallan para saber cuáles se deben almacenar y los que no.

En esta fase, fue indispensable analizar la estructura de datos de archivos LCA del formato ecoSpold 2 directamente desde su documentación, para filtrar la información relevante de su estructura de datos. En dichos archivos se encontraba toda la información sobre LCA, como los intercambios, las cantidades, los flujos y también la metainformación. Como se obtiene más información de la pertinente, se procedió a filtrar solo los datos que son de interés para este proyecto, es decir, la metainformación.

- Diseño conceptual

  El diseño conceptual comprende una descripción de alto nivel del contenido que tiene la base de datos, es decir, se define en un diagrama las entidades, importantes y las relaciones entre ellas. Como se puede ver en la *Fig. 8*, el diagrama está estructurado conforme a las actividades, procesos y metainformación disponible de los archivos LCA necesarios para las consultas personalizadas que deseen hacer usuarios finales.

- Diseño físico

  En esta fase se definen las estructuras de almacenamiento de la base de datos de forma física. Representa cómo se construirá el modelo en la base de datos, es decir, muestra todas las estructuras de la tabla, incluidos el nombre de columna, el tipo de datos, las restricciones, las claves y las relaciones entre las tablas. Como se puede ver en la *Fig. 9*, está la especificación de todas las tablas y columnas, así como las claves externas que se usan para identificar relaciones entre tablas.

- Diseño lógico

  El diseño lógico describe los datos con el mayor detalle posible, es decir, como se puede ver en la *Fig. 10*, se muestra la estructura de la base de datos INTI. Así, este nivel de diseño incluye las tablas, con sus filas, columnas y relaciones que se ingresarán en el SGBD.

- Implementación

  La base de datos INTI fue modelada y poblada de manera local, en la versión de postgreSQL 13.6. Para publicar la base de datos en el servidor, se hizo mediante la copia de archivos a través de Secure Shell (SSH) con

Secure Copy (SCP). SCP hace uso de SSH para hacer copias seguras y encriptadas entre el ordenador local y el servidor sin necesidad de abrir ningún programa pues lo hace desde consola.

Por lo tanto se copió el fichero del archivo que contiene la base de datos poblada en el servidor con el siguiente comando:

```
$ scp db_inti.sql usuario@dominio.com:/home/resources
```

Finalmente, se tiene la base de datos, funcional y poblada dentro del servidor de la Universidad de Cuenca provisto, para garantizar el fácil acceso a la información y de esta manera consumir la base de datos cuando sea necesario y pertinente. Las consultas pueden constar entre, ya sean para consultas personalizadas o para continuar con la población de la base de datos mediante el uso del prototipo.

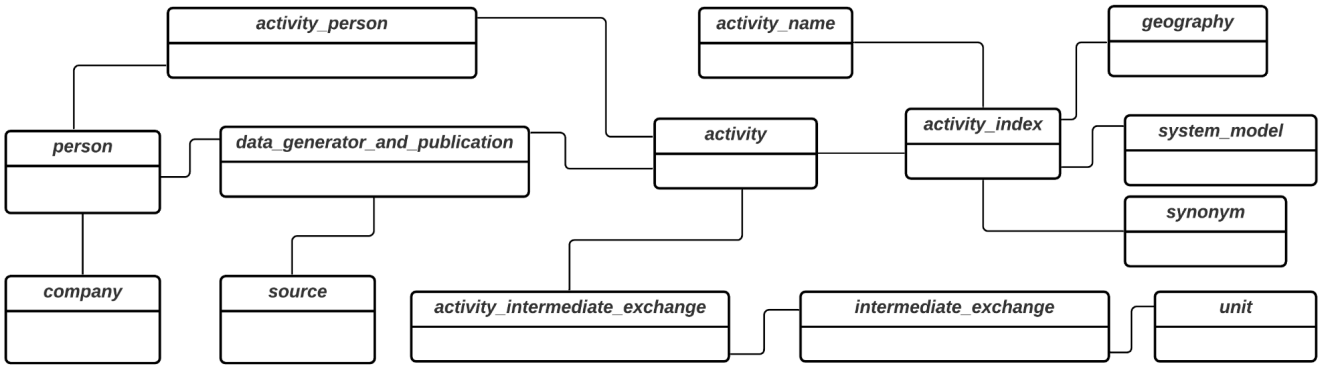


Fig. 8: Modelo de datos conceptuales de la base de datos INTI.

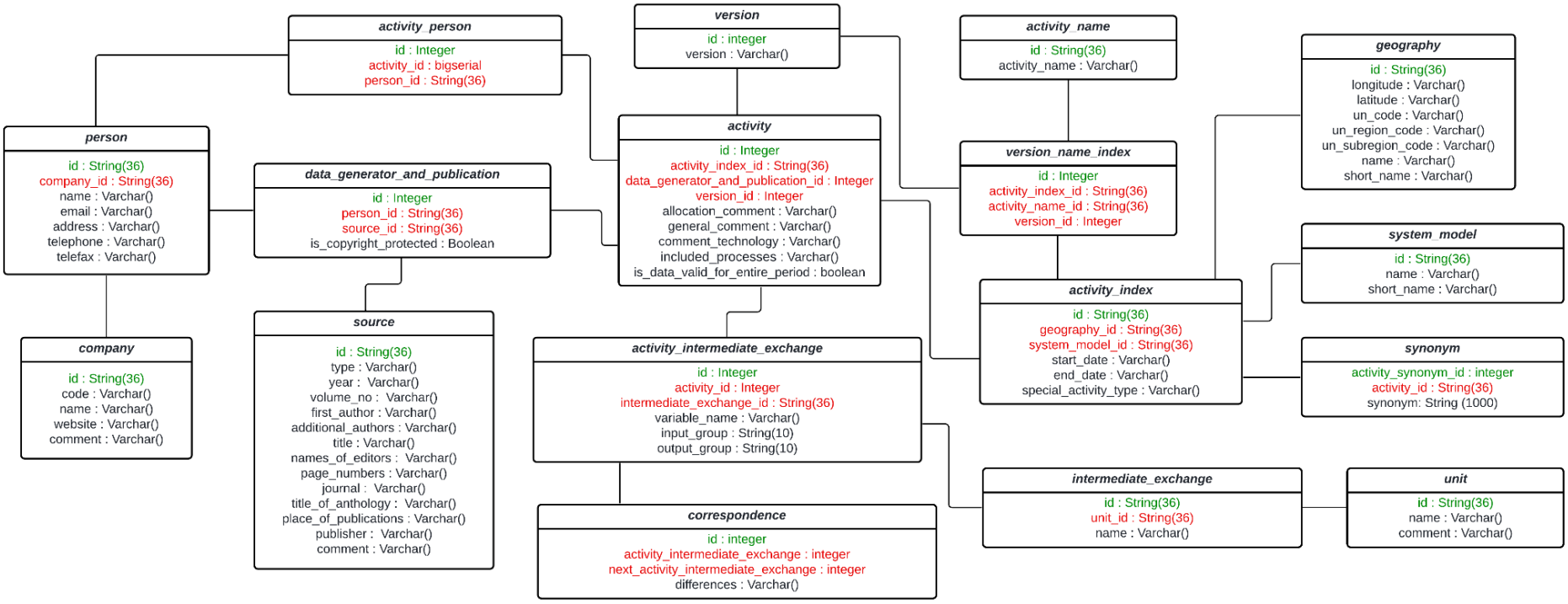


Fig. 9: Modelo de datos físico de la base de datos INTI. De color verde las claves primarias, de color rojo las claves foráneas.

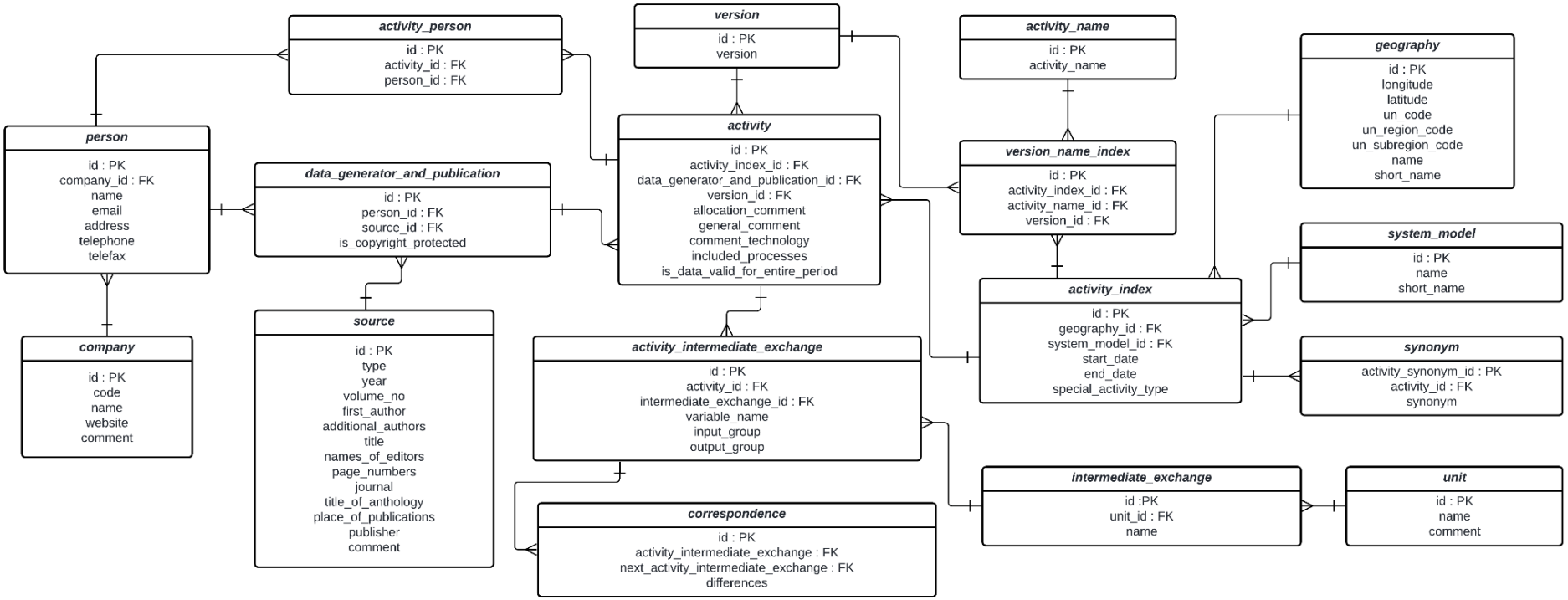


Fig. 10: Modelo de datos lógico de la base de datos INTI.

4.4.2. Modelado e Implementación de la API.

Para empezar el modelado de la API se debe tener en cuenta factores de diseño como: i) público objetivo en calidad de usuario final de la interfaz de programación de aplicaciones, ii) propósito funcional de las partes interesadas y iii) recursos en referencia al tipo y contexto de los datos a gestionar. De ello, se obtiene un proceso metodológico para lograr la consolidación en la gestión de información descrito en las siguientes fases de implementación. En este sentido, como se ha mencionado anteriormente en la Sección 3.4, se empleó Django Rest Framework para su desarrollo.

De esta manera, la API está enfocada para desarrolladores de todo tipo de experiencia debido al uso de la tecnología citada y su capacidad de escalabilidad. En definitiva, la API es de libre acceso gracias al uso del framework con licencia no privativa para el avance científico de su dominio. En este contexto, la *Fig. 11* representa la arquitectura de la interfaz de programación de aplicaciones. En primer lugar, el componente *Modelo* mapea las entidades definidas en la Base de Datos. Luego, el *Serializador* otorga un formato de tipo JSON o XML para su modelo de referencia. A continuación, la *Vista* define los distintos métodos de gestión de información dentro del marco REST. Finalmente, por cada vista se asocia un *URL* como punto final para el usuario.

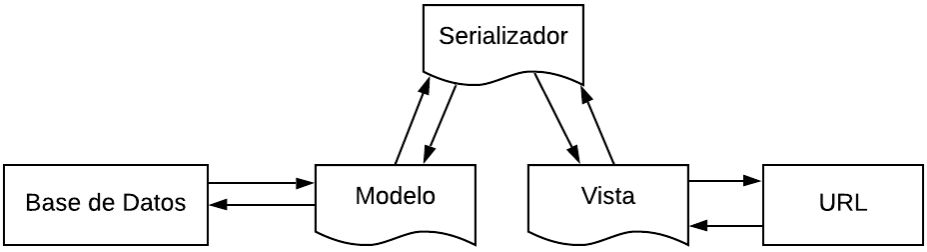


Fig. 11: Diagrama de la arquitectura general de la API.

4.4.2.1. Modelado de datos.

En referencia a la obtención del mapeo de las entidades ilustradas en la *Fig. 10,* se empleó un comando interno de Django llamado *inspectdb* que puede crear modelos mediante la introspección de una base de datos existente. Por lo tanto, después del diseño lógico de la base de datos y la aplicación de este comando se obtuvo un archivo comúnmente denominado *models.py* que contiene las entidades vinculadas de la base de datos que se especifica en el archivo de configuración del proyecto.

A manera de ejemplo del funcionamiento del comando *inspectdb*, el modelo de la entidad *activity* se presenta en la *Fig. 12.* Esta corresponde a las características de los campos con su respectiva referencia hacia la tabla de la base de datos heredada. De este modo, todas las entidades del diseño lógico son modeladas automáticamente con la particularidad de utilizar solo las clases que contienen información potencialmente pública.

```
class Activity(models.Model):
       activity_index = models.ForeignKey('ActivityIndex', models.DO_NOTHING)
       allocation_comment = models.CharField(max_length=32000, blank=True, null=True)
       general_comment = models.CharField(max_length=32000, blank=True, null=True)
       comment_technology = models.CharField(max_length=32000, blank=True, null=True)
       included_processes = models.CharField(max_length=32000, blank=True, null=True)
       id = models.BigAutoField(primary_key=True)
       is_data_valid_for_entire_period = models.BooleanField(blank=True, null=True)
       data_generator_and_publication =
models.ForeignKey('DataGeneratorAndPublication',                models.DO_NOTHING)
       version = models.ForeignKey('Version', models.DO_NOTHING)

       class Meta:
       managed = False
       db_table = 'activity'
```

Fig. 12: Clase Activity del archivo models.py

4.4.2.2. Vistas y URLs de la aplicación.

Seguidamente, fue necesario preparar un entorno para que los datos consolidados en los modelos puedan ser procesados para el dominio del proyecto. De este modo, fue preciso utilizar serializadores para que datos complejos, como conjuntos de consultas e instancias de modelos, se conviertan a tipos de datos nativos de Python que luego se puedan representar mediante distintos formatos JSON, XML u otros tipos de contenido. De esta manera, se creó un archivo llamado *general_serializer.py* que referencia al modelo particular como ejemplifica la *Fig. 13.* Tiene la singularidad de definir una función de representación con características predeterminadas de los campos de una entidad, en este caso *Activity.* Por otra parte, las variables definidas al inicio de la clase permiten indexar, mediante las claves foráneas, la información de otros serializadores relacionados; en este caso a *ActivitySerializer,* con el propósito de obtener instancias más completas de cada recurso.

```
class ActivitySerializer(serializers.ModelSerializer):
	activity_index = ActivityIndexSerializer()
	data_generator_and_publication = DataGeneratorAndPublicationSerializer()
	version = VersionSerializer()

	class Meta:
	model = Activity
	fields = '__all__'

	def to_representation(self, data):
	data = super(ActivitySerializer, self).to_representation(data)
	data['allocation_comment'] = " ".join(data['allocation_comment'].split())
	data['general_comment'] = " ".join(data['general_comment'].split())
	data['comment_technology'] = " ".join(data['comment_technology'].split())
	data['included_processes'] = " ".join(data['included_processes'].split())
	return data
```

Fig. 13: Clase ActivitySerializer del archivo general_serializer.py

Por consiguiente, en la arquitectura mencionada en la Sección 3.3, es necesario emplear vistas para que la información pueda liberarse mediante respuestas en el marco REST. Django Rest Framework utiliza vistas basadas en clases de las cuales resalta la clase propia *Viewsets* que contiene acciones preestablecidas para crear, leer, actualizar, y eliminar información. A modo de ejemplo, se puede observar en la *Fig. 14,* la clase *ActivityViewSet* que implementa las acciones de gestión de información mediante las funciones *create, update* y *destroy.* Es importante mencionar que la función *get_queryset* permite comunicarse con la información definida en el serializador y modelo de la entidad particular.

```
class ActivityViewSet(viewsets.ModelViewSet):
	serializer_class = general_serializer.ActivitySerializer

	def get_queryset(self, pk=None):
	if pk is None:
		return self.get_serializer().Meta.model.objects.all()
	else:
		return self.get_serializer().Meta.model.objects.filter(id = pk).first()

	def create(self, request):
	serializer = self.serializer_class(data = request.data)
	if serializer.is_valid():
		serializer.save()
		return Response({'message': 'Activity created successfully'},status=
			status.HTTP_201_CREATED)
	return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

	def update(self, request, pk=None):
	if self.get_queryset(pk):
		activity_serializer = self.serializer_class(self.get_queryset(pk), data
=			request.data)
		if activity_serializer.is_valid():
		activity_serializer.save()
		return Response(activity_serializer.data, status= status.HTTP_200_OK)
		return Response(activity_serializer.errors, status=
			status.HTTP_400_BAD_REQUEST)


	def destroy(self, request, pk=None):
	activity = self.get_queryset().filter(id = pk).first()
	if activity:
		activity.delete()
		return Response({'message':'Activity removed successfully'}, status=
```

```
                    status.HTTP_200_OK)
        return Response({'message':'No Activity found with this ID'}, status=
                    status.HTTP_400_BAD_REQUEST)
```

Fig. 14: Clase ActivityViewSet del archivo activity_viewsets.py

Finalmente, la utilización de la clase propia *viewsets* permite el uso de un enrutador, que determina automáticamente la URL según la acción registrada en dicha clase. Por lo tanto, solo es necesario asociar un nombre base y el *viewset* correspondiente para registrar cada enrutador como ilustra la *Fig. 15.* Django Rest Framework ofrece una interfaz de usuario para la gestión de la API. Como se puede observar en la *Fig. 16,* la URL matriz del proyecto contiene todas las direcciones específicas relacionadas a cada enrutador.

```
router = DefaultRouter()

router.register(r'activities',activity_viewsets.ActivityViewSet, basename= 'activity')
router.register(r'versions',version_viewsets.VersionViewSet, basename= 'version')
router.register(r'companies',company_viewsets.CompanyViewSet, basename='company')
router.register(r'people',person_viewsets.PersonViewSet, basename='person')
router.register(r'geographies',geography_viewsets.GeographyViewSet,
basename='geography')
router.register(r'units',unit_viewsets.UnitViewSet, basename='unit')
router.register(r'activities_index',activity_index_viewsets.ActivityIndexViewSet,
basename= 'activity index')
router.register(r'activities_name',activity_name_viewsets.ActivityNameViewSet,
basename= 'activity name')
router.register(r'activities_person',activity_person_viewsets.ActivityPersonViewSet,
basename= 'activity person')
router.register(r'data_generator_publications',data_generator_publication_viewsets.Dat
aGeneratorAndPublicationViewSet, basename= 'data generator and publications')
router.register(r'intermediate_exchanges',intermediate_exchange_viewsets.IntermediateE
xchangeViewSet, basename= 'intermediate exchange')
router.register(r'properties',property_viewsets.PropertyViewSet, basename= 'property')
router.register(r'sources',source_viewsets.SourceViewSet, basename= 'source')
router.register(r'synonyms',synonym_viewsets.SynonymViewSet, basename= 'synonym')
router.register(r'system_models',system_model_viewsets.SystemModelViewSet, basename=
'system model')
router.register(r'version_name_indexes',version_name_index_viewsets.VersionNameIndexVi
ewSet, basename= 'version name index')


urlpatterns = router.urls
```

Fig. 15: Enrutadores registrados de los viewsets almacenados en el archivo routers.py

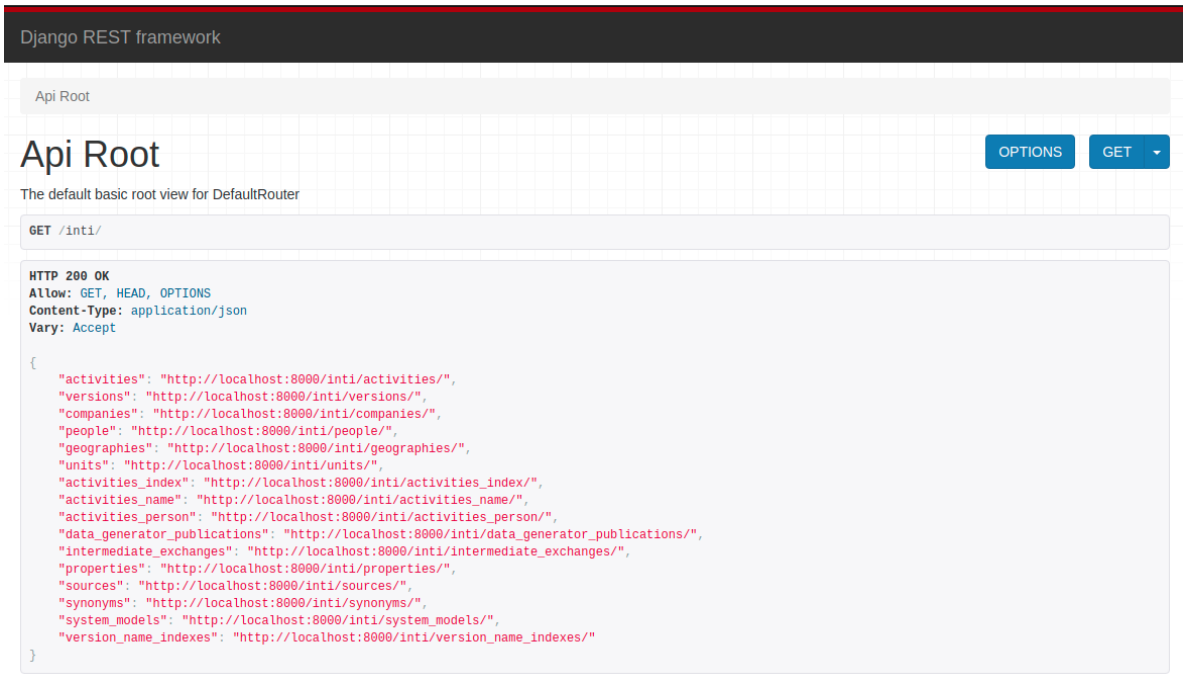


Fig. 16: API root asociada a la URL principal del proyecto.

4.4.2.3. Consultas personalizadas.

Es necesario mencionar el desarrollo y la implementación de las consultas personalizadas, debido a su importancia para el dominio de la aplicación, como se mencionó en la Sección 4.1. La Tabla 17 muestra las consultas idóneas para visualizar la interrelación entre las distintas entidades ilustradas en la *Fig. 10*. Este proceso se llevó a cabo utilizando el método GET, siguiendo la estructura de una API RESTful para poder ingresar los filtros que se desean como parte de la URL de cada vista de la aplicación.

A manera explicativa se tiene una tabla que describe el funcionamiento de las consultas, al lado izquierdo está una descripción en lenguaje natural para que el usuario identifique la funcionalidad de la consulta, y a la derecha está la URL que ejemplifica el uso correspondiente de la misma consulta dentro del proyecto.

Tabla 17: Descripción de las consultas personalizadas de la API.

	Definición de la consulta	**URL de ejemplificación**
1	Ver todas las actividades con el mismo nombre, filtrado por el nombre de la versión.	inti/activities/same_names/?version=*{version name}* ● *{version name}=ecoinvent 3.6 cut off*
2	Ver actividades con nombres similares, filtrado por el nombre o	inti/activities/similar_names/?name=*{activity name}* ● *{activity name}=aluminium*

	palabra clave del nombre de la actividad.	
3	Buscar actividades similares entre los flujos de referencia, filtrado por el nombre de la versión y el nombre o palabra clave de la actividad.	inti/activities/alike_flows/?version=*{version name}*&name={*activity name}* ● *{version name}=ecoinvent 3.6 cut off* ● *{name}=trawler*
4	Ver actividades que comparten la misma geografía, filtrado por el id de la geografía.	inti/geographies/{pk}/activities/ ● {pk}=34dbbff8-88ce-11de-ad60-0019e336be3a
5	Todos los flujos de referencia asociados con una actividad, filtrado por el id de la actividad y/o por el nombre de la versión.	inti/activities/{pk}/flows/ ● {pk}=184
		inti/activities/{pk}/flows/?version=*{version name}* ● {pk}=184 ● *{version name}=ecoinvent 3.6 cut off*
6	Actividades que tienen el mismo flujo de referencia, filtrado por el id del intercambio intermedio y/o por el nombre de la versión.	/inti/intermediate_exchanges/*{pk}*/activities/ ● {pk}=42761d87-05d9-4877-b21e-001ecf0c747d
		/inti/intermediate_exchanges/*{pk}*/activities/?version=*{version name}* ● *{pk}=42761d87-05d9-4877-b21e-001ecf0c747d* ● *{version name}=ecoinvent 3.6 cut off*
7	Referencias asociadas a una actividad, filtrado por el id de la actividad.	inti/activities/*{pk}*/references/ ● {pk}=184
8	Productores asociados a un intercambio intermedio, filtrado por el id del intercambio intermedio.	/inti/intermediate_exchanges/*{pk}*/producers/ ● {pk}=42761d87-05d9-4877-b21e-001ecf0c747d
9	Personas asociadas con una actividad, filtrado por el id de la actividad.	inti/activities/*{pk}*/people/ ● {pk}=184
10	Actividades asociadas con una persona, filtrado por el id de la persona.	inti/people/*{pk}*/activities/ ● {pk}=253759c9-c631-44cc-b293-955a2f6d4cba

El proceso de desarrollo de las consultas personalizadas, se implementó dentro de las respectivas clases *ViewSet* que definen las tablas. Como se observó en la *Fig. 14*, estas clases contienen métodos internos, por lo que para implementar las consultas personalizadas, se agregaron nuevos métodos que corresponden a las consultas personalizadas.

A manera de ejemplo se dará detalle de un método implementado dentro de la clase *ActivityViewSet*, este método se llama *same_names().* El nombre de la función y los parámetros que recibe son lo que definen la dirección URL. Como se puede ver a continuación en la *Fig. 17*, el proceso del método es el siguiente,

primero recibe como variable al nombre de la versión. Segundo, se inicia una conexión con la base de datos mediante la función *cursor()* y ejecuta la consulta sql previamente definida. Cabe mencionar que la sentencia sql cambia para todas las consultas personalizadas, pues cada una de ellas trabaja con diferentes tablas de la base de datos para responder diferentes preguntas. En este caso la consulta sql se centra en obtener los nombres únicos de las actividades según la versión ingresada como parámetro. Tercero, después de ejecutar la consulta sql y obtener una respuesta tipo tupla, el método construye un objeto, con el cual recorre las tuplas obtenidas en la respuesta sql y las va anidando según corresponda y asignándoles un nombre de atributo. Luego se debe construir el JSON, esto se da mediante la función *json.dumps(),* la cual convierte el objeto construído previamente en un JSON string; luego, con la función *json.loads(),* se deserializa el string previo, para convertirlo en un Diccionario Python. Finalmente, se validan si se tienen datos dentro del objeto, para poder enviar la respuesta ya transformada a JSON y paginada a la URL definida por medio de la función *Response*.

```
def same_names(self, request):
     version=request.GET.get('version','') #Version name = ecoinvent 3.6 cut off
     data = {}
     data = []
     cursor1 = conexion.cursor()
     select = "select DISTINCT an.id, vni.activity_index_id, S1.activity_name,
          S1.version from (select activity_name.activity_name, version.version,
count(*)
          from activity_name, version_name_index, version where
          activity_name.id=version_name_index.activity_name_id and
          version.id=version_name_index.version_id and version.version='"+version+"'
          group by activity_name.activity_name, version.id) S1, activity_name an,
          version_name_index vni where an.id=vni.activity_name_id and
          an.activity_name=S1.activity_name and vni.activity_name_id=an.id"
    cursor1.execute(select)
    select = cursor1.fetchall()
    for row in select:
         data.append({
              "activity index id": str(row[0]),
              "activity name": str(row[2]),
              "version": str(row[2])
         })
     s1 = json.dumps(data)
     d1 = json.loads(s1)
     d2=self.paginate_queryset(d1)
     if len(select) == 0:
          return Response({'response': 'no data found'})
     else:
          return Response(d2)
```

Fig. 17: Método same_names() de la clase ActivityViewSet del archivo activity_viewsets.py

En recopilación y como se mencionó, todas las consultas personalizadas siguen un proceso similar al mencionado en la *Fig. 17*. Se define una nueva función para cada consulta personalizada dentro de cada clase *ViewSet*, se construye un JSON a partir de la respuesta *sql* que se obtiene según sea el caso de la consulta y se envía la respuesta al URL que indica el nombre de la función.

4.4.2.4. Intercambio de versiones.

Finalmente, el intercambio de versiones es un proceso llevado a cabo para que el usuario final pueda conocer la información de actividades de una versión definida a una futura. Este proceso se da igual que en el caso de las consultas personalizadas, utilizando el método GET y obteniendo una URL específica, como se detalla en la Tabla 18. A manera explicativa, se tiene al lado izquierdo de la tabla una descripción en lenguaje natural para que el usuario identifique la funcionalidad de la consulta, y a la derecha está la URL que ejemplifica el uso correspondiente de la misma consulta dentro del proyecto.

Tabla 18: Descripción de la consulta de correspondencia de la API.

Definición de la consulta	URL de ejemplificación
Ver todas las actividades y su intercambio de referencia correspondiente entre las versiones, filtrado por el id de la actividad y el id del intercambio intermedio.	/inti/correspondence/version/?ai-id=*{ai-id}*&ie-name =*{ie-name}* ● *{ai-id}=08550bca-a4c1-4373-890b-8ef2ceaae42c* ● *{ie-name}=bagasse, from sugarcane*

Para generar esta URL, y como indica la *Fig. 18*, se crea una función llamada *version()* dentro de la clase *CorrespondenceViewSet*. El proceso de este método es el siguiente. Primero, recibe como parámetro de entrada el id de la actividad y el nombre del intercambio que se desea conocer. Se trabaja de esta manera pues una actividad puede tener varios intercambios intermedios, por lo que en este caso se filtra para un intercambio único y así obtener como resultado las actividades con su intercambio intermedio correspondiente de las siguientes versiones.

Entonces, inicia el objeto donde se concatenan las respuestas de las consultas. Segundo, se inicia una conexión con la base de datos mediante la función *cursor()* y se ejecuta la consulta sql previamente definida. Cabe mencionar que la sentencia sql cambia durante la ejecución del método, pues cada una de las consultas trabaja con las respuestas de la consulta anterior, es decir, es un método recursivo para obtener todas las actividades. Tercero, después de ejecutar la consulta sql que corresponda al paso en el que esté y obtener una respuesta tipo tupla, el método igualmente construye un objeto, que asigna cada valor de la respuesta sql a un atributo del objeto. Este paso sucede con todas las consultas que se den, pues como se mencionó, cada consulta recibe como parámetro la actividad y el intercambio de la respuesta anterior hasta que ya no encuentre una correspondencia a una actividad futura. Luego de concatenar todas las respuestas al objeto, se debe construir el JSON, como se detalló en la sección anterior. Este proceso se da mediante la función *json.dumps(),* que convierte el objeto en un JSON string, y luego, con la función *json.loads(),* se convierte el string en un

Diccionario Python. Finalmente, se validan si se tienen datos dentro del objeto, para poder enviar la respuesta ya transformada a JSON y paginada a la URL definida por medio de la función *Response*.

```
def version(self, request):
     ai_id=request.GET.get('ai-id','')
     ie_name=request.GET.get('ie-name','') # ie.name='bagasse, from sugarcane'
     data = {}
     data = []
     cursor1 = conexion.cursor()
     select1 = "select c.activity_intermediate_exchange,
          c.next_activity_intermediate_exchange, a.activity_index_id from
correspondence
          c, activity_intermediate_exchange aie, intermediate_exchange ie, activity a
          where ie.id=aie.intermediate_exchange_id and
          c.activity_intermediate_exchange=aie.id and aie.activity_id=a.id and
          a.activity_index_id='"+ai_id+"' and ie.name='"+ie_name+"';"
     cursor1.execute(select1)
     select1 = cursor1.fetchall()
     if len(select1)!=0:
          select1_1 = "select ie.name intermediate_exchange, a.activity_index_id from
               intermediate_exchange ie, activity_intermediate_exchange aie , activity
a
               where a.id=aie.activity_id and aie.intermediate_exchange_id=ie.id and
               aie.id='"+str(select1[0][1])+"';"
          cursor1.execute(select1_1)
          select1_1 = cursor1.fetchall()
          if len(select1_1)!=0:
               for row in select1:
                    data.append({
                         "activity intermediate exchange 3_1": str(row[0]),
                         "activity intermediate exchange 3_1 id": str(row[2]),
                         "activity intermediate exchange 3_2": str(row[1]),
                         "activity intermediate exchange 3_2 id":
str(select1_1[0][1]),
                     })

     select2 = "select c.activity_intermediate_exchange aie_3_1,
          c.next_activity_intermediate_exchange aie_3_2 from correspondence c,
          activity_intermediate_exchange aie, intermediate_exchange ie, activity a
          where ie.id=aie.intermediate_exchange_id and
          c.activity_intermediate_exchange=aie.id and aie.activity_id=a.id and
          a.activity_index_id='"+select1_1[0][1]+"' and
ie.name='"+select1_1[0][0]+"';"
     cursor1.execute(select2)
     select2 = cursor1.fetchall()
     if len(select2)!=0:
          select2_1 = "select ie.name intermediate_exchange, a.activity_index_id
               from intermediate_exchange ie, activity_intermediate_exchange aie ,
               activity a where a.id=aie.activity_id and
               aie.intermediate_exchange_id=ie.id and
aie.id='"+str(select2[0][1])+"';"
          cursor1.execute(select2_1)
          select2_1 = cursor1.fetchall()
          if len(select2_1)!=0:
               for row in select2:
                    data.append({
                         "activity intermediate exchange 3_3": str(row[1]),
                         "activity intermediate exchange 3_3 id":
str(select2_1[0][1]),
                    })
.
.
.
     s1 = json.dumps(data)
     d1 = json.loads(s1)
     d2=self.paginate_queryset(d1)
     if len(data) == 0:
```

```
            return Response({'response': 'no data found'})
    else:
            return Response(d2)
```

Fig. 18: Extracto del método version() de la clase CorrespondenceViewSet del archivo correspondence_viewsets.py

Finalmente, con este método, es posible conocer el recorrido de una versión, en cuanto a saber su nombre en versiones posteriores si es que existe, y si ya no se encuentra una versión posterior, conocer hasta qué versión fué hábil tal actividad e intercambio. Para concluir, la respuesta que se obtiene en el JSON, está conformada por atributos denominados: *intercambios intermedios,* los cuales corresponden a las diferentes versiones que exista esa actividad. Con esto, el resultado puede ser interpretado y analizado a gusto del usuario, pues ahora ya tiene a su alcance la información de correspondencia de una actividad.

4.4.3. Desarrollo de la interfaz para la gestión de archivos LCA.

Dentro de los planteamientos de la aplicación, se requirió la implementación de una interfaz de usuario para el proceso de ingreso de archivos LCA. Esta interfaz está enfocada al requisito funcional de *gestión de la base de datos*, la cuál como su nombre lo indica, gestiona el proceso de lectura y filtrado de los archivos de LCA, para que se pueda tener acceso a la Meta información y a su vez publicarla en la base de datos INTI.

Para el desarrollo de esta fase, se implementa el proyecto con el mismo framework de desarrollo web: *Django*. A continuación se detalla el proceso de desarrollo de la interfaz, que inicia con la descripción del envío del archivo LCA. Luego, continúa con el tratamiento que se da al archivo publicado. Finalmente, se presenta una breve explicación del proceso de validación de datos para la posterior publicación de los mismos, todo esto por medio de la API.

4.4.3.1. Publicación de los archivos.

El primer paso para este desarrollo es tener acceso a la interfaz de usuario donde se carga la información requerida. Tal interfaz, como se puede ver en la *Fig.19,* tiene dos variables de entrada manejadas por un formulario html que se acciona con el uso del botón. Las entradas son el archivo como tal, que debe ser subida en formato *.zip,* y la segunda entrada permite especificar la versión del LCA a la cual corresponde.

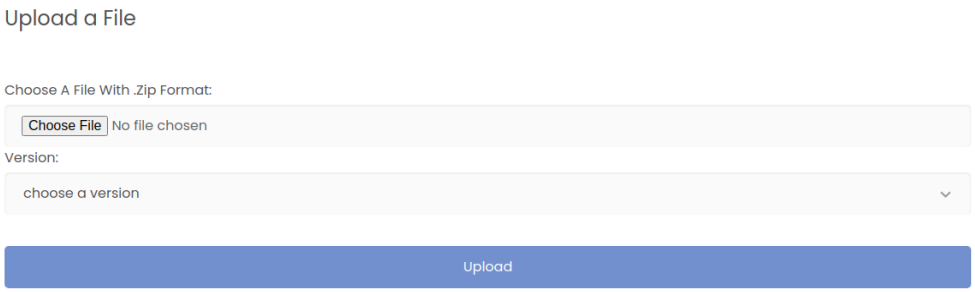


Fig. 19: Interfaz para el envío de archivos.

4.4.3.2. Tratamiento de los archivos.

Luego del envío del formulario y teniendo acceso al mismo, en la *Fig. 20* se puede visualizar un diagrama que sirve de guía para comprender el proceso de tratamiento de los datos. El usuario sube el archivo LCA seleccionando la versión que corresponde. Se ubica temporalmente el archivo comprimido en un directorio de la API. El sistema (API) se encarga del proceso ETL sobre dicho archivo (descomprimirlo, leerlo y obtener los atributos necesarios). Se almacena de una en una las tuplas dentro de la base de datos, y finalmente elimina el directorio con el archivo descomprimido y comprimido.

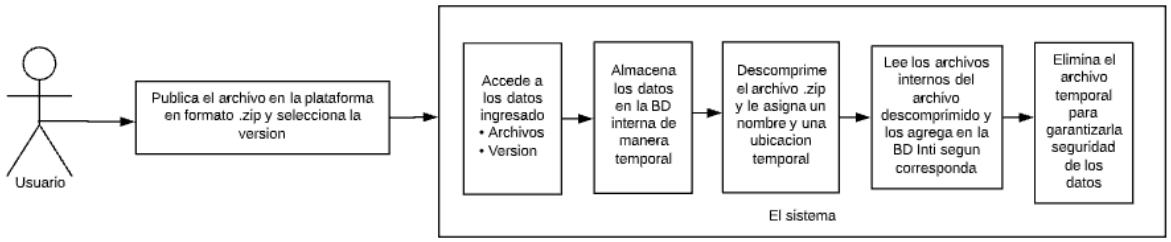


Fig. 20: Diagrama del proceso de tratamiento de los datos.

En lo referente a los detalles técnicos, la *Fig. 21* muestra el código del proceso de tratamiento de los datos. Así, lo primero que se hace, es acceder a las variables ingresadas (*Fig. 21.a*), asignarlas a un modelo previamente definido y guardarlo (*Fig. 21.b*). Esto con la finalidad que tiene el modelo de almacenar el archivo enviado en un directorio específico dentro del proyecto. Recordando que el archivo se sube en formato .zip, y se debe descomprimir (*Fig. 21.c*). En este caso se asigna un nuevo directorio temporal llamado */temp* que contará con el archivo descomprimido (*Fig. 21.d*) y con sus dos subcarpetas, que son en donde se encuentran los archivos que se debe leer su información y filtrarlos adecuadamente.

```
def upload_files(request):
    if request.method == "POST":
```

```
    #FETCHING THE FORM DATA
a   file_title = request.POST["file_title"]
    uploaded_file = request.FILES["uploaded_file"]
    # SAVING THE INFORMATION IN THE DATABASE
    document = models.Document(
b       title=file_title,
        uploadedFile=uploaded_file
    )
    document.save()
    #UNZIP THE UPLOADED FILE INTO A DIRECTORY
c   with ZipFile(directorio+document.uploadedFile.url, 'r') as zip:
        zip.extractall(directorio+'/temp')
        print('File is unzipped in temp folder')
    #NAME OF THE UNZIP FILE
    a = os.path.split(document.uploadedFile.url)
d   folder_name = os.path.splitext(a[1])
    route = dir_temp+folder_name+md
    routeDS = dir_temp+folder_name+ds
    '''post_data = {"username": "admin",
                    "password": "admin"}
    response = requests.post(
        "http://host:port/inti/api_generate_token", data=post_data)
    content = response.content
    token = content.decode('utf8').split('"')'''
    #METHODS TO READ THE FILE AND ADD TO THE DATABASE
    companies(route + 'Companies.xml')
    sources(route + 'Sources.xml')
    persons(route + 'Persons.xml')
    activity_name(route + 'ActivityNames.xml')
    geography(route + 'Geographies.xml')
e   unit(route + 'Geographies.xml')
    intermediateExchange(route + 'IntermediateExchanges.xml')
    system_model(route + 'SystemModels.xml')
    property(route + 'Properties.xml')
    id_version = search_version(document.title)
    activityIndexEntry(route + 'ActivityIndex.xml', id_version)
    leerActividadGenerica(routeDS, id_version)
    # DELETE THE TEMPORARY AND THE UPLOADED FILE
f   rmtree(dir_temp+'ecoinvent 3.6_apos_ecoSpold02')
    rmtree(directorio+'/media/uploaded_files')

documents = models.Document.objects.all()

return render(request, "core_index.html", context={
    "files": documents
})
```

Fig. 21: Función upload_files del archivo views.py.

Continuando con el tratamiento del archivo, se asignan las rutas para los directorios que están dentro del archivo LCA subido. Como se puede ver en el código de la *Fig. 22* se asignan variables globales. La variable *directorio*, es la ubicación del proyecto. La variable *dir_temp* depende de la ubicación asignada para descomprimir el archivo enviado por el formulario. Las variables *md* y *ds* corresponden a los directorios de la ubicación de las carpetas que están dentro del archivo LCA y que es necesario acceder y diferenciar.

Por ello, se cuenta con dos variables: *route* y *routeDS,* las cuales corresponden al directorio de la carpeta */MasterData* y */datasets,* respectivamente.

```
directorio = '/home/T_projects/my_env/DjangoFileUpload'
dir_temp = '/home/T_projects/my_env/DjangoFileUpload/temp/'
```

```
md = '/MasterData/'
ds = '/datasets/'
```

Fig. 22: Sección de código del archivo views.py

Finalmente, cómo se ve en el código de la *Fig. 21.e,* se llevan a cabo una serie de métodos para la lectura de los diferentes archivos. Estas funciones se llevan a cabo de manera ordenada siguiendo la estructura de poblado para la base de datos. Todas las funciones reciben como parámetro la ubicación del directorio con el nombre del archivo a acceder. Cada una de ellas tiene un proceso de validación que será descrito con mayor detalle en la Sección 4.4.3.3.

La función *leerActividadGenerica* se encuentra en el directorio */datasets/* que contiene una serie de archivos en formato *.spold*. En este caso en particular se renombran los archivos a un formato *.xml* como se puede ver en la parte de código de la *Fig. 23*.

```
def leerActividadGenerica(route, version):
    for archivo in os.listdir(route):
        cadena_archivo = archivo
        cortar = cadena_archivo.split('_')
        os.rename(route + cadena_archivo, route + str(cortar[0]) + ".xml")
```

Fig. 23: Sección de código inicial de la Función leerActividadGenerica del archivo views.py

Por otro lado, para las funciones: *companies()*, *sources()*, *persons()*, *activity_name()*, *geography()*, *unit()*, *intermediateExchange()*, *system_model()*, *property()* y *activityIndexEntry()*, que se encuentran en el directorio */MasterData/*, todos los archivos en este directorio están en formato *.xml.* Por lo tanto, la lectura de estos y los mencionados con anterioridad será similar al ejemplo de la función que se puede ver en la *Fig. 24*. Todas las funciones tienen el mismo método de lectura del archivo *.xml*, asignando a nuevas variables las correspondientes con los atributos y elementos del *.xml*.

```
def companies(route):
    xmlReader = minidom.parse(route)
    companies = xmlReader.getElementsByTagName("company")
    for company in companies:
        id = company.getAttribute("id")
        code = company.getAttribute("code")
        website = company.getAttribute("website")
        if len(company.getElementsByTagName("name")) != 0:
            try:
                name = company.getElementsByTagName("name")[0].firstChild.data
            except AttributeError:
                name = "not name provided"
        else:
            name = "not name provided"
        if len(company.getElementsByTagName("comment")) != 0:
            try:
                comment = company.getElementsByTagName("comment")[0].firstChild.data
            except AttributeError:
                comment = "not comment provided"
        else:
            comment = "not comment provided"
```

```
        '''GET THE COMPANY ID '''
        response = requests.get('http://host:port/inti/companies/'+id)
        if response.status_code == 404:
            post_data = {"id": id,
                         "code": code,
                         "name": name,
                         "website": website,
                         "comment": comment}
            response = requests.post(
                "http://host:port/inti/companies/", data=post_data)
        else:
            if response.status_code == 200:
                print("already exist")
    return
```

Fig. 24: Función companies del archivo views.py

Una vez finalizado el proceso de lectura de todas las funciones, cómo se ve en el código de la *Fig. 21.f,* se procede a eliminar el directorio temporal y el archivo subido. Esto para mantener la privacidad e integridad de los datos, al no tenerlos dentro del servidor como archivos, sino solo las variables necesarias dentro de la base de datos.

4.4.3.3. Proceso de validación de datos.

Este es un proceso que se lleva a cabo cada vez que se accede a las tuplas de datos que contienen los archivos *.xml*. El funcionamiento general se muestra en la *Fig. 25* e inicia con el acceso al archivo para leer la primera tupla. Luego, se procede a validar si dicha tupla existe en la base de datos INTI. Si no existe la tupla se procede a insertar dicha tupla y se pasa a leer la siguiente tupla. Si la tupla existe en la base de datos, se procede a leer directamente la siguiente tupla. Y con eso inicia nuevamente el proceso hasta que se lean todas las tuplas del archivo.

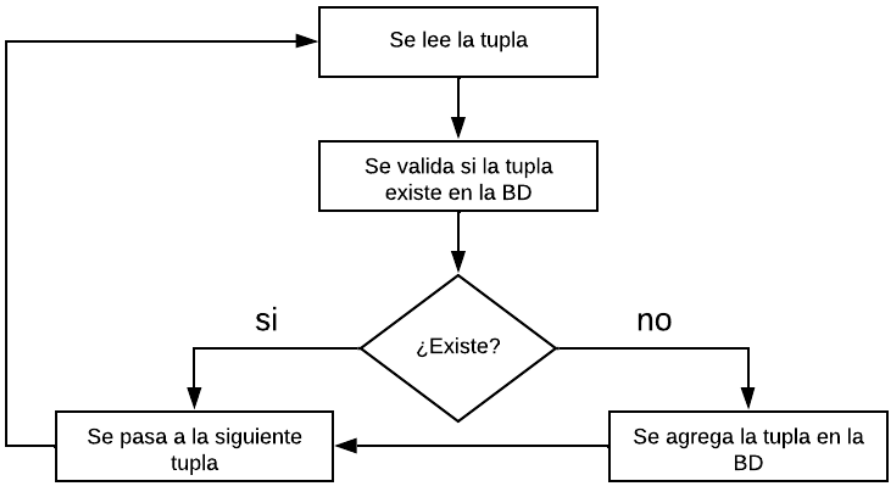


Fig. 25: Diagrama del proceso de validación de datos

En lo referente a la implementación técnica de la validación, cada que se leen las tuplas de datos se hace una consulta por medio del método *get* a la API para validar si existe dicha tupla como se puede ver en la *Fig. 26*. El resultado que se obtiene de dicha consulta a la BD, indica si la tupla ya existe en la base de datos, o si es el caso contrario se envía la tupla por medio del método *post* para su inserción en la base de datos.

```
.
.
.
response = requests.get('http://host:port/inti/companies/'+id)
if response.status_code == 404:
     post_data = {"id": id,
                  "code": code,
                  "name": name,
                  "website": website,
                  "comment": comment}
     response = requests.post("http://host:port/inti/companies/", data=post_data)
else:
     if response.status_code == 200:
          print("already exist")
.
.
.
```

Fig. 26: Sección de codigo del archivo views.py dónde se puede visualizar el método GET y POST con la API

Cabe recalcar que, este es solo un ejemplo de las múltiples validaciones que se realizan dentro del código, pues también existen consultas personalizadas para el ingreso de datos con la API. Tal es el siguiente caso, como se puede ver en la *Fig. 27,* que se hace por medio del método *post* la consulta a la API, enviando las variables de las cuales se quiere obtener una respuesta específica de la consulta a la base de datos.

```
.
.
.
post_data = {"activity_index_id": activity_index_id,
             "geography_id": geography_id,
             "start_date": start_date,
             "end_date": end_date,
             "special_activity_type": special_activity_type,
             "system_model_id": system_model_id}
response = requests.post("http://127.0.0.1:8000/inti/request_activityIndex/",
             data=post_data)
al = response.json()
content = al.get('response')
if content == '0':
     '''PUT IN TABLE activity_index'''
     post_data = {"id": activity_index_id,
                  "start_date": start_date,
                  "end_date": end_date,
                  "special_activity_type": special_activity_type,
                  "geography": geography_id,
                  "system_model": system_model_id}
     response = requests.post("http://127.0.0.1:8000/inti/activities_index/",
                  data=post_data)
     '''PUT IN TABLE version_name_index'''
     post_data = {"activity_index": activity_index_id,
```

```
                    "activity_name": activity_name_id,
                    "version": version}
        response = requests.post("http://127.0.0.1:8000/inti/version_name_indexes/",
                    data=post_data)
.
.
.
```

Fig. 27: Sección de código del archivo views.py dónde se puede visualizar el método POST para validar con la API.

4.4.3.4. Interfaz de usuario (User Interface).

Dentro del proceso del desarrollo de la Interfaz de Usuario (UI), se toma en cuenta el acceso a la parte gráfica del prototipo y por ende también se considera la confidencialidad del mismo. Como se mencionó con anterioridad, es de suma importancia velar por la seguridad de los datos, por lo que para ingresar a la información mostrada en la UI del prototipo, se debe iniciar sesión con un usuario por medio de la vista de la *Fig. 28.a*. Esta vista está validada para que no ingrese al sistema si es que las credenciales ingresadas no son las correctas como se puede ver en la *Fig. 28.b*.

Para fines informativos, se utiliza como ejemplo al usuario administrador, pues tiene acceso a todas las vistas de la UI.

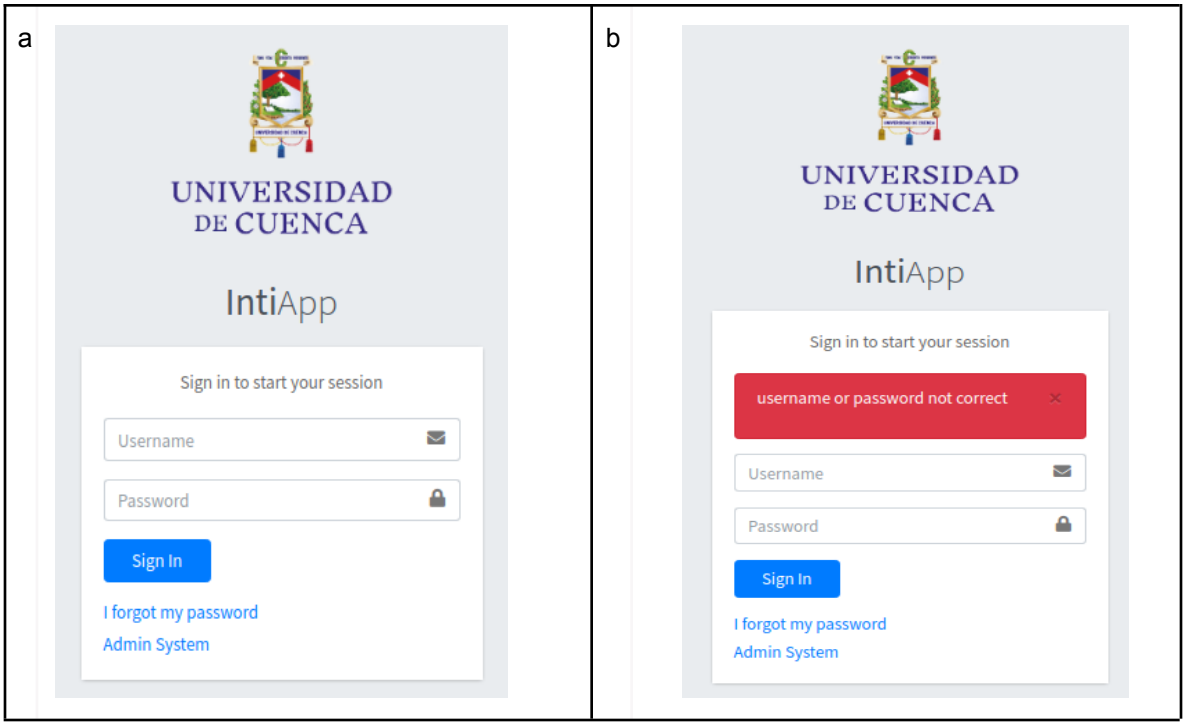


Fig. 28: Vista del login de la UI.

Concretamente la UI trabaja con código HTML, consumiendo las URLs de las consultas personalizadas, que fueron implementadas en la sección 4.4.2.3. con la finalidad de construir gráficamente el ingreso de las variables y las tablas de respuesta a partir del JSON que generan. Por lo tanto, como se puede ver en la *Fig. 29.a.* se tiene un extracto del código HTML encargado de construir la cabecera de la página, y la *Fig. 29.b.* contiene el código que construye la estructura del filtro con un método *for* en un tipo de botón *selectpicker*. A Continuación, en la *Fig. 29.c.* está la parte de código encargada de dar nombre a la tabla de manera que el usuario la pueda identificar en la UI y de la misma forma la *Fig. 29.d.* muestra el extracto de código encargado de estructurar la tabla, y asignándole un identificador único que sirve de referencia para que el extracto de código de la *Fig. 29.e* pueda hacer referencia a dicha tabla. Finalmente como se puede ver en el extracto de código de la *Fig. 29.f* se tiene la estructura de reset de la tabla para que cuando se ingrese otro tipo de variable en el filtro (*Fig. 29.b*), se obtenga la nueva consulta.

```
.
.
.
 |¯<div class="col-sm-6">
a|     <h1>Similar Names Activities</h1>
 |_</div>
.
.
.
 |¯<div class="card-body">
 |    <div class="row-fluid">
 |         <center>
 |              <label>Activity Name:</label>
 |              <select class="selectpicker" data-show-subtext="true"
 |                        data-live-search="true" id="activity" name="activity">
b|                   {% for activity in activities %}
 |                        <option>{{ activity.activity_name }}</option>
 |                   {% endfor %}
 |              </select>
 |         </center>
 |    </div>
 |_</div>
.
.
.
 |¯<div class="card-header">
c|         <h3 class="card-title">DataTable with default features</h3>
 |_</div>
 |=<div class="card-body">
 |    <table id="activitiesTable" class="table table-bordered table-striped">
 |         <thead>
 |              <tr>
 |                   <th>Activity Index Id</th>
 |                   <th>Activity Name</th>
 |              </tr>
 |         </thead>
d|         <tbody>
 |         </tbody>
 |         <tfoot>
 |              <tr>
 |                   <th>Activity Index Id</th>
 |                   <th>Activity Name</th>
 |              </tr>
```

```
 |         </tfoot>
 |    </table>
 |_</div>
.
.
.
 |¯<script>
 |    function activitiesList(activity_name){
 |    $.ajax({
 |         url: "/inti/activities/similar_names/?name="+activity_name,
 |         type: "get",
 |         dataType: "json",
 |         cache: false,
 |         success: function(response){
 |              console.log(response);
 |              $('#activitiesTable tbody').html("");
 |              for(let i = 0;i < response.length; i++){
 |                   let fila = '<tr>';
e|                   fila += '<td>' + response[i]['activity index id'] + '</td>';
 |                   fila += '<td>' + response[i]['activity name'] + '</td>';
 |                   fila += '</tr>';
 |                   $('#activitiesTable tbody').append(fila);
 |              }
 |              $('#activitiesTable').DataTable({
 |              "responsive": true, "lengthChange": false, "autoWidth": false,
 |              "buttons": ["copy", "csv", "excel", "pdf", "print", "colvis"]
 |         }).buttons().container().appendTo('#activitiesTable_wrapper
.col-md-6:eq(0)');
 |         },
 |         error: function(response){
 |              console.log("error");
 |         }
 |    });
 |    }
 |_</script>
.
.
.
 |¯<script>
 |    var select = document.getElementById('activity');
 |    select.addEventListener('change',
 |    function(){
 |         $('#activitiesTable').DataTable().destroy();
f|         var selectedOption = this.options[select.selectedIndex];
 |         console.log(selectedOption.value + ': ' + selectedOption.text);
          activitiesList(selectedOption.text)
 |    });
 |_</script>
.
.
.
```

Fig. 29: Secciones de código del archivo similar_names.html dónde se puede visualizar la estructura de una de las vistas de las consultas personalizadas en la UI.

Así mismo, como se puede ver en la *Fig. 30.a*, se tiene la vista gráfica generada a partir de las secciones de código *a, b, c, d* de la *Fig. 29.* Así mismo como indica la *Fig. 30.b,* se realiza una consulta, y se obtiene su resultado en una tabla estructurada que corresponde a las secciones de código *e, f* de la *Fig. 29.*

Fig. 30: Vista de la UI para realizar y obtener la respuesta a la consulta personalizada "similar names".

Por otro lado también cabe destacar la implementación de un dashboard dentro del desarrollo del prototipo, con la finalidad de analizar los resultados obtenidos en diferentes consultas que se tienen registradas en la base de datos INTI. Como ejemplo, además de las tablas de consultas personalizadas se generan gráficas como se puede ver en la *Fig. 31.a.* Esta señala el número de actividades disponibles en la base de datos INTI, y agrupadas por versión. Proceso similar sucede en la *Fig. 31.b* que indica en manera individual esta métrica, es decir, indica el número de actividades disponibles por versiones. Por otro lado, como señala la *Fig. 31.c*, después de ingresar el filtro correspondiente a indicar la

versión por la cual se desea agrupar la gráfica, indica el número de actividades segmentadas por año durante los últimos 10 años.

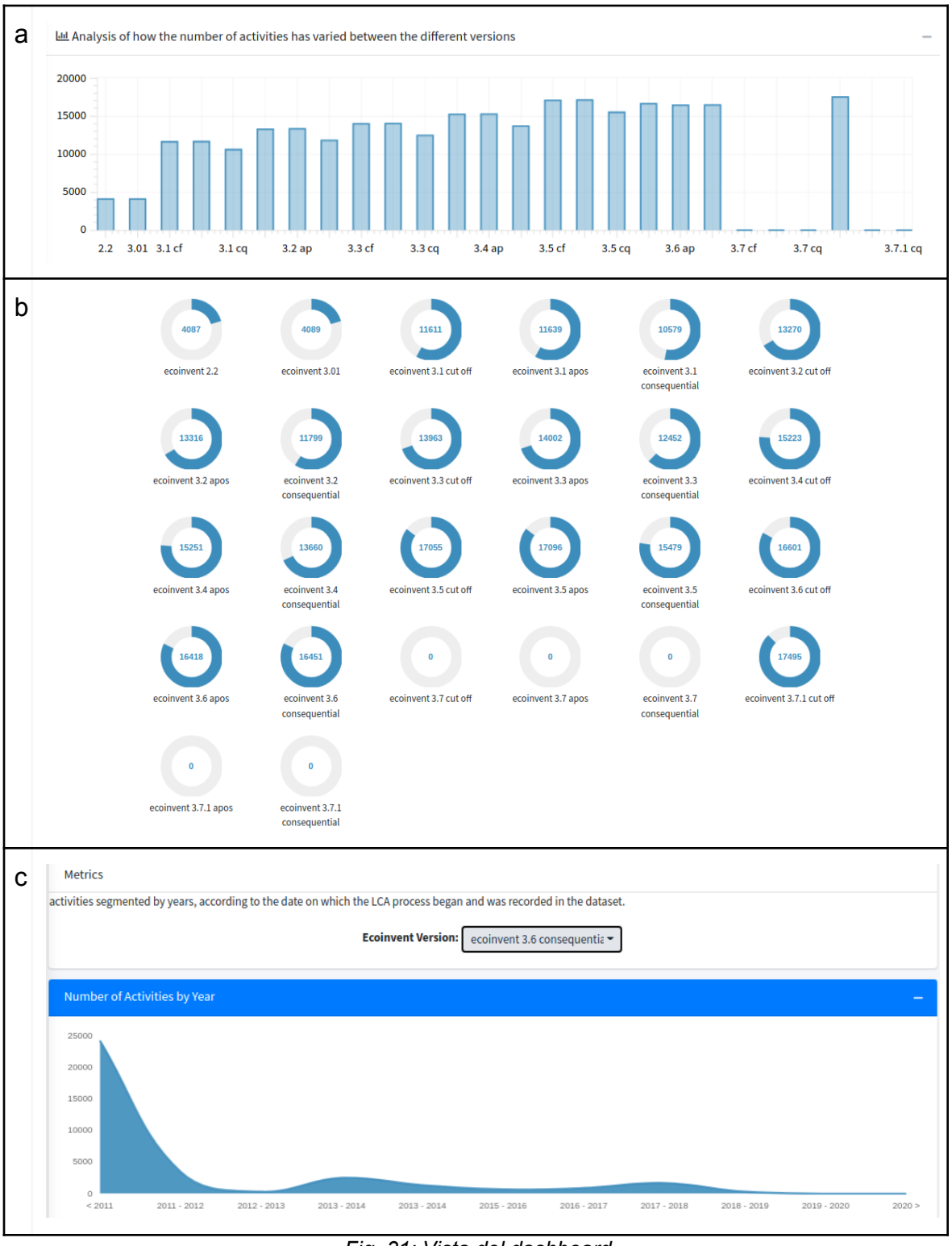


Fig. 31: Vista del dashboard

Para concluir con este apartado, el desarrollo de esta interfaz de usuario brinda una funcionalidad eficiente con respecto al proyecto de titulación. Debido a que se le brinda al usuario una forma, visualmente atractiva, para consultar los datos que se están manejando a través de la API. En este caso ejemplo, generando las consultas personalizadas y mostrandolas a partir de tablas dinámicas, que muestran los atributos de las diferentes consultas. De la misma forma, una parte importante que cumple la Interfaz de Usuario, es como se mencionó al inicio de esta sección, respetar la confidencialidad de los datos. Por ello se maneja un usuario administrador que tenga acceso a todos los datos sin restricciones. Con esto se pueden agregar más usuarios, con diferentes credenciales de interacción en cuanto a creación, lectura, y actualización de los datos de la base de datos INTI. El usuario final, ahora puede analizar los datos que le brindan las consultas personalizadas de una manera visual y comprensible a simple vista, ya que tiene la posibilidad navegar a través de las pestañas y realizar las diferentes consultas alternando los filtros. De igual manera, como parte de la exportación de los datos generados, se tiene la opción de descarga de la consulta en un archivo de texto con formato .csv, para ser procesada en un futuro y de valor agregado al análisis que pueda hacer el usuario.

5. Resultados y Discusión:

La finalidad del proyecto de titulación fue implementar una API que de soporte a la gestión de conjuntos de datos de inventarios de ciclo de vida. Con el trabajo de titulación se desea garantizar que su producto final satisfaga las necesidades del cliente y por ende recopilar nuevas ideas para garantizar que todos los aspectos del producto y su lanzamiento sean perfectos. Para lograrlo, se implementa un test de usabilidad, que se puede revisar en la sección 5.3, y pruebas de concepto sobre el desarrollo del trabajo de titulación, para medir las expectativas del cliente (Anexo 3), hacer ajustes y lanzar el prototipo con éxito.

5.1. Pruebas e Interoperabilidad.

En función de las cuatro fases de la prueba de concepto para ejecutarse con éxito (definición, preparación, ejecución y análisis), se consiguió el siguiente resultado sobre la implementación del prototipo. Esta prueba se realizó según el conjunto de requerimientos iniciales y evolucionó según nuevas modificaciones sugeridas por los Stakeholders durante las diferentes iteraciones, tanto en el modelado de la base de datos como en el modelado de la API.

- Diagrama ER de la base de datos

  En la *Fig. 32* se observa el modelo de datos conceptuales inicial de la base de datos relacional. En los requerimientos iniciales este diagrama ER no almacenaba la versión, no hacía diferencia entre los diferentes datasets, o la conexión entre diferentes versiones, pero durante una de las iteraciones/sprints se sugirió que se agreguen las entidades pertinentes, para mejorar la calidad de información que se estaría brindando a futuro.

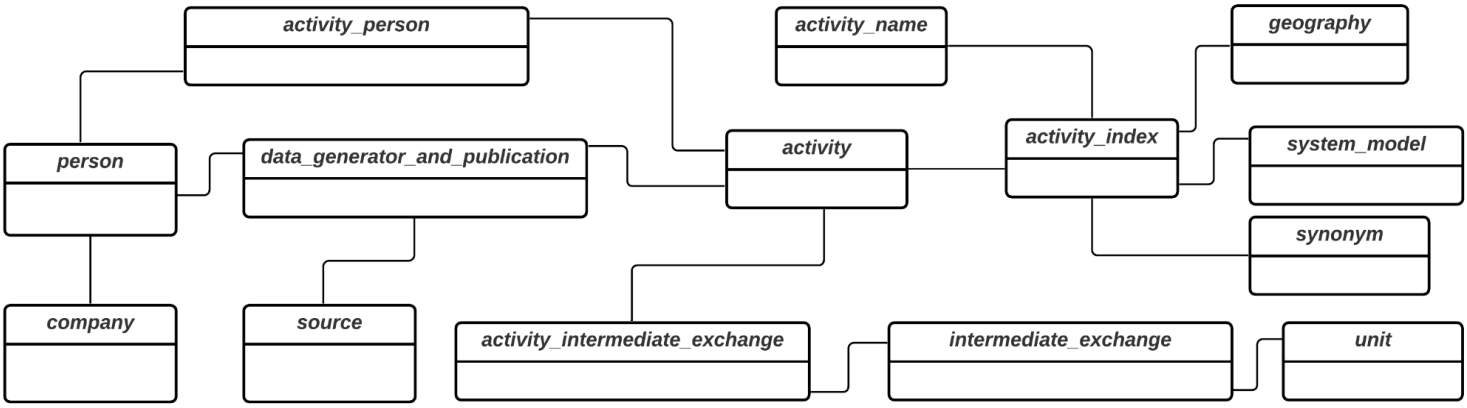


Fig. 32: Primer Modelo de datos conceptuales de INTI.

De esta manera, como se pudo ver en la *Fig. 10*, en el modelo de datos lógico, o diagrama ER, se puede almacenar y diferenciar entre versiones de datasets, analizar la correspondencia de versiones, y distinguir la información de cada versión, lo cual es una mejora importante a mencionar.

- Modelado de la API

Se maneja el control de errores con mensajes personalizados y con códigos de estado de *HTTP* pertinentes para cada caso. Esto fue necesario para implementar consultas personalizadas sobre la API. Inicialmente, el prototipo del sistema no podía filtrar la información. Por sugerencia de los Stakeholders, se reestructuró la API para que el usuario pueda filtrar su consulta personalizada.

En la *Fig. 33* se observa una de las vistas de los modelos de dichas consultas personalizadas. Para este ejemplo se desea conocer las actividades en base a una versión definida por el usuario. Se tiene más detalle de las consultas personalizadas programadas en la Sección 4.4.2.3 en la Tabla 14.


```
{
    "response": "0",
    "response1": [
        {
            "activity id": "0021a538-9534-4fd4-835e-a2fa8f76bb9d",
            "activity name": "market for stone wool, packed  ",
            "version": "ecoinvent 3.6 cut off "
        },
        {
            "activity id": "002477cd-70fa-49c1-afc3-c372abbcf72b",
            "activity name": "market for gas motor, mini CHP plant ",
            "version": "ecoinvent 3.6 cut off "
        },
        {
            "activity id": "002510a3-dcbb-4fb9-85f3-54e72b794bff",
            "activity name": "market for refractory material, acid ",
            "version": "ecoinvent 3.6 cut off "
        },
        {
            "activity id": "002fa54c-1829-4231-82ed-47e35a6745d4",
            "activity name": "market for mischmetal ",
            "version": "ecoinvent 3.6 cut off "
        },
        {
            "activity id": "003682ba-f70c-474f-ba5f-8203d2d00512",
            "activity name": "heat and power co-generation, wood chips, 6667 kW",
            "version": "ecoinvent 3.6 cut off "
        },
        {
            "activity id": "003a2fba-a774-4ff8-b8da-3adb2e14aeba",
            "activity name": "market for operation, housing system, pig, fully-slatted floor, per pig place  ",
```


Fig. 33: Consulta personalizada a la API.

Gracias al desarrollo ágil y el uso de las pruebas de concepto, se consiguió la evaluación de varias secciones del prototipo informático para la toma de decisiones en el desarrollo del sistema completo.

En este trabajo de titulación se presentó una prueba de concepto que ayuda a mejorar el proceso para el diseño y desarrollo de un prototipo funcional. La constante retroalimentación permitió un desarrollo centrado en las necesidades del usuario. Sin embargo, hay áreas de oportunidad y de crecimiento que no han

podido ser incluidas en esta versión del prototipo, pero se detallarán más adelante como trabajos futuros en la sección 7.

5.2. Reporte de Resultados.

Para el trabajo de titulación se implementó una API que da soporte y estructura a los archivos LCA para que el usuario final pueda visualizarlos de manera ordenada. Además, se implementó el análisis y estructuración de una Base de Datos Relacional que modela la estructura del formato de archivos LCA. Y, finalmente, se construyó una interfaz de usuario que consume la API y pueda plasmar la información recopilada.

Con este contexto, la API permite el proceso de lectura de los archivos LCA en su formato original, para luego del tratamiento adecuado, almacenar los datos en la base de datos previamente implementada. Se pobló la base de datos con archivos de formato EcoSpold 2 en cuatro versiones:

- *ecoinvent 3.6_apos_ecoSpold02*
- *ecoinvent 3.6_consequential_ecoSpold02*
- *ecoinvent 3.6_cut-off_ecoSpold02* y
- *ecoinvent 3.7.1_cutoff_ecoSpold02*.

Como parte de los objetivos del trabajo se tenía la propuesta de probar el prototipo informático con un caso de estudio práctico. En función de esto, se obtuvieron los resultados ejemplo que se detallan a continuación.

Tomando en cuenta que una vez almacenados los datasets de las versiones mencionadas, se puede indagar entre sus relaciones y responder diferentes preguntas estándar (*Fig.32*), que se pueden filtrar con el id correspondiente de las siguientes tablas:

- Actividades.
- Versiones.
- Compañías.
- Personas.
- Geografías.
- Unidades.
- Intercambios intermedios de actividades.
- Nombres de actividades.
- Personas de una actividad.
- Generador de datos y publicaciones.
- Intercambios intermedios.

- Propiedades.
- Recursos.
- Modelos del sistema.

El proceso para las consultas estándar, es similar para todas ellas, en donde se ingresa el id como filtro y se obtiene la tupla correspondiente como respuesta. A manera explicativa, para la tabla índice de actividades, como se puede ver en la *Fig. 34.a* se obtienen más de *264,000* tuplas, por lo que, al agregar esta funcionalidad de filtrar por el identificador, como se puede ver en la *Fig. 34.b*, se obtiene toda la información correspondiente a dicha variable.

a

b

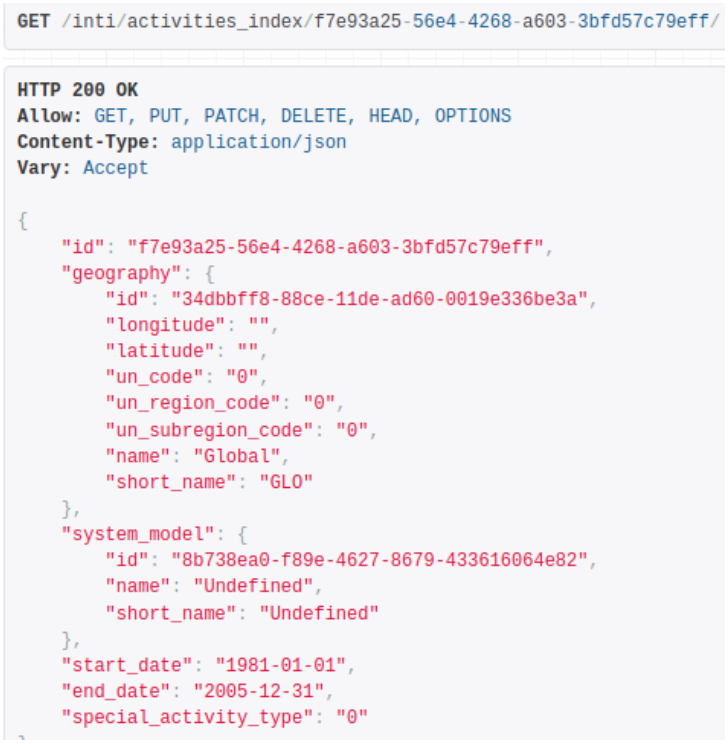


Fig. 34.Conocer los flujos de referencia asociado a una actividad. a. Captura de pantalla de las variables de entrada de la consulta. b. Captura de pantalla de la salida de la consulta.

Lo mismo sucede con las demás tablas, se puede filtrar mediante su identificador y obtener la información correspondiente de una única tabla deseada.

Tomando en cuenta la existencia de las consultas estándar, también están las consultas personalizadas (*Fig.33*) para los usuarios. Estas consultas trabajan directamente con las tablas que se acaban de mencionar para generar consultas que responden a las preguntas descritas en la *Tabla 14*, indicadas a continuación:

- Ver todas las actividades con el mismo nombre, filtrado por el nombre de la versión.
- Ver actividades con nombres similares, filtrado por el nombre o palabra clave del nombre de la actividad.
- Buscar similares entre los flujos de referencia, filtrado por el nombre de la versión y el nombre o palabra clave de la actividad.
- Ver actividades con la misma geografía, filtrado por el id de la geografía.
- Todos los flujos de referencia asociados con una actividad, filtrado por el id de la actividad y/o por el nombre de la versión.
- Actividades que tienen el mismo flujo de referencia, filtrado por el id del intermediate exchange y/o por el nombre de la versión.
- Referencias asociadas a una actividad, filtrado por el id de la actividad.
- Productores asociados a un intercambio intermedio, filtrado por el id del intermediate exchange.
- Personas asociadas con una actividad, filtrado por el id de la actividad.
- Actividades asociadas con una persona, filtrado por el id de la persona.

El proceso es similar para todas las consultas personalizadas, en donde se ingresan las variables como filtros y se obtiene la respuesta correspondiente. A manera explicativa, una de ellas implicaba conocer todos los flujos de referencia asociados con una actividad en particular. Como se puede ver en la *Fig. 34.a* se ingresa como variable al nombre de la versión y el nombre de la actividad. Luego, en la *Fig. 34.b* se aprecia el resultado de un extracto del formato JSON de estructuración de la respuesta de la consulta realizada; y, como se puede ver en la *Fig. 34.c,* se aprecia el resultado de las primeras 25 tuplas de la consulta realizada.

a

b


```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "intermediate exchange id": "7502366c-d1ac-46a9-9ae6-0f1b05fd6cb8",
        "intermediate exchange name": "trichloroacetic acid",
        "version": "ecoinvent 3.6 cut off"
    },
    {
        "intermediate exchange id": "c40263a4-8ee0-4162-8785-59f0c1d793d3",
        "intermediate exchange name": "brass removed by turning, primarily dressing, conventional",
        "version": "ecoinvent 3.6 cut off"
    },
    {
        "intermediate exchange id": "66c93e71-f32b-4591-901c-55395db5c132",
        "intermediate exchange name": "electricity, high voltage",
        "version": "ecoinvent 3.6 cut off"
    },
    {
        "intermediate exchange id": "3d413f1b-247d-4c9a-8412-5a19c1d67ab2",
        "intermediate exchange name": "cement, blast furnace slag 81-95%",
        "version": "ecoinvent 3.6 cut off"
    },
    {
        "intermediate exchange id": "9b9edcf3-0539-4642-9516-0df642a5c41a",
        "intermediate exchange name": "land tenure, arable land, measured as carbon net primary productivity, perennial crop",
        "version": "ecoinvent 3.6 cut off"
    },
    {
        "intermediate exchange id": "7805ee60-8cdd-4e8b-9d35-217e0318f955",
        "intermediate exchange name": "petrol, 5% ethanol by volume from biomass",
        "version": "ecoinvent 3.6 cut off"
    },
    {
```


c

Intermediate Exchange Id	Intermediate Exchange Name
0451c9ba-5f09-4e23-a8b4-3d7ee441a6fc	sludge, pig iron production
0b404a7a-c140-4b1b-a555-a7082a6012ef	used vegetable cooking oil
1b30b018-ac39-41f4-a9e0-92057eef8bb8	waste mineral oil
1d9db9e3-2ddc-42cb-8bcb-eb6d259aee55	used tyre
1f41586d-0d8a-4c7c-8473-dd8351bab538	clinker
2bf7e6c9-e683-4f7f-b6dd-0a9724d81352	building, budget hotel
3a91cb7d-843b-495d-9973-33ba5efee877	inert waste
3b01838a-28d0-43f5-8024-3fad9a5392c2	sewage sludge, dried
3f5f5aef-3833-43d4-8b81-44123105afd7	irrigation
6326adcd-063f-4c42-ae99-973bd21cbb20	graphite
Intermediate Exchange Id	**Intermediate Exchange Name**

Showing 1 to 10 of 25 entries

Previous 1 2 3 Next

Fig. 35.Conocer los flujos de referencia asociado a una actividad. a. Captura de pantalla de las variables de entrada de la consulta. b. Captura de pantalla de la salida de la consulta.

5.3. Dificultades encontradas.

Al desarrollar el prototipo informático de soporte para gestionar conjuntos de datos de inventarios de ciclo de vida, se detectaron problemas relacionados con la estructura de dichos datos. Se conocen varios sistemas de software dedicados al LCA tanto públicos como privados, pero cada uno de ellos maneja un formato diferente que se adapte a sus requerimientos. Por lo tanto, la dificultad estuvo en que no se ha definido hasta el momento un formato estándar para publicar los resultados de LCA, lo que lleva a una baja confidencialidad de interpretación de los datos, siendo de diferentes formatos.

Al trabajar con el formato de ecoinvent, se debe tener en cuenta la confidencialidad del uso de los mismos. Este trabajo de titulación se desarrolló con la finalidad de ser un prototipo de código abierto por lo que manejar información sensible es una tarea delicada debido a que no se puede mostrar cierta información.

La discusión en cuanto al análisis de datos de inventario de ciclo de vida en el Ecuador, inició con el hecho que no se contaba con datos en el formato adecuado. Los datasets corresponden al análisis de ciclo de vida de energía hidroeléctrica y termoeléctrica. Estos archivos se consideraban LCA, pero su formato no se adapta a una estructura existente. Actualmente, y gracias al apoyo del desarrollador del software Antelope, se reestructuraron estos datasets a uno de los formatos existentes de LCA, para su uso un proyecto LCA diferente.

Lastimosamente, al momento de analizar la factibilidad del proceso de inserción de estos dataset dentro del prototipo informático de la base de datos INTI, no se pudo dar con éxito, debido a que el formato OpenLCA tiene una estructura diferente al formato ecoinvent. Esto está implicado directamente a que el modelo de procesamiento de lectura de archivos que se tiene actualmente en la API, está basado en el formato de la versión EcoSpold 2, y como se mencionó anteriormente, este es el formato más completo en cuanto a estructura, pero no similar al otro.

5.4. Test de usabilidad.

Una vez se concluyó con la fase de desarrollo del prototipo informático, se procedió a poner en el servidor de la Universidad de Cuenca a modo de prueba dentro del dominio controlado. Se pudo acceder al prototipo a manera de test por los stakeholders definidos para el proyecto, a los cuales se les dió las credenciales

y el acceso para sus correspondientes pruebas y medición de resultados en un ambiente controlado.

La metodología que se utilizó para la evaluación es la denominada: respuesta individual, que consiste en asignar una serie de tareas a un usuario para que sean ejecutadas sobre el prototipo a evaluar. El instrumento de medición fue un formulario, para esto, el facilitador brinda las instrucciones a realizar y el usuario responde con las acciones que llevará a cabo, afirmando o no si fueron posibles realizarlas. De esta manera, se puede medir la forma en la que el usuario reacciona ante las indicaciones precisas de las tareas sobre el prototipo, detectando la usabilidad del prototipo en sí. Posteriormente, se pide retroalimentación al usuario evaluado, donde exponen sus dudas, sugerencias, pensamientos e ideas que tuvieron durante y después de la prueba de usabilidad.

Al realizar los estudios con los stakeholder, se pudo encontrar el beneficio para el uso entre la comunidad científica, debido a la falta de una herramienta de uso sencillo. Sin embargo, se encontraron algunos comentarios sobre la usabilidad, como por ejemplo:

- **Visibilidad:** Algunas páginas tenían poca información referente a la consulta que están respondiendo, por lo que se debería detallar más la descripción.
- **Enlaces ocultos:** Existen opciones importantes para el filtrado de información que son accesibles desde la API root.
- **Detalle de la información:** Las consultas realizadas, pueden mostrar más atributos que permita un mejor detalle y comprensión de la misma.
- **Agregar comentarios:** A los usuarios les gustaría agregar comentarios conforme a lo que obtuvieron para que sirva de referencia para otros usuarios futuros.

El prototipo informático INTI obtuvo una aceptable retroalimentación en el test de usabilidad. Aunque tuvo opiniones y comentarios diversos, en general se asegura que la plataforma tiene un gran potencial. Es cierto que existen algunos errores y algunas dificultades como se mencionaron en la sección 5.3. Sin embargo, casi todos los comentarios de mejora residen en la parte visual y son cambios que se en su mayoría se pueden realizar sin mayor inconveniente de desarrollo e implementación.

5.5. Discusión

Efectivamente, la implementación de este prototipo informático de soporte para gestionar conjuntos de datos de inventarios de ciclo de vida, ha sido un gran avance para el estudio de LCA. Si bien es cierto, y como se ha mencionado con anterioridad, la metadata está disponible para los usuarios que deseen analizar un LCI. Tomando en consideración que los archivos de metadata están comúnmente almacenados en varios archivos de texto y puede llegar a ser difícil su análisis, comprensión y/u obtención de relaciones entre los mismos datos.

Es por ello que para el contexto de LCA, implementar una estructura de almacenamiento que permita un fácil acceso a los metadatos, es de sumo provecho. Durante el desarrollo de esta propuesta de prototipo, y al realizar los diferentes análisis previos y de desarrollo, implementación y validación, se pudo comprobar el factor clave que generar una API que brinde acceso a este tipo de información, vuelve más eficiente el LCA para los diferentes usuarios. Dicho de otra manera, se puede consumir la API implementada para INTI, y obtener varios datos, relacionados entre sí y convertirlos en información de calidad y relevante para quién busca realizar un análisis sobre la metadata, es decir, se puede obtener diferentes datos o información al consumir las URLs provistas en este proyecto.

Esta es una mejora muy amplia para el estudio del LCA, ya que analizando el procesamiento sobre los archivos de metadata efectuado con herramientas de LCA, por ejemplo con Antelope, la lectura y transformación de datos de la metadata lleva más tiempo. Esto se afirma porque el proceso de carga de información, es menos eficiente debido a que se leen los archivos desde memoria interna, por lo tanto se necesita tener almacenado el archivo en en el mismo dispositivo que se está ejecutando el software. Además, el proceso de lectura es diferente, pues el software lee de un archivo en un archivo y no genera las diferentes conexiones entre los archivos para abstraer los datos más relevantes y generar información trascendente. Con esto se obtiene una mejora en el procesamiento, tanto en el de memoria, como en el de tiempo ya sea de ejecución o su contraparte de lectura y obtención de datos. Al implementar este prototipo, estamos garantizando que los usuarios no mantengan en la memoria interna de sus dispositivos los archivos referente a la Metadata, y que puedan recopilar información de una manera sumamente más rápida. Por la misma razón, estamos brindando la funcionalidad que puedan indagar sobre diferentes metadatos sin la necesidad de tenerlos en sus dispositivos. Tienen a disponibilidad consumir las URLs dedicadas a cada consulta, no sólo personalizada, sino también consulta estándar como se mencionó en el capítulo anterior.

Esto entonces se vuelve un punto muy relevante dentro de este trabajo de titulación, pues generar un prototipo informático de código abierto con la factibilidad de brindar acceso no solo a la metadata previamente almacenada en la base de datos INTI, sino también a brindar la opción a usuarios para que puedan insertar metadatos de interés en la base de datos, aumentando así la cantidad y calidad de datos que se puedan generar. Al mismo tiempo, se debe mencionar que, al generar una API que brinda datos de una forma más sencilla, rápida y útil para los usuarios, se le da valor agregado al producto. El mismo hecho de no tener que analizar archivo por archivo, línea por línea, dato por dato, da libertad al usuario para que pueda manejar los datos obtenidos por medio de la API. El usuario entonces tiene acceso a diferentes datos e información que podrá moldear a su gusto, teniendo a disponibilidad el código para que continúe agregando diferentes consultas personalizadas de ser el caso. Finalmente, cabe mencionar que el proceso de inserción de datos en la base de datos INTI, es un proceso lento debido a la gran cantidad (miles) de datos por versión que se deben extraer, transformar, validar y cargar en la misma. Por ello, aunque esta fase del procedimiento es demorada, y toma más tiempo procesar, a futuro se nota la mejora en el tiempo de procesamiento, pues la extracción de datos relacionados por medio de consultas se vuelve más rápida y sencilla. Esta sí es una mejora con respecto a tener que acceder a un solo archivo, hacer una extracción, y tener los datos necesarios, para luego acceder a otro archivo y así sucesivamente. Por lo tanto, implementar esta estructura de almacenamiento, aminora tiempos de ejecución en cuanto a la obtención de datos en particular gracias a las consultas personalizadas, brindando importancia y relevancia al producto final que se está entregando.

6. Conclusiones:

El objetivo general del trabajo de titulación fue desarrollar un prototipo informático de soporte para gestionar conjuntos de datos LCI representativos del contexto ecuatoriano. Es evidente que la base de datos INTI otorga un cimiento para la investigación y desarrollo para estudios posteriores enfocados en la gestión de información referente a LCA, que se centren en estructurar nuevos requerimientos y arquitecturas de sistemas de software que puedan extender los objetivos específicos planteados de nuestro proyecto.

Para concluir con nuestro primer objetivo, al revisar los conjuntos de datos de procesos de LCA, se desea conseguir información de calidad para la estructura de datos propuesta. Con estas revisiones se obtienen diferentes recursos por parte de los sistemas que proporcionan conjuntos de datos de LCA, tanto nacionales como internacionales. Sin embargo, nos dimos cuenta que existen requisitos complejos debido a la falta de estandarización de datos y el desarrollo actual de información por la existencia de diferentes formatos de estructuración. Así mismo, para dar solución a la problemática actual en cuanto a la generación de datos LCI, consideramos clave tomar en cuenta criterios de interoperabilidad entre bases de datos LCI previamente implementadas, así como aspectos de gestión de datos. Dicho esto, y en base a la experiencia obtenida durante el desarrollo de este prototipo, no se debe omitir la retroalimentación recibida de la parte interesada, para así asegurar la calidad del conjunto de información. Se deben construir fuentes de información que permitan integrar estos formatos diferentes y brindar valor a los datos que cada uno recolecta.

Adicionalmente, es fundamental concientizar el análisis previo y exhaustivo que se llevó a cabo para el prototipo informático. Gracias a este factor crucial de examinar las diferentes herramientas y software existentes, se facilitó el proceso de planificar el alcance del proyecto de titulación, de estructurar los requerimientos funcionales y no funcionales, para finalmente plantear una metodología correcta de desarrollo. En base a lo mencionado, con respecto al diseño y la implementación del prototipo informático, se utilizaron entornos de desarrollo de código abierto, en particular, el uso del lenguaje Python como base de desarrollo, debido a su fácil aprendizaje, acceso y alcance al incluir varios módulos que facilitaron el desarrollo del prototipo. Es importante destacar que, para esta implementación, se tuvo en cuenta los requerimientos obtenidos en la fase inicial del proyecto enfocados a la idea de la integración de este prototipo informático con software existente en cuanto a gestión de información LCA. En síntesis, los objetivos específicos de la tesis están plasmados en las pruebas realizadas al prototipo informático, que contiene conjuntos de datos LCI, y su resultados

satisfactorios para centralizar el manejo de dichos datos creando una posibilidad de estandarización nacional.

Finalmente, al obtener un análisis a nivel teórico, tecnológico y metodológico en cuanto a la estructuración y conformación de una base de datos LCI, ponemos en comparación nuestro prototipo con los software de código abierto existentes. En relación a INTI, no se busca reemplazar algún software, pero si decidimos brindar un nuevo tipo de información referente al LCA, abordando este análisis de una forma novedosa brindando acceso a la metainformación y al intercambio de versiones. La práctica de LCA está muy limitada por su dependencia hacia grandes bases de datos que carecen tanto de transparencia como de soporte técnico. Por lo tanto, nuestra solución de prototipo brinda accesibilidad a los usuarios interesados, aunque en la mayoría de los sistemas se menciona el acceso a este tipo de información, el usuario lo haría de forma manual. En cambio, se puede indicar que somos los primeros en estructurar este tipo de solución informática dirigida para los stakeholders de LCA, los cuales encontrarán una forma de analizar conjuntos de datos LCA de una forma amigable y personalizada a manera que encuentren datos que comparten propiedades en común con un conjunto de datos específico. Para concluir, a pesar de los desafíos atravesados en el proceso de educarnos desde un inicio y sin bases previas en el tema ambiental, este proyecto brindó un aporte mayor mayor para nuestro desarrollo como profesionales, al mismo tiempo que se comprende el alcance de los archivos LCA, y se incursiona al desarrollo tecnológico para facilitar la práctica de LCA.

7. Trabajos futuros.

El primer punto a trabajar a futuro es el de implementar un subsistema de gestión dentro de la interfaz de usuario, para crear una siguiente versión del prototipo en la que se pueda publicar los demás formatos disponibles y se puedan almacenar en la estructura de datos diseñada en este trabajo de titulación. Así mismo, se debe trabajar en la difusión a nivel nacional en el uso de este prototipo para que los posibles stakeholders futuros puedan implementar su funcionamiento y agregar valor a la metadata de los LCI. En efecto, un paso necesario para dicha difusión es la conexión entre INTI y AMARU, para generar valor y mejoras referente a la ingeniería de software de nuestro proyecto de titulación en el ámbito ecuatoriano. Debido a que este prototipo puede ser accedido por el software Amaru, y con ello también generar sus propias consultas personalizadas en referencia a la metainformación que se está dando acceso referente al LCA.

Finalmente, está el proceso de la mejora continua del prototipo. En cuanto a la API, se pueden optimizar las consultas personalizadas, agregando un mayor detalle y complementando a las actuales. Se recomienda un sistema de backups enfocado a la base de datos, con la finalidad de mantener un funcionamiento adecuado de la misma y un correcto manejo de la integridad de la información, así como la confidencialidad de datos. Con este contexto, también será importante analizar la factibilidad de la implementación de una base de datos que abarque los cuatro grupos de información de procesos LCI, como se menciona en la *Sección 1.2.* Es decir*,* no solo enfocada en la metadata, sino también en intercambios, cantidades y flujos de los procesos LCA, para lo cual se recomienda una nuevo SGBD, para que soporte una mayor cantidad de datos. Por último, es importante mencionar que en cuanto a la UI las mejoras que se pueden dar, están referenciadas al consumo de recursos que tiene, pues la gran cantidad de datos que se manejan implica que los procesos se ralentizan. Por lo tanto se debe encontrar una forma de aminorar tiempos de respuesta con el uso de contenedores, que puede ser motivo de un próximo trabajo. Al mismo tiempo se debe tomar en consideración la experiencia del usuario en cuanto a la usabilidad de la interfaz, con lo cuál se espera una mejora contínua enfocada a la funcionalidad del sistema y al fácil acceso a los módulos que él mismo ofrece.

8. Referencias Bibliográficas:

Amaru. (2022). http://amaru.io/

Arias, D., Albuja, S., Herrera, S., Pinto, D., Valenzuela, J., Vásquez, R., Viteri, A., Ruiz, E., Calle, C., Vergara, D., Vozmediano, G., Taramuel, D., Sarango, A., Baquero, G., Cruz, A., Sandoval, C., Fabara, A., Granda, Lady, Vélez, C., … Montalvo, D. (2019). *Informe de avance del cumplimiento de la Agenda 2030 para el Desarrollo Sostenible* (N.º 1; p. 245). https://www.planificacion.gob.ec/wp-content/uploads/downloads/2019/07/Informe-Avance-Agenda-2030-Ecuador-2019.pdf

Aurea. (2017, noviembre 17). *Desarrollo de base de datos*. aurea. https://aurea.es/desarrollo-base-de-datos/

Azapagic, A. (2017). A Life Cycle Approach to Measuring Sustainability. En M. A. Abraham (Ed.), *Encyclopedia of Sustainable Technologies* (pp. 71-79). Elsevier. https://doi.org/10.1016/B978-0-12-409548-9.10036-3

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001, marzo). *Manifiesto por el Desarrollo Ágil de Software*. https://agilemanifesto.org/iso/es/manifesto.html

Bodega Fernanz, J. (2019). *Aplicación de metodologías de creación de APIs en diferentes frameworks de desarrollo web utilizando elementos ORM - 3447*. 63.

Camps Paré, R., Casillas Santillán, L. A., Costal Costa, D., Gibert Ginestà, M., Martín Escofet, C., & Pérez Mora, O. (2005). *Bases de datos*. https://www.uoc.edu/pdf/masters/oficiales/img/913.pdf

Ccori, W. (2018, septiembre 7). *Los 10 patrones comunes de arquitectura de software*. Medium. https://medium.com/@maniakhitoccori/los-10-patrones-comunes-de-arquitectura-de-software-d8b9047edf0b

Chomkhamsri, K., Mungcharoen, T., & Yuvaniyama, C. (2017). 10-year experience with the Thai national LCI database: Case study of “refinery products”. *The International Journal of Life Cycle Assessment*, *22*(11), 1760-1770. https://doi.org/10.1007/s11367-016-1160-3

Ciroth, A. (2007). ICT for environment in life cycle applications openLCA — A new open source software for life cycle assessment. *The International Journal of*

Life Cycle Assessment, *12*(4), 209. https://doi.org/10.1065/lca2007.06.337

Ciroth, A. (2019). *Current challenges and the way forward*. 12.

Ciroth, A., Hildenbrand, J., Zamagni, A., & Geos, E. (2015). *Life Cycle Inventory Dataset Review Criteria*. https://doi.org/10.13140/RG.2.1.2383.4485

Cohn, M. (2009). *Succeeding with Agile: Software Development Using Scrum*.

Corporación Internacional de Aplicaciones Científicas. (2006). *Life Cycle Assessment: Principles and Practice.* https://cfpub.epa.gov/si/si_public_record_report.cfm?Lab=NRMRL&dirEntryId=155087

Creative Commons. (2021). *Application programming interface (API)*. 8.

ecoinvent. (2021). *Data Releases*. Ecoinvent. https://ecoinvent.org/the-ecoinvent-database/data-releases/

Fauzi, R. T., Lavoie, P., Sorelli, L., Heidari, M. D., & Amor, B. (2019). Exploring the Current Challenges and Opportunities of Life Cycle Sustainability Assessment. *Sustainability*, *11*(3), 636. https://doi.org/10.3390/su11030636

Galvez, S. (2017). *Tipos de Bases de Datos*. http://www.lcc.uma.es/~galvez/ftp/bdst/Tema2.pdf

Gómez Fuentes, M. del C. (2013). *Notas del curso Bases de Datos*. http://www.cua.uam.mx/pdfs/conoce/libroselec/Notas_del_curso_Bases_de_Datos.pdf

Gorschek, T., Garre, P., Larsson, S., & Wohlin, C. (2006). A Model for Technology Transfer in Practice. *IEEE Software*, *23*(6), 88-95. https://doi.org/10.1109/MS.2006.147

Hamilton, T. (2021, diciembre 25). *What is Test Driven Development (TDD)? Tutorial with Example*. https://www.guru99.com/test-driven-development.html

Hauschild, M. Z., Rosenbaum, R. K., & Olsen, S. I. (Eds.). (2018). *Life Cycle Assessment*. Springer International Publishing. https://doi.org/10.1007/978-3-319-56475-3

Ihobe, S. P. de G. A. (2017). *Guía para la aplicación conjunta de los Análisis de Ciclo de Vida Ambiental (LCA) y de Costes (LCC)*. https://www.euskadi.eus/contenidos/documentacion/guia_lca_lcc/es_def/adjuntos/Guia_%20aplicacion_conjunta_LCC_LCA_cast.pdf

Interfaces para los programas de aplicación (APIs). (2015). 24.

ISO 14044. (2006). *ISO 14044:2006(es), Gestión ambiental—Análisis del ciclo de vida—Requisitos y directrices*. https://www.iso.org/obp/ui/#iso:std:iso:14044:ed-1:v1:es

Jakl, M. (2005). *What is REST ? Representational State Transfer*. https://www.semanticscholar.org/paper/What-is-REST-Representational-State-Transfer-Jakl/77e2a1497e69be0a2aec49450599aa3a04b8821c

Klöpffer, W. (2008). Life cycle sustainability assessment of products. *The International Journal of Life Cycle Assessment*, *13*(2), 89. https://doi.org/10.1065/lca2008.02.376

Kuczenski, B. (2020). *Antelope LCA*. GitHub. https://github.com/AntelopeLCA

Kuczenski, B., Davis, C. B., Rivela, B., & Janowicz, K. (2016). Semantic catalogs for life cycle assessment data. *Journal of Cleaner Production*, *137*, 1109-1117. https://doi.org/10.1016/j.jclepro.2016.07.216

Kuczenski, Brandon. (2018). Disclosure of Product System Models in Life Cycle Assessment: Achieving Transparency and Privacy. Journal of Industrial Ecology. 10.1111/jiec.12810.

Kusche, O., Fazio, S., Zampori, L., European Commission, & Joint Research Centre. (2015). *Life Cycle Data Network handbook for users and data developers.* Publications Office.

Liebsch, T. (2020, abril 15). Life Cycle Assessment Software Tools—Overview & Comparison. *Ecochain*. https://ecochain.com/knowledge/life-cycle-assessment-software-overview-comparison/

Mariño, S. I., & Alfonzo, P. L. (2014). *Implementación de SCRUM en el diseño del proyecto del Trabajo Final de Aplicación*. *19*(4), 7.

Matthews, H., Hendrickson, C., & Matthews, D. (2014). *Life Cycle Assessment: Quantitative Approaches for Decisions that Matter*.

Miah, J. H., Griffiths, A., McNeill, R., Halvorson, S., Schenker, U., Espinoza-Orias, N., Morse, S., Yang, A., & Sadhukhan, J. (2018). A framework for increasing the availability of life cycle inventory data based on the role of multinational companies. *The International Journal of Life Cycle Assessment*, *23*(9), 1744-1760. https://doi.org/10.1007/s11367-017-1391-y

Moore, L. (2018, julio). *What is MySQL?* SearchOracle. https://searchoracle.techtarget.com/definition/MySQL

Mutel, C. (2016). *Brightway2 LCA framework*. https://brightway.dev/

Mutel, C. (2017). Brightway: An open source framework for Life Cycle Assessment. *The Journal of Open Source Software*, *2*(12), 236. https://doi.org/10.21105/joss.00236

Muthu, S. S. (2020). 2 - Ways of measuring the environmental impact of textile processing: An overview. En S. S. Muthu (Ed.), *Assessing the Environmental Impact of Textiles and the Clothing Supply Chain (Second Edition)* (pp. 33-56). Woodhead Publishing. https://doi.org/10.1016/B978-0-12-819783-7.00002-8

MySQL. (2021). *MySQL :: Por qué MySQL?* https://www.mysql.com/why-mysql/

OpenLCA. (2016). https://www.openlca.org/

Oracle. (2021). *Learn About the Latest Oracle Database*. https://www.oracle.com/in/database/technologies/

Orix Systems. (2015, julio 12). *¿Qué es un framework y para qué se utiliza?* https://www.orix.es/que-es-un-framework-y-para-que-se-utiliza

PNUMA. (2011). *Global guidance principles for life cycle assessment databases :* UNEP,. http://digitallibrary.un.org/record/785158

PostgreSQL. (2021a). *PostgreSQL: About*. https://www.postgresql.org/about/

PostgreSQL. (2021b, julio 1). *PostgreSQL Global Development*. PostgreSQL. https://www.postgresql.org/

Ramirez, A. D., Rivela, B., Boero, A., & Melendres, A. M. (2019). Lights and shadows of the environmental impacts of fossil-based electricity generation technologies: A contribution based on the Ecuadorian experience. *Energy Policy*, *125*, 467-477. https://doi.org/10.1016/j.enpol.2018.11.005

Reed, D. (2012). Life-Cycle Assessment in Government Policy in the United States. *Doctoral Dissertations*. https://trace.tennessee.edu/utk_graddiss/1394

REST. (2016). *REST 101: The Beginner's Guide to Using and Testing RESTful APIs*. https://static1.smartbear.co/smartbear/media/ebooks/rest-101.pdf

Slant. (2021). *Django vs Symfony detailed comparison as of 2021*. Slant. https://www.slant.co/versus/1746/3758/~django_vs_symfony

Sommerville, I. (2005). *Ingenieria del Software* (7ma.). PEARSON. http://zeus.inf.ucv.cl/~bcrawford/AULA_ICI_3242/Ingenieria%20del%20Software%207ma.%20Ed.%20-%20Ian%20Sommerville.pdf

Sonnemann, G., Vigon, B., Broadbent, C., Curran, M. A., Finkbeiner, M., Frischknecht, R., Inaba, A., Schanssema, A., Stevenson, M., Ugaya, C. M. L., Wang, H., Wolf, M.-A., & Valdivia, S. (2011). Process on "global guidance for LCA databases". *The International Journal of Life Cycle Assessment*, *16*(1), 95-97. https://doi.org/10.1007/s11367-010-0243-9

Tébar, E. (2020, febrero 13). *Frameworks en el desarrollo web: Las mejores prácticas para tu negocio online*. https://www.wearemarketing.com/es/blog/frameworks-en-el-desarrollo-web-las-mejores-practicas-para-tu-negocio-online.html

Tena, M. (2018, noviembre 20). Metodología «Agile». La revolución de las formas de trabajo. *¿Qué es la metodología «agile»?* https://www.bbva.com/es/metodologia-agile-la-revolucion-las-formas-trabajo/

United Nations. (2015). *Transforming our world: The 2030 Agenda for Sustainable Development .:. Sustainable Development Knowledge Platform*. https://sustainabledevelopment.un.org/post2015/transformingourworld

Vadenbo, C., Notten, P., Ugaya, C., Ramirez, A., Rivela, B., Colomb, V., Bajaj, S., Hussein, W., Vázquez-Rowe, I., Kahhat, R., Kumarasena, S., Arjeevani, U., Wernet, G., Turner, D., Walakira, P., & Tashobya, D. (2020). *Hoja de ruta para la creación de bases de datos nacionales de análisis del ciclo de vida Orientaciones y recomendaciones de diferentes partes del mundo*.

Vázquez-Rowe, I., Kahhat, R., & Sánchez, I. (2019). Perú LCA: Launching the Peruvian national life cycle database. *The International Journal of Life Cycle Assessment*, *24*(11), 2089-2090. https://doi.org/10.1007/s11367-019-01668-w

Vigon, B., Sonnemann, G., Asselin, A., Schrijvers, D., Ciroth, A., Chen, S. S., Braga, T., Poolsawad, N., Mungkalasiri, J., Boureima, F., & Milà i Canals, L. (2017). Review of LCA datasets in three emerging economies: A summary of learnings. *The International Journal of Life Cycle Assessment*, *22*(11), 1658-1665. https://doi.org/10.1007/s11367-016-1198-2

Wells, D. (2013). *Extreme Programming: A Gentle Introduction.* http://www.extremeprogramming.org/

Wolf, M.-A. (2014). *National LCA databases*. 66.

Wolf, M.-A., & Chomkhamsri, K. (2012). *Selecting the Environmental Indicator for Decoupling Indicators*. 5.

9. Anexos:

Anexo 1: Manual de Usuario.

UCUENCA

Desarrollo de un prototipo informático de soporte para gestionar conjuntos de datos de inventarios de ciclo de vida representativos del contexto ecuatoriano

Manual de Usuario

Versión: 2.0

Fecha: 25/02/2022

[2.0]

UCUENCA

Hoja de Control

Organismo	Universidad de Cuenca		
Proyecto	Desarrollo de un prototipo informático de soporte para gestionar conjuntos de datos de inventarios de ciclo de vida representativos del contexto ecuatoriano		
Entregable	Manual de Usuario		
Autores	Kamila Nicole Molina Orellana & Estuardo Mateo Quizhpi Peralta		
Versión/Edición	2.0	**Fecha Versión**	25/02/2022
		Nº Total de Páginas	22

Registro de cambios

Versión	Causa del Cambio	Responsable del Cambio	Fecha del Cambio
1.0	Versión inicial	Kamila Nicole Molina Orellana	15/09/2021
		Estuardo Mateo Quizhpi Peralta	
2.0	Versión con control de versiones de LCA	Kamila Nicole Molina Orellana	25/02/2022
		Estuardo Mateo Quizhpi Peralta	

Control de distribución

Nombre y Apellidos
Kamila Nicole Molina Orellana
Estuardo Mateo Quizhpi Peralta

Índice

1 Descripción del sistema

A continuación se dará detalle del propósito, alcance y funcionalidad del documento de Manual de usuario.

1.1 Propósito

Este documento tiene el propósito de informar y especificar al potencial usuario el funcionamiento del sistema. Así mismo, este manual sirve como un control de versiones, ya que en un futuro se pueden agregar nuevas funcionalidades. A su vez, este documento ayuda al mantenimiento del software.

1.2 Alcance

El siguiente manual es una guía para los usuarios que utilizarán al sistema, con el cuál conocerán los requerimientos del mismo, y el alcance que tiene su interacción.

1.3 Funcionalidad

Este sistema tiene como propósito la implementación de una base de datos relacional para almacenar metadatos de LCA. La base de datos está modelada sobre la estructura de un formato de archivos de LCA. Con la base de datos, se pueden formular preguntas y obtener información de calidad y sobre todo de valor a través de una interfaz de programación de aplicaciones. Este prototipo puede ser modificado.

2 Desarrollo del manual de usuario de la API

Por fines didácticos, se subdivide el capítulo en las funcionalidades que presenta el sistema, que son las consultas personalizadas que pueden hacer los usuarios, y a su vez el hecho de subir archivos al sistema.

2.1 Consultas estándar sobre los Archivos LCA

Estas son las consultas más sencillas que puede realizar el usuario, pues está buscando directamente en la base de datos toda la información relacionada con cierta tabla. Las consultas pueden ser realizadas a los módulos presentados en la *Fig.A.1.* En este caso, se filtra la información solo por el identificador único de cada tabla, por lo que el usuario puede ingresar ese valor de atributo y analizar su correspondiente respuesta como se puede ver en la *Fig.A.2*. Cabe destacar que solo el administrador tiene acceso a las vistas mencionadas.

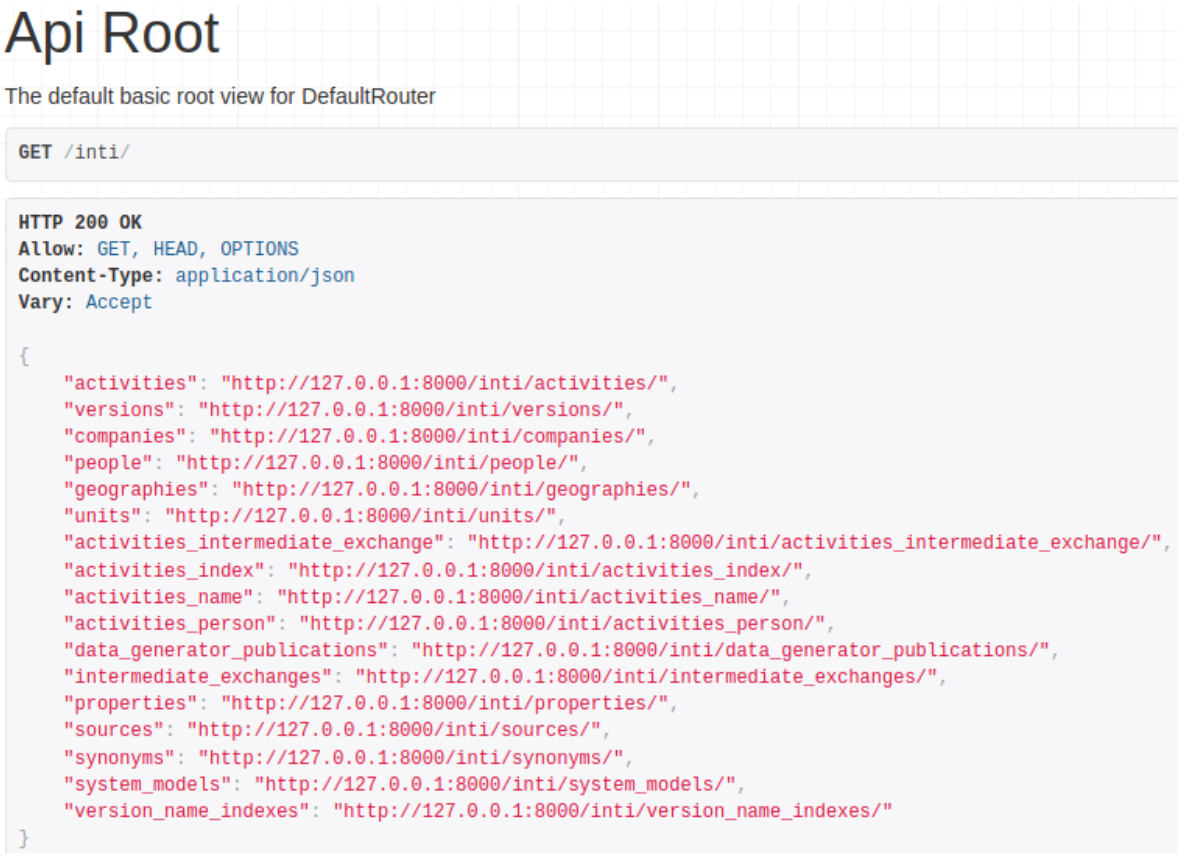


Fig. A.1: Captura de pantalla de los módulos disponibles para las consultas estándar de los archivos LCA.

```
GET /inti/activities_index/f7e93a25-56e4-4268-a603-3bfd57c79eff/
```

```
HTTP 200 OK
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "id": "f7e93a25-56e4-4268-a603-3bfd57c79eff",
    "geography": {
        "id": "34dbbff8-88ce-11de-ad60-0019e336be3a",
        "longitude": "",
        "latitude": "",
        "un_code": "0",
        "un_region_code": "0",
        "un_subregion_code": "0",
        "name": "Global",
        "short_name": "GLO"
    },
    "system_model": {
        "id": "8b738ea0-f89e-4627-8679-433616064e82",
        "name": "Undefined",
        "short_name": "Undefined"
    },
    "start_date": "1981-01-01",
    "end_date": "2005-12-31",
    "special_activity_type": "0"
```

Fig. A.2: Captura de pantalla de una sección de la salida de un ejemplo de consulta.

2.2 Consultas Personalizadas LCA

A continuación se dará detalle de las consultas personalizadas que se pueden realizar en el prototipo, para que el administrador y los usuarios conozcan cómo utilizarlas. Estas consultas son presentadas a nivel de código y también gráficamente con la UI a nivel de detalle. Cabe destacar que el administrador tiene acceso a ambas vistas que se van a mencionar a continuación.

2.2.1 Usuario en general.

Para poder acceder al sistema se debe iniciar sesión en la plataforma, como se puede ver en la *Fig. A.3,* ingresando nombre de usuario y contraseña. Posteriormente seleccionando el botón “Sign in”

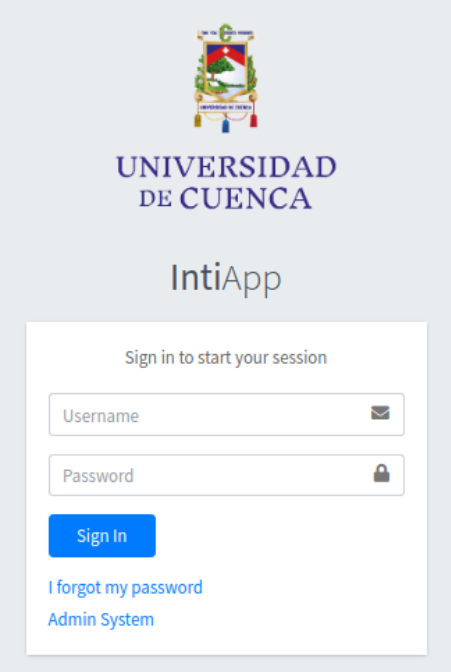
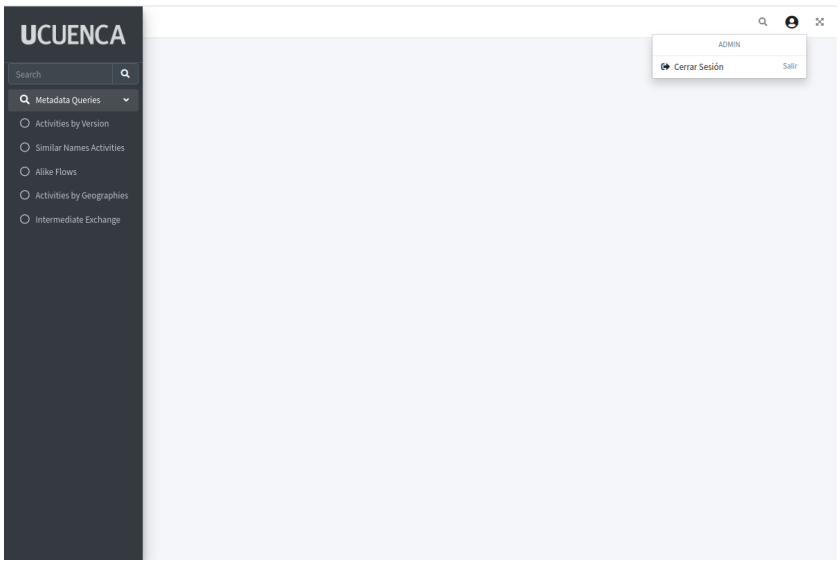


Fig. A.3: Captura de pantalla del login a la plataforma.

Una vez iniciada la sesión, el usuario tiene acceso al menú principal de la izquierda, como indica la *Fig. A.4.a*. Aquí se encuentran agrupadas las consultas personalizadas conforme lo que definen y desean responder. Al dar click sobre ellas o cualquiera de ellas se podrá acceder a la vista que corresponde a dicha consulta como se puede ver en la *Fig. A.4.b*. que accede a la consulta personalizada de Similar Names Activities.

a



b

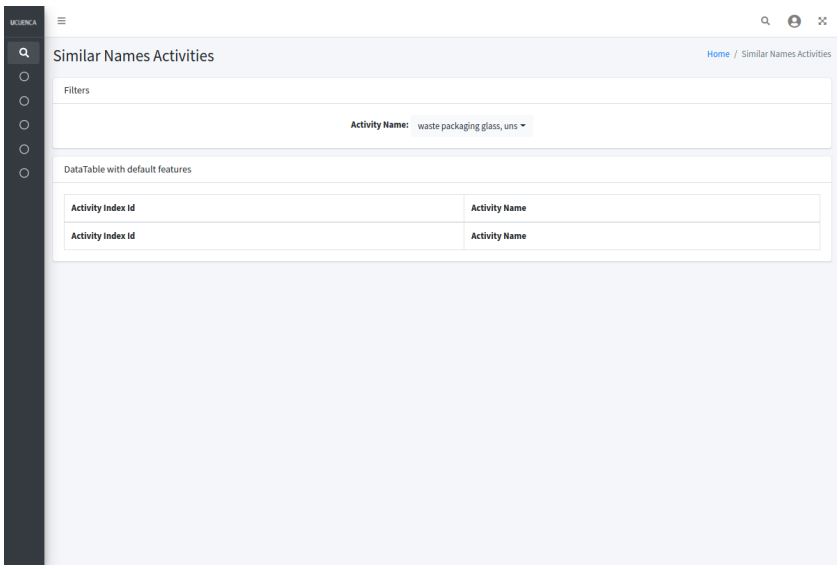


Fig. A.4: Captura de pantalla del menú principal de la plataforma.

Continuando con el proceso de consulta, se deben escoger valores para los filtros que indique cada vista como se señala en la *Fig. A.5,* se obtiene la respuesta correspondiente a la consulta realizada tras colocar el filtro deseado. Por ello la tabla dinámica mostrará los datos obtenidos a partir de dicha consulta y mostrará los resultados de la misma a manera abstraída, debido a que por la gran cantidad de datos el sistema puede demorarse más en procesar los datos.

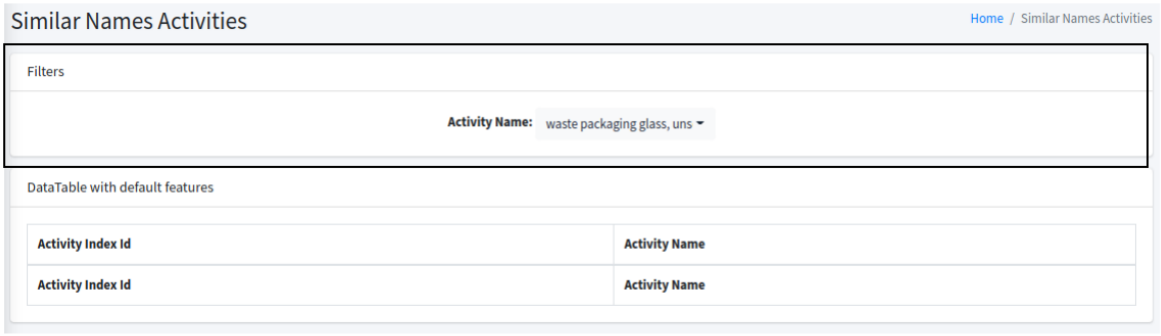


Fig. A.5: Captura de pantalla que indica los filtros de la consulta.

Una vez ingresados los valores, se obtiene la misma vista, pero con la tabla actualizada como se puede observar en la *Fig.A.6*. que ya está mostrando información según la consulta realizada.

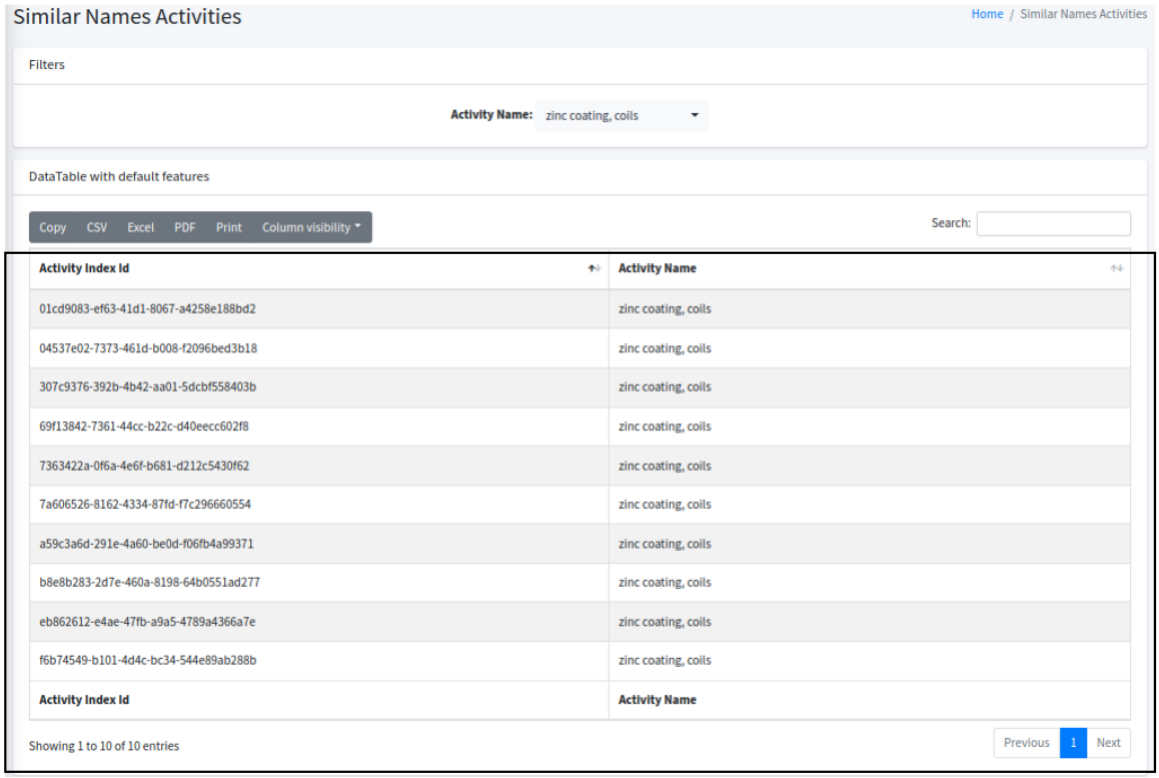


Fig. A.6: Captura de pantalla que indica la tabla llenada con los datos de la consulta realizada.

Finalmente, se efectúa el mismo proceso cuando se quiere realizar alguna otra consulta, dirigiéndose al menú y seleccionando el tipo de consulta que desea probar, y de igual manera ingresando los filtros y obteniendo su respectiva tabla de información.

2.2.1.1 Descarga de información

Una vez realizada la consulta, y generada la respuesta en la tabla, el usuario tiene la opción de descargar lo obtenido, como se puede ver en la Fig.A.7.a, se tienen varias opciones de descarga y queda a decisión del usuario cual desee. Una vez de click en una de ellas, la descarga inicia automáticamente y se descargan los datos que aparecen en la tabla. Y como se puede ver en la Fig.A.7.b, se tiene también una opción de descarga de datos, que descargará todos los datos obtenidos para dicha consulta.

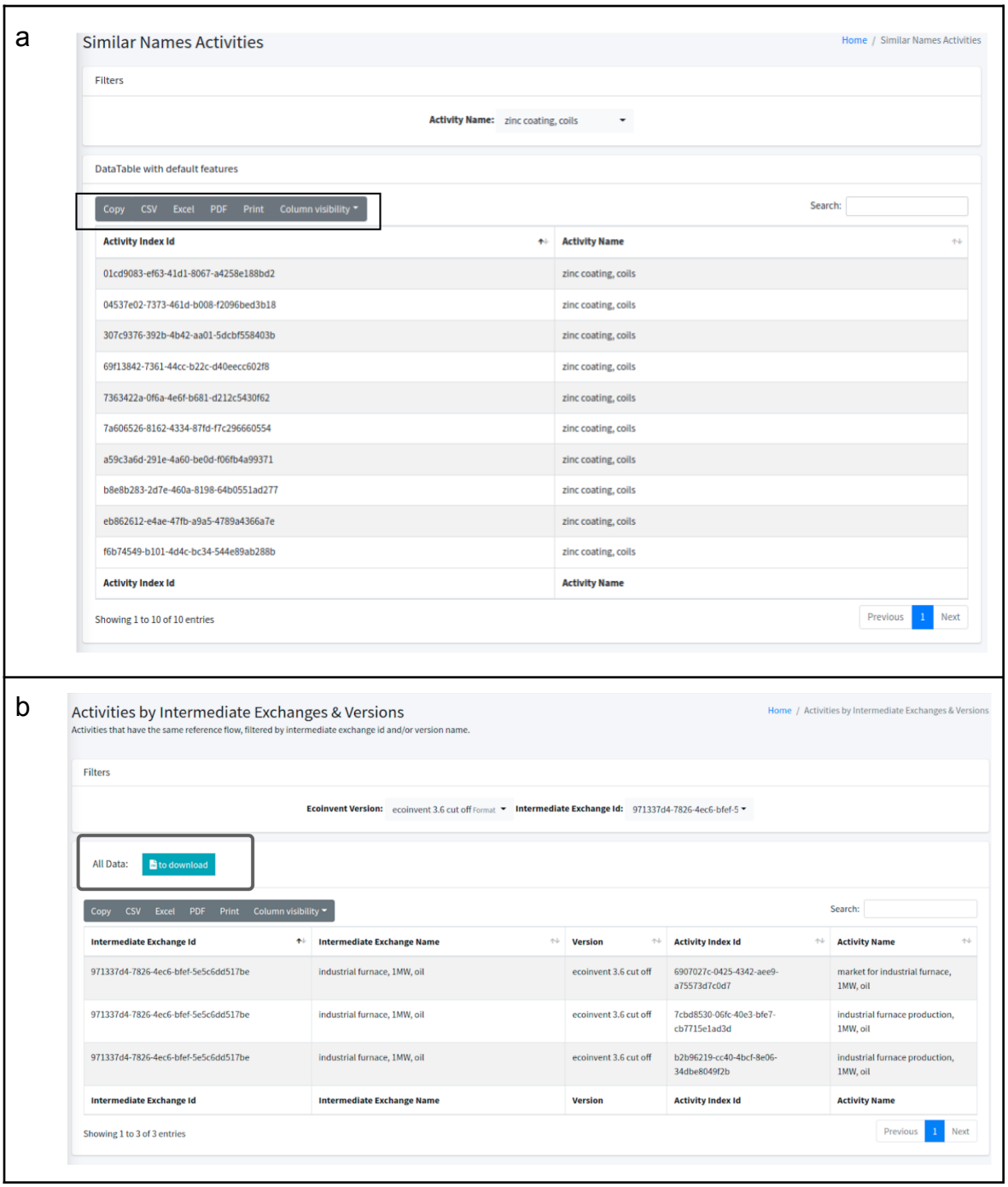


Fig. A.7: Captura de pantalla que indica las opciones de descarga de la consulta realizada.

2.2.2 Administrador

El administrador tiene el privilegio de acceder a la UI de la aplicación ingresando usuario y contraseña, para utilizar la interfaz como se indicó el punto anterior. También puede acceder al API root y realizar las consultas por medio de las URLs. A continuación se detalla el manual de usuario para el administrador en

el caso de las consultas personalizadas desde el API root.

Todas las respuestas a las consultas hechas están construidas internamente en formato JSON y paginadas, si su primer atributo llamado: *response*, es igual a *0* quiere decir que se llevó a cabo la consulta, caso contrario, si es igual a *1*, quiere decir que no se llevó a cabo la consulta.

A continuación se detallan las consultas personalizadas construidas en base a los archivos LCA, pues como su nombre lo indica, se desea obtener cierta información en base a filtros.

2.2.3 Actividades con el mismo nombre, filtrado por la versión.

Se accede a esta página con el URL ejemplo:

/inti/activities/same_names/?version=ecoinvent 3.6 cut off

- **Proceso**

Si el método se realiza correctamente se obtiene la vista de la *Fig.A.8*. La cual es la respuesta a la consulta y se obtiene en formato JSON y paginada.

```
{
    "response": "0",
    "response1": [
        {
            "activity id": "0021a538-9534-4fd4-835e-a2fa8f76bb9d",
            "activity name": "market for stone wool, packed  ",
            "version": "ecoinvent 3.6 cut off "
        },
        {
            "activity id": "002477cd-70fa-49c1-afc3-c372abbcf72b",
            "activity name": "market for gas motor, mini CHP plant ",
            "version": "ecoinvent 3.6 cut off "
        },
        {
            "activity id": "002510a3-dcbb-4fb9-85f3-54e72b794bff",
            "activity name": "market for refractory material, acid ",
            "version": "ecoinvent 3.6 cut off "
        },
        {
            "activity id": "002fa54c-1829-4231-82ed-47e35a6745d4",
            "activity name": "market for mischmetal ",
            "version": "ecoinvent 3.6 cut off "
        },
        {
            "activity id": "003682ba-f70c-474f-ba5f-8203d2d00512",
            "activity name": "heat and power co-generation, wood chips, 6667 kW",
            "version": "ecoinvent 3.6 cut off "
        },
        {
            "activity id": "003a2fba-a774-4ff8-b8da-3adb2e14aeba",
            "activity name": "market for operation, housing system, pig, fully-slatted floor, per pig place  ",
            "version": "ecoinvent 3.6 cut off "
        },
        {
```


Fig. A.8: Captura de pantalla de una sección de la salida de la consulta.

2.2.4 Actividades con nombres similares, filtrado por la versión y el nombre o palabra clave de la actividad.

Se accede a esta página con el URL ejemplo:

/inti/activities/similar_names/?name=aluminium

- **Proceso**

Si el método se realiza correctamente se obtiene la vista de la *Fig.A.9.* La cual es la respuesta a la consulta y se obtiene en formato JSON y paginada.


```
{
    "response": "0",
    "response1": [
        {
            "intermediate exchange id": "e9f3d81d-43dc-484c-9b16-2934618d8724",
            "intermediate exchange name": "heat and power co-generation unit, organic Rankine cycle, 3MW electrical ",
            "version": "ecoinvent 3.6 cut off ",
            "activity id": "61de8608-d184-4e0f-a26a-3cad0db490fb",
            "activity name": "market for heat and power co-generation unit, organic Rankine cycle, 3MW electrical  "
        }
    ]
}
```


Fig. A.9: Captura de pantalla de la salida de la consulta.

2.2.5 Actividades con la misma geografía, filtrado por el nombre de la geografía,

Se accede a esta página con el URL ejemplo:

inti/geographies/{pk}/activities/

{pk}=34dbbff8-88ce-11de-ad60-0019e336be3a

- **Proceso**

Si el método se realiza correctamente se obtiene la vista de la *Fig.A.10.* La cual es la respuesta a la consulta y se obtiene en formato JSON y paginada.


```
{
    "response": "0",
    "response1": [
        {
            "activity id": "39e0fc4b-8bc3-4363-b224-d961dc5141b7",
            "activity name": "anaerobic digestion plant construction, for biowaste",
            "geography name": "Global "
        },
        {
            "activity id": "c2d58788-238b-464b-89c5-6b075d323033",
            "activity name": "2-butanol production by hydration of butene",
            "geography name": "Global "
        },
        {
            "activity id": "e40735fd-bd80-4a0d-9e84-3e1d1724c27c",
            "activity name": "acetaldehyde oxidation",
            "geography name": "Global "
        },
        {
            "activity id": "a049209a-ac75-4213-bbab-0f65dbd878e8",
            "activity name": "beet sugar production ",
            "geography name": "Global "
        },
        {
            "activity id": "b04261d4-55a1-4b44-a327-b51c1ed5387f",
            "activity name": "coking ",
            "geography name": "Global "
        },
        {
            "activity id": "6dbed587-a801-4791-8c4d-321d807ff31f",
            "activity name": "maleic anhydride production by direct oxidation of n-butane  ",
            "geography name": "Global "
        },
        {
```


Fig. A.10: Captura de pantalla de una sección de la salida de la consulta.

2.2.6 Flujos de referencia asociados con una actividad, filtrado por la versión y el nombre de la actividad.

Se accede a esta página con el URL ejemplo:

/inti/activities/{pk}/flows/

{pk}=184

/inti/activities/{pk}/flows/?version=ecoinvent 3.6 cut off

{pk}=184

- **Proceso**

Si el método se realiza correctamente se obtiene la vista de la *Fig.A.11*. La cual es la respuesta a la consulta y se obtiene en formato JSON y paginada.


```
{
    "response": "0",
    "response1": [
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "version": "ecoinvent 3.6 cut off ",
            "activity id": "2e1eab53-4be9-4d10-8ad2-958a5001e780",
            "activity name": "trawler maintenance, steel  "
        },
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "version": "ecoinvent 3.6 cut off ",
            "activity id": "2e1eab53-4be9-4d10-8ad2-958a5001e780",
            "activity name": "trawler maintenance, steel  "
        },
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "version": "ecoinvent 3.6 cut off ",
            "activity id": "2e1eab53-4be9-4d10-8ad2-958a5001e780",
            "activity name": "trawler maintenance, steel  "
        },
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "version": "ecoinvent 3.6 cut off ",
            "activity id": "2e1eab53-4be9-4d10-8ad2-958a5001e780",
            "activity name": "trawler maintenance, steel  "
        },
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "version": "ecoinvent 3.6 cut off ",
            "activity id": "2e1eab53-4be9-4d10-8ad2-958a5001e780",
            "activity name": "trawler maintenance, steel  "
        },
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "version": "ecoinvent 3.6 cut off ",
            "activity id": "63b2e106-de11-4ace-8bbd-77b03c0d0ba5",
            "activity name": "market for trawler maintenance, steel"
        }
    ]
}
```


Fig. A.11: Captura de pantalla de la salida de la consulta.

2.2.7 Actividades que tienen el mismo flujo de referencia, filtrado por la versión y el nombre del intermediate exchange.

Se accede a esta página con el URL ejemplo:

/inti/intermediate_exchanges/{pk}/activities/

{pk}=42761d87-05d9-4877-b21e-001ecf0c747d

/inti/intermediate_exchanges/{pk}/activities/?version=ecoinvent 3.6 cut off

{pk}=42761d87-05d9-4877-b21e-001ecf0c747d

- **Proceso**

Si el método se realiza correctamente se obtiene la vista de la *Fig.A.12.* La cual es la respuesta a la consulta y se obtiene en formato JSON y paginada.


```
{
    "response": "0",
    "response1": [
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "version": "ecoinvent 3.6 cut off ",
            "activity id": "2e1eab53-4be9-4d10-8ad2-958a5001e780",
            "activity name": "trawler maintenance, steel  "
        },
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "version": "ecoinvent 3.6 cut off ",
            "activity id": "2e1eab53-4be9-4d10-8ad2-958a5001e780",
            "activity name": "trawler maintenance, steel  "
        },
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "version": "ecoinvent 3.6 cut off ",
            "activity id": "2e1eab53-4be9-4d10-8ad2-958a5001e780",
            "activity name": "trawler maintenance, steel  "
        },
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "version": "ecoinvent 3.6 cut off ",
            "activity id": "2e1eab53-4be9-4d10-8ad2-958a5001e780",
            "activity name": "trawler maintenance, steel  "
        },
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "version": "ecoinvent 3.6 cut off ",
```


Fig. A.12: Captura de pantalla de una sección de la salida de la consulta.

2.2.8 Flujos de referencia por nombre o palabra clave, filtrado por la versión y el nombre del intermediate exchange.

Se accede a esta página con el URL ejemplo:

inti/activities/alike_flows/?version=ecoinvent 3.6 cut off&activity_name=trawler

- **Proceso**

Si el método se realiza correctamente se obtiene la vista de la *Fig.A.13.* La cual es la respuesta a la consulta y se obtiene en formato JSON y paginada.

```
{
    "response": "0",
    "response1": [
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "version": "ecoinvent 3.6 cut off "
        },
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "version": "ecoinvent 3.6 cut off "
        },
        {
            "intermediate exchange id": "199596a7-3977-4402-8c43-7e3aafa1b205",
            "intermediate exchange name": "used trawler, steel",
            "version": "ecoinvent 3.6 cut off "
        },
        {
            "intermediate exchange id": "a554cf7d-1ca9-479f-af21-14d11ca0f679",
            "intermediate exchange name": "trawler, steel  ",
            "version": "ecoinvent 3.6 cut off "
        },
        {
            "intermediate exchange id": "a554cf7d-1ca9-479f-af21-14d11ca0f679",
            "intermediate exchange name": "trawler, steel  ",
            "version": "ecoinvent 3.6 cut off "
        },
        {
            "intermediate exchange id": "a554cf7d-1ca9-479f-af21-14d11ca0f679",
            "intermediate exchange name": "trawler, steel  ",
            "version": "ecoinvent 3.6 cut off "
        },
        {
            "intermediate exchange id": "a554cf7d-1ca9-479f-af21-14d11ca0f679",
            "intermediate exchange name": "trawler, steel  ",
            "version": "ecoinvent 3.6 cut off "
        },
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "version": "ecoinvent 3.6 cut off "
        },
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "version": "ecoinvent 3.6 cut off "
        },
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "version": "ecoinvent 3.6 cut off "
```


Fig. A.13: Captura de pantalla de una sección de la salida de la consulta.

2.2.9 Referencias asociadas a una actividad, filtrado por el activity intermediate exchange.

Se accede a esta página con el URL ejemplo:

inti/activities/{pk}/references/

{pk}=184

- **Proceso**

Si el método se realiza correctamente se obtiene la vista de la *Fig.A.14.* La cual es la respuesta a la consulta y se obtiene en formato JSON y paginada.

```
{
    "response": "0",
    "response1": [
        {
            "activity intermediate exchange id": "1107",
            "intermediate exchange id": "d51e5a32-c20a-4da6-8d58-31f344b50c00",
            "intermediate exchange name": "waste graphical paper "
        }
    ]
}
```


Fig. A.14: Captura de pantalla de la salida de la consulta.

2.2.10 Productores asociados a un intercambio intermedio, filtrado por el intermediate exchange.

Se accede a esta página con el URL ejemplo:

/inti/intermediate_exchanges/{pk}/producers/

{pk}=42761d87-05d9-4877-b21e-001ecf0c747d

- **Proceso**

Si el método se realiza correctamente se obtiene la vista de la *Fig.A.15*. La cual es la respuesta a la consulta y se obtiene en formato JSON y paginada.


```
{
    "response": "0",
    "response1": [
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "activity name name": "trawler maintenance, steel  ",
            "data generator and publication person name": "Angel Avadi  "
        },
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "activity name name": "trawler maintenance, steel  ",
            "data generator and publication person name": "[System]  "
        },
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "activity name name": "market for trawler maintenance, steel",
            "data generator and publication person name": "[System]  "
        },
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "activity name name": "patagonian grenadier, capture by trawler and landing in fish blocks, frozen ",
            "data generator and publication person name": "Angel Avadi  "
        },
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "activity name name": "patagonian grenadier, capture by trawler and landing in fish blocks, frozen ",
            "data generator and publication person name": "Ian Vázquez-Rowe"
        },
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "activity name name": "hake, capture by trawler and landing whole, fresh",
            "data generator and publication person name": "Angel Avadi  "
        },
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "activity name name": "hake, capture by trawler and landing whole, fresh",
            "data generator and publication person name": "Angel Avadi  "
        },
        {
            "intermediate exchange id": "42761d87-05d9-4877-b21e-001ecf0c747d",
            "intermediate exchange name": "trawler maintenance, steel  ",
            "activity name name": "hake, capture by trawler and landing whole, fresh",
            "data generator and publication person name": "Ian Vázquez-Rowe"
        }
    ]
}
```


Fig. A.15: Captura de pantalla de la salida de la consulta.

2.2.11 Personas asociadas con una actividad, filtrado por el activity intermediate exchange.

Se accede a esta página con el URL ejemplo:

inti/activities/{pk}/people/

{pk}=184

- **Proceso**

Si el método se realiza correctamente se obtiene la vista de la *Fig.A.16*. La cual es la respuesta a la consulta y se obtiene en formato JSON y paginada.


```
{
    "response": "0",
    "response1": [
        {
            "activity name name": "treatment of waste graphical paper, municipal incineration",
            "person name": "Emilia Moreno Ruiz "
        },
        {
            "activity name name": "treatment of waste graphical paper, municipal incineration",
            "person name": "Gregor Wernet"
        },
        {
            "activity name name": "treatment of waste graphical paper, municipal incineration",
            "person name": "Niels Jungbluth "
        },
        {
            "activity name name": "treatment of waste graphical paper, municipal incineration",
            "person name": "[System]  "
        },
        {
            "activity name name": "treatment of waste graphical paper, municipal incineration",
            "person name": "Tereza Levova"
        }
    ]
}
```


Fig. A.16: Captura de pantalla de la salida de la consulta.

2.2.12 Actividades asociadas con una persona, filtrado por la persona.

Se accede a esta página con el URL ejemplo:

inti/people/{pk}/activities/

{pk}=253759c9-c631-44cc-b293-955a2f6d4cba

- **Proceso**

Si el método se realiza correctamente se obtiene la vista de la *Fig.A.17*. La cual es la respuesta a la consulta y se obtiene en formato JSON y paginada.

```
{
    "response": "0",
    "response1": [
        {
            "person name": "Caroline  Gaudreault  ",
            "activity name name": "containerboard production, linerboard, kraftliner"
        },
        {
            "person name": "Caroline  Gaudreault  ",
            "activity name name": "folding boxboard carton production"
        },
        {
            "person name": "Caroline  Gaudreault  ",
            "activity name name": "containerboard production, fluting medium, recycled "
        },
        {
            "person name": "Caroline  Gaudreault  ",
            "activity name name": "containerboard production, linerboard, testliner "
        },
        {
            "person name": "Caroline  Gaudreault  ",
            "activity name name": "kraft paper production"
        },
        {
            "person name": "Caroline  Gaudreault  ",
            "activity name name": "white lined chipboard carton production "
        },
        {
            "person name": "Caroline  Gaudreault  ",
            "activity name name": "paper production, newsprint, recycled"
        },
        {
            "person name": "Caroline  Gaudreault  ",
            "activity name name": "solid bleached and unbleached board carton production  "
        },
        {
            "person name": "Caroline  Gaudreault  ",
            "activity name name": "paper sack production "
        },
        {
            "person name": "Caroline  Gaudreault  ",
            "activity name name": "containerboard production, fluting medium, semichemical"
        },
        {
            "person name": "Caroline  Gaudreault  ",
            "activity name name": "corrugated board box production"
        }
    ]
}
```


Fig. A.17: Captura de pantalla de la salida de la consulta.

2.3 Intercambio de versiones

El intercambio de versiones está considerado como un método de consultas, debido a que se filtra la información para conocer las versiones futuras de una ingresada como parámetro.

Se accede a esta página con el URL ejemplo:

inti/correspondences/version/?ai-id={ai-id}&ie-name={ie-name}

{pk}=08550bca-a4c1-4373-890b-8ef2ceaae42c

{ie-name}=bagasse, from sugarcane

- **Proceso**

Si el método se realiza correctamente se obtiene la vista de la *Fig.A.15*. La cual es la respuesta a la consulta y se obtiene en formato JSON y paginada.

```
{
    "response": [
        {
            "aie_3_1": "1132157",
            "aie_3_1 id": "08550bca-a4c1-4373-890b-8ef2ceaae42c",
            "aie_3_2": "1093832",
            "aie_3_2 id": "db326332-e3ee-4ef8-ae1c-f462d1ee06a5"
        },
        {
            "aie_3_3": "1053440",
            "aie_3_3 id": "613d0bc4-e7f8-47de-a14f-006b88aab577"
        },
        {
            "aie_3_4": "1020025",
            "aie_3_4 id": "3efe8d62-4319-4ff6-a783-3d58dffddb06"
        },
        {
            "aie_3_5": "959788",
            "aie_3_5 id": "c18292f2-76bc-499d-994c-38e16af7d620"
        },
        {
            "aie_3_6": "51485",
            "aie_3_6 id": "54b95cba-c256-4c92-a104-0a752a30cda7"
        },
        {
            "aie_3_7_1": "423679",
            "aie_3_7_1 id": "bbb1597c-59bc-4409-b7d1-4a6a22d15d82"
        }
    ]
}
```


Fig. A.15: Captura de pantalla de la salida de la consulta.

3 Desarrollo del manual de usuario de la interfaz para la gestión de archivos LCA.

3.1 Subir Archivo

El usuario se encontrará con la interfaz relacionada a subir un archivo LCA; esta interfaz como se ve en la *Fig.A.16.* Es un formulario que cuenta con dos atributos que deben ser introducidos, caso contrario no se llevará a cabo la acción.

Para el primer atributo a introducir, se debe tomar en cuenta que el formato del archivo debe ser en *.zip*, caso contrario no se procesa su información. Ejemplificando lo que se espera que el usuario ingrese, el nombre del archivo a subir debe tener la siguiente estructura: $ecoinvent\ 3.6\_apos\_ecoSpold02$ y el archivo en formato zip debe tener la siguiente estructura: $ecoinvent\ 3.6\_apos\_ecoSpold02.zip$

Así mismo, se debe escoger entre las opciones desplegables el nombre de la versión del archivo que se está subiendo. Por lo tanto, al escoger ambos parámetros ya se puede dar click en el botón Upload. Después de esto, se debe esperar unos minutos hasta que se complete la acción, tomando en cuenta que

son varios archivos que deben ser procesados.

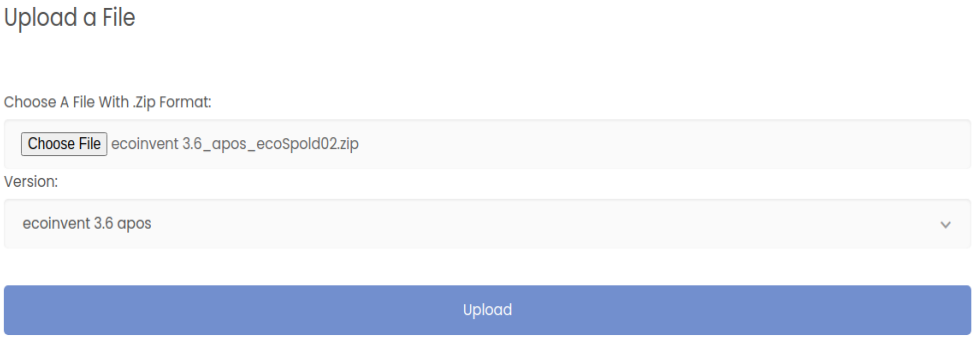


Fig. A.16: Interfaz para subir un archivo LCA.

4 FAQ

1. **¿Se puede ingresar otro formato de archivos para poblar la base?**

No, se espera que el usuario suba archivos del formato ecoSpold 2 solamente.

2. **¿Qué pasa si el archivo que ingresó ya existe?**

El sistema igual acepta la petición de subir el archivo pero con las validaciones internas no duplica los datos en la base de datos.

3. **¿Puedo hacer otro tipo de consultas personalizadas?**

Por el momento no, pero sí se pueden agregar más consultas dependiendo de las futuras necesidades. Se espera una siguiente actualización del sistema que contemple este alcance.

Anexo 2: Manual Técnico.

UCUENCA

Desarrollo de un prototipo informático de soporte para gestionar conjuntos de datos de inventarios de ciclo de vida representativos del contexto ecuatoriano

Manual Técnico

Versión: 2.0

Fecha: 25/02/2022

[2.0]

UCUENCA

Hoja de Control

Organismo	Universidad de Cuenca		
Proyecto	Desarrollo de un prototipo informático de soporte para gestionar conjuntos de datos de inventarios de ciclo de vida representativos del contexto ecuatoriano		
Entregable	Manual Técnico		
Autores	Kamila Nicole Molina Orellana & Estuardo Mateo Quizhpi Peralta		
Versión/Edición	2.0	**Fecha Versión**	25/02/2021
		Nº Total de Páginas	39

Registro de cambios

Versión	Causa del Cambio	Responsable del Cambio	Fecha del Cambio
1.0	Versión inicial	Kamila Nicole Molina Orellana	15/09/2021
		Estuardo Mateo Quizhpi Peralta	
2.0	Versión con control de versiones de LCA	Kamila Nicole Molina Orellana	25/02/2022
		Estuardo Mateo Quizhpi Peralta	

Control de distribución

Nombre y Apellidos
Kamila Nicole Molina Orellana
Estuardo Mateo Quizhpi Peralta

Índice

1 Descripción del sistema

1.1 Propósito

Este documento tiene el propósito de informar y especificar al potencial usuario la estructura y conformación del sistema con el fin que puedan dar soporte y hacer las respectivas modificaciones o actualizaciones al prototipo en general.

1.2 Alcance

El siguiente manual es una guía para los usuarios que darán soporte al sistema, con el cuál conocerán los requerimientos y la estructura para la construcción del prototipo. Aquí se muestran las herramientas necesarias para la construcción y la funcionalidad del sistema.

1.3 Funcionalidad

Este documento tiene como propósito informar y especificar a los usuarios sobre la estructura y conformación del sistema con el fin que puedan hacer soporte, modificaciones o actualizaciones al sistema en general. Este sistema tiene como propósito la implementación de una base de datos relacional para almacenar metadatos de LCA. La base de datos está modelada sobre la estructura de un formato de archivos LCA. Con la base de datos, se pueden formular preguntas y obtener información de calidad y sobre todo de valor a través de una interfaz de programación de aplicaciones.

2 Requisitos del sistema

- **Requerimientos de hardware**

Equipo, teclado, mouse, monitor.

- **Requerimientos de software**

Cualquier sistema operativo.
Django.
Python.
Conexión a internet.
Cualquier IDE de desarrollo.

3 Herramientas utilizadas para el desarrollo

- Python 3.8.10
- Visual Studio Code 1.64
- Django 3.2.1
- Django Rest Framework 3.12.4
- Servidor de base de datos (PostgreSQL) 13.6
- Bootstrap v.4

4 Descripción del manual técnico de la Base de Datos.

La creación de la base de datos, se dió por medio de SGBD PostgreSQL. se creó una nueva estructura de base de datos con el nombre *db_metadata* y con la configuración que se puede ver a continuación en la Fig. B. 1.

```
CREATE DATABASE db_metadata
    WITH
    OWNER = postgres
    ENCODING = 'UTF8'
    LC_COLLATE = 'en_US.UTF-8'
    LC_CTYPE = 'en_US.UTF-8'
    TABLESPACE = pg_default
    CONNECTION LIMIT = -1;
```

Fig. B. 1: Script para la creación de la Base de Datos.

Una vez creada la estructura de la base de datos, se procedió a crear los esquemas correspondientes a las diferentes tablas definidas en el modelo ER para el prototipo. A continuación, en la Fig. B. 2. se muestra el script de creación de las diferentes tablas dentro de la base de datos previamente creada.

```
CREATE TABLE public.activity_intermediate_exchange (
    activity_id bigint NOT NULL,
    intermediate_exchange_id character(36) NOT NULL,
    variable_name character varying,
    id bigint NOT NULL,
    input_group character varying,
    output_group character varying
);

CREATE TABLE public.activity (
    activity_index_id character(36) NOT NULL,
    allocation_comment character varying,
    general_comment character varying,
    comment_technology character varying,
    included_processes character varying,
    id bigint NOT NULL,
    is_data_valid_for_entire_period boolean,
    data_generator_and_publication_id bigint,
    version_id bigint NOT NULL
);

CREATE TABLE public.activity_index (
    id character(36) NOT NULL,
    geography_id character(36),
    system_model_id character(36),
    start_date character varying,
    end_date character varying,
    special_activity_type character varying
);

CREATE TABLE public.activity_name (
    id character(36) NOT NULL,
    activity_name character varying
);

CREATE TABLE public.activity_person (
    person_id character(36) NOT NULL,
    activity_id bigint NOT NULL
```

```
);

CREATE TABLE public.company (
    id character(36) NOT NULL,
    code character varying,
    name character varying,
    website character varying,
    comment character varying
);

CREATE TABLE public.correspondence (
    id bigint NOT NULL,
    activity_intermediate_exchange bigint NOT NULL,
    next_activity_intermediate_exchange bigint NOT NULL,
    differences character varying
);

CREATE TABLE public.data_generator_and_publication (
    person_id character(36),
    source_id character(36),
    is_copyright_protected boolean,
    id bigint NOT NULL
);

CREATE TABLE public.geography (
    id character(36) NOT NULL,
    longitude character varying,
    latitude character varying,
    un_code character varying,
    un_region_code character varying,
    un_subregion_code character varying,
    name character varying,
    short_name character varying
);

CREATE TABLE public.intermediate_exchange (
    id character(36) NOT NULL,
    unit_id character(36),
    name character varying
);

CREATE TABLE public.person (
    id character(36) NOT NULL,
    company_id character(36),
    name character varying,
    email character varying,
    address character varying,
    telephone character varying,
    telefax character varying
);

CREATE TABLE public.property (
    id character(36) NOT NULL,
    unit_id character(36),
    default_variable_name character varying,
    name character varying
);

CREATE TABLE public.source (
    id character(36) NOT NULL,
    type character varying,
    volume_no character varying,
    first_author character varying,
    additional_authors character varying,
    title character varying,
    names_of_editors character varying,
    page_numbers character varying,
    journal character varying,
    title_of_anthology character varying,
    place_of_publications character varying,
```

```
    publisher character varying,
    comment character varying,
    year character varying,
    short_name character varying
);

CREATE TABLE public.synonym (
    activity_id character(36),
    synonym character varying,
    id bigint NOT NULL
);

CREATE TABLE public.system_model (
    id character(36) NOT NULL,
    name character varying,
    short_name character varying
);

CREATE TABLE public.unit (
    id character(36) NOT NULL,
    name character varying,
    comment character varying
);

CREATE TABLE public.version (
    id bigint NOT NULL,
    version character varying
);

CREATE TABLE public.version_name_index (
    id bigint NOT NULL,
    activity_index_id character(36),
    activity_name_id character(36),
    version_id bigint NOT NULL
);

ALTER TABLE ONLY public.activity_intermediate_exchange ADD CONSTRAINT
acitivity_intermediate_exchange_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.activity_index ADD CONSTRAINT activity_index_pkey PRIMARY KEY
(id);

ALTER TABLE ONLY public.activity_name ADD CONSTRAINT activity_name_pkey PRIMARY KEY
(id);

ALTER TABLE ONLY public.activity_person ADD CONSTRAINT activity_person_pkey PRIMARY
KEY (person_id, activity_id);

ALTER TABLE ONLY public.activity ADD CONSTRAINT activity_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.company ADD CONSTRAINT company_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.correspondence ADD CONSTRAINT correspondence_pk PRIMARY KEY
(id);

ALTER TABLE ONLY public.data_generator_and_publication ADD CONSTRAINT
data_generator_and_publication_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.geography ADD CONSTRAINT geography_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.intermediate_exchange ADD CONSTRAINT
intermediate_exchange_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.person ADD CONSTRAINT person_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.source ADD CONSTRAINT source_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.synonym ADD CONSTRAINT synonym_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.system_model ADD CONSTRAINT system_model_pkey PRIMARY KEY
```

```
(id);

ALTER TABLE ONLY public.unit ADD CONSTRAINT unit_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.version_name_index ADD CONSTRAINT version_name_index_pkey
PRIMARY KEY (id);

ALTER TABLE ONLY public.version ADD CONSTRAINT version_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.activity ADD CONSTRAINT "FK_activity__version" FOREIGN KEY
(version_id) REFERENCES public.version(id) NOT VALID;

ALTER TABLE ONLY public.activity ADD CONSTRAINT "FK_activity_activity_index" FOREIGN
KEY (activity_index_id) REFERENCES public.activity_index(id) NOT VALID;

ALTER TABLE ONLY public.activity ADD CONSTRAINT
"FK_activity_data_generator_and_publication_id" FOREIGN KEY
(data_generator_and_publication_id) REFERENCES
public.data_generator_and_publication(id) NOT VALID;

ALTER TABLE ONLY public.activity_intermediate_exchange ADD CONSTRAINT "FK_activity_id"
FOREIGN KEY (activity_id) REFERENCES public.activity(id);

ALTER TABLE ONLY public.activity_index ADD CONSTRAINT "FK_activity_index_geography"
FOREIGN KEY (geography_id) REFERENCES public.geography(id);

ALTER TABLE ONLY public.activity_index ADD CONSTRAINT "FK_activity_index_system_model"
FOREIGN KEY (system_model_id) REFERENCES public.system_model(id);

ALTER TABLE ONLY public.correspondence ADD CONSTRAINT
"FK_activity_intermediate_exchange" FOREIGN KEY (activity_intermediate_exchange)
REFERENCES public.activity_intermediate_exchange(id);

ALTER TABLE ONLY public.activity_person ADD CONSTRAINT "FK_activity_person_activity"
FOREIGN KEY (activity_id) REFERENCES public.activity(id);

ALTER TABLE ONLY public.activity_person ADD CONSTRAINT "FK_activity_person_person"
FOREIGN KEY (person_id) REFERENCES public.person(id);

ALTER TABLE ONLY public.data_generator_and_publication ADD CONSTRAINT
"FK_data_generatot_and_publication_person" FOREIGN KEY (person_id) REFERENCES
public.person(id);

ALTER TABLE ONLY public.data_generator_and_publication ADD CONSTRAINT
"FK_data_generatot_and_publication_source" FOREIGN KEY (source_id) REFERENCES
public.source(id);

ALTER TABLE ONLY public.activity_intermediate_exchange ADD CONSTRAINT
"FK_intermediate_exchange_id" FOREIGN KEY (intermediate_exchange_id) REFERENCES
public.intermediate_exchange(id);

ALTER TABLE ONLY public.intermediate_exchange ADD CONSTRAINT
"FK_intermediate_exchange_unit" FOREIGN KEY (unit_id) REFERENCES public.unit(id);

ALTER TABLE ONLY public.correspondence ADD CONSTRAINT
"FK_next_activity_intermediate_exchange" FOREIGN KEY
(next_activity_intermediate_exchange) REFERENCES
public.activity_intermediate_exchange(id);

ALTER TABLE ONLY public.person ADD CONSTRAINT "FK_person_company" FOREIGN KEY
(company_id) REFERENCES public.company(id);

ALTER TABLE ONLY public.synonym ADD CONSTRAINT "FK_synonym_activity_index" FOREIGN KEY
(activity_id) REFERENCES public.activity_index(id);

ALTER TABLE ONLY public.version_name_index ADD CONSTRAINT
"FK_version_name_index__activity_index" FOREIGN KEY (activity_index_id) REFERENCES
public.activity_index(id);

ALTER TABLE ONLY public.version_name_index ADD CONSTRAINT
```

```
"FK_version_name_index__activiy_name" FOREIGN KEY (activity_name_id) REFERENCES
public.activity_name(id);

ALTER TABLE ONLY public.version_name_index ADD CONSTRAINT
"FK_version_name_index__version" FOREIGN KEY (version_id) REFERENCES
public.version(id) NOT VALID;
```

Fig. B. 2: Script para la creación de las tablas dentro de la Base de datos.

Como se pudo observar, cada tabla tiene sus respectivos atributos. Cada uno de ellos tiene un nombre y está detallado el tipo de dato de cada atributo de las respectivas tablas.

El script a su vez indica las claves primarias de las tablas, y de ser necesario según el detalle del modelo ER de la base de datos, también están definidas las claves foráneas de ellas.

5 Descripción del manual técnico de la interfaz para la publicación de archivos LCA.

El proyecto puede ser encontrado en el siguiente link: https://github.com/kmolina20/DjangoFileUpload.git. Este proyecto Django cuenta con su directorio raíz y a su vez cuenta con dos directorios dentro de esta. La primera denominada /Core, es donde está el desarrollo de la aplicación, y la carpeta /DjangoFileUpload, cuenta con la configuración del proyecto.

5.1 Configuración del proyecto

Dentro del directorio /DjangoFileUpload, esta el archivo *settings.py* en el cual deben estar las aplicaciones instaladas para que funcionen correctamente el proyecto, agregando 'Core' a su codificación, pues es el nombre del directorio que contiene el desarrollo de la aplicación, como se puede ver en la *Fig. B. 3*.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    "Core",
]
```

Fig. B. 3: Installed_apps dentro de settings.py

Así mismo en el mismo directorio, encontramos el archivo *urls.py*, donde debemos configurar el path de origen que contendrá las urls de la aplicación como se puede ver en la *Fig. B. 4*:

```
urlpatterns = [
    path("", include("Core.urls")),
]
```

Fig. B. 4: urlpatterns dentro de urls.py

5.2 Desarrollo del proyecto

Dentro del directorio */Core*, nos encontramos con diferentes subcarpetas, como se puede ver en la *Fig. B. 5*. En la carpeta */static* se encuentran las carpetas necesarias para los estilos del html. En la carpeta */templates/Core* se encuentran los archivos .html necesarios para el proyecto. Así mismo encontramos los diferentes archivos .py encargados del desarrollo del proyecto.

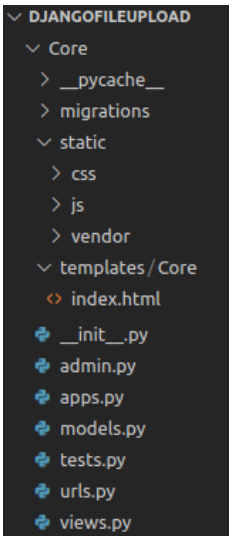


Fig. B. 5: Arquitectura del directorio /Core/

5.2.1 index.html

Dentro de esta implementación se puede destacar el uso del elemento *<form>...</form>*, mismo que sirve para manejar un formulario de ingreso de datos para el usuario como se puede ver en la *Fig. B. 6*, el cual indica que la información

recibida en base al uso del botón de tipo: *submit* en el formulario, será manejada en la función llamada *upload_files* dentro del *views.py* mediante el metodo *POST.*

```
<form "{% url 'Core:upload_files' %}" method="POST" enctype="multipart/form-data">
     <div class="p-t-15">
          <label class="label">Choose a file with .zip format:</label>
          <input class="input--style-4" type="file" name="uploaded_file">
     </div>
     <div class="input-group">
          <label class="label">Version:</label>
          <div class="rs-select2 js-select-simple select--no-search">
               <select name="file_title">
                    <option disabled="disabled" selected="selected">choose a
                         version</option>
                    <option>ecoinvent 3.6 cut off</option>
                    <option>ecoinvent 3.6 apos</option>
                    <option>ecoinvent 3.6 consequential</option>
                    <option>ecoinvent 3.7 cut off</option>
                    <option>ecoinvent 3.7 apos</option>
                    <option>ecoinvent 3.7 consequential</option>
                    <option>ecoinvent 3.7.1 cut off</option>
                    <option>ecoinvent 3.7.1 apos</option>
                    <option>ecoinvent 3.7.1 consequential</option>
               </select>
               <div class="select-dropdown"></div>
          </div>
     </div>
     {% csrf_token %}
     <div class="p-t-15">
          <input class="btn btn--radius-2 btn--blue" type="submit" value="Upload">
     </div>
</form>
```

Fig. B. 6: Seccion de codigo del archivo .html

Cabe recalar el uso de *{% csrf_token %}* antes del botón para que la información pueda recolectarse y enviar a la función requerida.

5.2.2 models.py

Dentro de este archivo, se crean un modelo para manejar los datos ingresados por medio del formulario en el archivo *.html*, con la finalidad de poder acceder a su información después, dado que se está asegurando que el archivo se almacena el el directorio */uploaded_files/* como se indica en la *Fig.B.7*.

```
class Document(models.Model):
    title = models.CharField(max_length = 200)
    uploadedFile = models.FileField(upload_to = "uploaded_files/")
```

Fig. B. 7: Seccion de codigo del archivo models.py

5.2.3 urls.py

En este archivo se deben introducir las urls para el proyecto, en este caso como indica la *Fig. B. 8* se asigna la url *upload/* que hace referencia a la función *views.upload_files*.

```
app_name = "Core"

urlpatterns = [
    path('upload/', views.upload_files, name = "upload_files"),
]

if settings.DEBUG:
    urlpatterns += static(
        settings.MEDIA_URL,
        document_root = settings.MEDIA_ROOT
    )
```

Fig. B. 8: Seccion de codigo del archivo urls.py

5.2.4 views.py

En este archivo se encuentra toda la lógica del proyecto. Se inicia con el método que se observa en la *Fig. B. 9 upload_files*, el cual obtiene los datos ingresados por el usuario en el formulario y le da su correspondiente tratado.

```
def upload_files(request):
    if request.method == "POST":
        #FETCHING THE FORM DATA
        file_title = request.POST["file_title"]
        uploaded_file = request.FILES["uploaded_file"]
        # SAVING THE INFORMATION IN THE DATABASE
        document = models.Document(
            title=file_title,
            uploadedFile=uploaded_file
        )
        document.save()
        #UNZIP THE UPLOADED FILE INTO A DIRECTORY
        with ZipFile(directorio+document.uploadedFile.url, 'r') as zip:
            zip.extractall(directorio+'/temp')
            print('File is unzipped in temp folder')
        #NAME OF THE UNZIP FILE
        a = os.path.split(document.uploadedFile.url)
        folder_name = os.path.splitext(a[1])
        route = dir_temp+folder_name+md
        routeDS = dir_temp+folder_name+ds
        '''post_data = {"username": "admin",
                        "password": "admin"}
        response = requests.post(
            "http://host:port/inti/api_generate_token", data=post_data)
        content = response.content
        token = content.decode('utf8').split('"')'''
        #METHODS TO READ THE FILE AND ADD TO THE DATABASE
        companies(route + 'Companies.xml')
        sources(route + 'Sources.xml')
        people(route + 'Persons.xml')
        activity_name(route + 'ActivityNames.xml')
        geography(route + 'Geographies.xml')
        unit(route + 'Geographies.xml')
        intermediateExchange(route + 'IntermediateExchanges.xml')
        system_model(route + 'SystemModels.xml')
        property(route + 'Properties.xml')
```

```
        id_version = search_version(document.title)
        activityIndexEntry(route + 'ActivityIndex.xml', id_version)
        leerActividadGenerica(routeDS, id_version)
        # DELETE THE TEMPORARY AND THE UPLOADED FILE
        rmtree(dir_temp+'ecoinvent 3.6_apos_ecoSpold02')
        rmtree(directorio+'/media/uploaded_files')

    documents = models.Document.objects.all()

    return render(request, "Core/index.html", context={
        "files": documents
    })
```

Fig. B. 9: Función upload_files del archivo views.py.

Lo primero que se hace es obtener las variables ingresadas, ya sean el nombre y el archivo en sí para poder asignarlo al modelo creado y a su vez almacenarlo en el directorio definido */media/upload_files/*. Con esto se garantiza que conocemos su ubicación dentro del directorio del proyecto y podremos acceder al mismo.

Recordando que el archivo se sube en formato *.zip*, lo debemos descomprimir, en este caso le estamos asignando a un nuevo directorio llamado */temp/*. Se puede ver la jerarquía del proyecto en la *Fig. B. 10* cómo ha cambiado, después de guardar el archivo y de descomprimirlo.

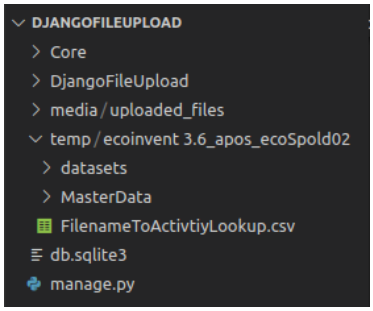


Fig. B. 10: Arquitectura del proyecto DjangoFileUpload.

Ahora se debe asignar las rutas para los directorios como se puede ver en la *Fig. B. 11*, como los directorios dependen de la ubicación del proyecto DjangoFileUpload/ (carpeta raíz). En este caso, la variable *dir_temp* es el directorio creado para descomprimir el archivo y los directorios *md* y *ds* corresponden a las carpetas que están dentro del archivo que fue subido por el usuario.

```
directorio = '[...]/DjangoFileUpload'
dir_temp = '[...]/DjangoFileUpload/temp/'
md = '/MasterData/'
```

```
ds = '/datasets/'
```

Fig. B. 11: Seccion de codigo del archivo views.py

Continuando con la lectura de la función, nos encontramos con que se van a llevar a cabo una serie de métodos para la lectura de los diferentes archivos de formato .xml que forman parte del directorio subido por el usuario. Estas funciones se llevan a cabo de manera ordenada siguiendo la estructura de poblado para la base de datos. Se inicia con los archivos que están en el directorio *md*. Estas funciones reciben como parámetro el directorio específico del archivo a acceder, excepto la última función que aparece en este apartado que accede al directorio *ds*.

5.2.4.1 Función: *companies*

La función accede al directorio ingresado como parámetro e inicia su proceso de lectura en base al funcionamiento de un archivo *.xml*, se obtienen los atributos necesarios para almacenar en la base de datos según la estructura del archivo. Una vez obtenidas las variables se procede a realizar un proceso de validación por medio del método GET sobre la API, donde se consulta si ya existe el identificador único de esta tabla. Si la respuesta es que no existe dicho identificador, se agrega en la base de datos por medio del método POST; o, caso contrario, se omite el ingresar la tupla. A continuación, en la *Fig. B. 12* se presenta el código correspondiente al proceso mencionado.

```
def companies(route):
    xmlReader = minidom.parse(route)
    companies = xmlReader.getElementsByTagName("company")
    for company in companies:
        id = company.getAttribute("id")
        code = company.getAttribute("code")
        website = company.getAttribute("website")
        if len(company.getElementsByTagName("name")) != 0:
            try:
                name = company.getElementsByTagName("name")[0].firstChild.data
            except AttributeError:
                name = "not name provided"
        else:
            name = "not name provided"
        if len(company.getElementsByTagName("comment")) != 0:
            try:
                comment = company.getElementsByTagName("comment")[0].firstChild.data
```

```
            except AttributeError:
                comment = "not comment provided"
        else:
            comment = "not comment provided"
        '''GET THE COMPANY ID '''
        response = requests.get('http://host:port/inti/companies/'+id)
        if response.status_code == 404:
            post_data = {"id": id,
                         "code": code,
                         "name": name,
                         "website": website,
                         "comment": comment}
            response = requests.post(
                "http://host:port/inti/companies/", data=post_data)
        else:
            if response.status_code == 200:
                print("already exist")
    return
```

Fig. B. 12: Función companies()

5.2.4.2 Función: *sources*

La función accede al directorio ingresado como parámetro e inicia su proceso de lectura en base al funcionamiento de un archivo *.xml*. Se obtienen los atributos necesarios para almacenar en la base de datos según la estructura del archivo. Una vez obtenidas las variables se procede a realizar un proceso de validación por medio del método GET sobre la API, donde se consulta si ya existe el identificador único de esta tabla. Si la respuesta es que no existe dicho identificador, se agrega en la base de datos por medio del método POST; o, caso contrario, se omite el ingresar la tupla. A continuación, en la *Fig. B. 13* se presenta el código correspondiente al proceso mencionado.

```
def sources(route):
    xmlReader = minidom.parse(route)
    sources = xmlReader.getElementsByTagName("source")
    for source in sources:
        source_id = source.getAttribute("id")
        source_type = source.getAttribute("sourceType")
        year = source.getAttribute("year")
        volume_no = source.getAttribute("volumeNo")
        first_author = source.getAttribute("firstAuthor")
        additional_authors = source.getAttribute("additionalAuthors")
        title = source.getAttribute("title")
        names_of_editors = source.getAttribute("namesOfEditors")
        short_name = source.getAttribute("shortName")
        page_numbers = source.getAttribute("pageNumbers")
        journal = source.getAttribute("journal")
        title_of_anthology = source.getAttribute("titleOfAnthology")
        place_of_publications = source.getAttribute("placeOfPublications")
```

```
        publisher = source.getAttribute("publisher")
        if len(source.getElementsByTagName("comment")) != 0:
            try:
                aux = ""
                comment = ""
                j = 0
                for comm in source.getElementsByTagName("comment"):
                    aux = source.getElementsByTagName(
                        "comment")[j].firstChild.nodeValue
                    comment = comment + '_' + aux
                    j += 1
            except AttributeError:
                comment = "not comment provided by the provider"
        else:
            comment = "not comment provided by the provider"
        '''GET THE SOURCES ID '''
        response = requests.get(
            'http://host:port/inti/sources/'+source_id)
        if response.status_code == 404:
            post_data = {"id": source_id,
                         "type": source_type,
                         "year": year,
                         "volume_no": volume_no,
                         "first_author": first_author,
                         "additional_authors": additional_authors,
                         "title": title,
                         "names_of_editors": names_of_editors,
                         "short_name": short_name,
                         "page_numbers": page_numbers,
                         "journal": journal,
                         "title_of_anthology": title_of_anthology,
                         "place_of_publications": place_of_publications,
                         "publisher": publisher,
                         "comment": comment}
            response = requests.post(
                "http://host:port/inti/sources/", data=post_data)
            content = response.content
    return
```

Fig. B. 13: Función sources()

5.2.4.3 Función: *people*

La función accede al directorio ingresado como parámetro e inicia su proceso de lectura en base al funcionamiento de un archivo *.xml*, se obtienen los atributos necesarios para almacenar en la base de datos según la estructura del archivo. En este caso se analizó que no todos los archivos contenían uno de los atributos necesarios, por lo que se procedió a asignar un identificador neutro; en este caso, a la variable *company_id* para que no se quede vacío. Una vez obtenidas las variables se procede a realizar un proceso de validación por medio del método

GET sobre la API, donde se consulta si ya existe el identificador único de esta tabla. Si la respuesta es que no existe dicho identificador, se agrega en la base de datos por medio del método POST; o, caso contrario, se omite el ingresar la tupla. A continuación, en la *Fig. B. 14* se presenta el código correspondiente al proceso mencionado.

```
def people(route):
    xmlReader = minidom.parse(route)
    sources = xmlReader.getElementsByTagName("person")
    for source in sources:
        person_id = source.getAttribute("id")
        person_name = source.getAttribute("name")
        person_address = source.getAttribute("address")
        person_telephone = source.getAttribute("telephone")
        person_telefax = source.getAttribute("telefax")
        person_email = source.getAttribute("email")
        company_id = source.getAttribute("companyId")
        if len(source.getAttribute("companyId")) == 0:
            company_id = "00000000-0000-0000-0000-000000000000"
        '''GET THE PERSON ID '''
        response = requests.get(
            'http://host:port/inti/people/'+person_id)
        if response.status_code == 404:
            post_data = {"id": person_id,
                         "name": person_name,
                         "email": person_email,
                         "address": person_address,
                         "telephone": person_telephone,
                         "telefax": person_telefax,
                         "company_id": company_id}
            response = requests.post(
                "http://host:port/inti/people/", data=post_data)
            content = response.content
    return
```

Fig. B. 14: Función people()

5.2.4.4 Función: *activity_name*

La función accede al directorio ingresado como parámetro e inicia su proceso de lectura en base al funcionamiento de un archivo *.xml*, se obtienen los atributos necesarios para almacenar en la base de datos según la estructura del archivo. Una vez obtenidas las variables se procede a realizar un proceso de validación por medio del método GET sobre la API, donde se consulta si ya existe el identificador único de esta tabla. Si la respuesta es que no existe dicho identificador, se agrega en la base de datos por medio del método POST; o, caso contrario, se omite el

ingresar la tupla. A continuación, en la *Fig. B. 15* se presenta el código correspondiente al proceso mencionado.

```
def activity_name(route):
    xmlReader = minidom.parse(route)
    activity_names = xmlReader.getElementsByTagName("activityName")
    for activity_name in activity_names:
        activity_name_id = activity_name.getAttribute("id")
        if len(activity_name.getElementsByTagName("name")) != 0:
            try:
                activity_name = activity_name.getElementsByTagName("name")[
                    0].firstChild.data
            except AttributeError:
                activity_name = "not name provided by the provider"
        else:
            activity_name = "not name provided by the provider"
        '''GET THE ACTIVITY_NAME ID '''
        response = requests.get(
            'http://host:port/inti/activities_name/'+activity_name_id)
        if response.status_code == 404:
            post_data = {"id": activity_name_id,
                         "activity_name": activity_name}
            response = requests.post(
                "http://host:port/inti/activities_name/", data=post_data)
            content = response.content
    return
```

Fig. B. 15: Función activity_name()

5.2.4.5 Función: *geography*

La función accede al directorio ingresado como parámetro e inicia su proceso de lectura en base al funcionamiento de un archivo *.xml*, se obtienen los atributos necesarios para almacenar en la base de datos según la estructura del archivo. Una vez obtenidas las variables se procede a realizar un proceso de validación por medio del método GET sobre la API, donde se consulta si ya existe el identificador único de esta tabla. Si la respuesta es que no existe dicho identificador, se agrega en la base de datos por medio del método POST; o, caso contrario, se omite el ingresar la tupla. A continuación, en la *Fig. B. 16* se presenta el código correspondiente al proceso mencionado.

```
def geography(route):
    xmlReader = minidom.parse(route)
    geographies = xmlReader.getElementsByTagName("geography")
    for geography in geographies:
```

```
        geography_id = geography.getAttribute("id")
        longitude = geography.getAttribute("longitude")
        latitude = geography.getAttribute("latitude")
        un_code = geography.getAttribute("uNCode")
        un_region_code = geography.getAttribute("uNRegionCode")
        un_subregion_code = geography.getAttribute("uNSubregionCode")
        if len(geography.getElementsByTagName("name")) != 0:
            try:
                name = geography.getElementsByTagName(
                    "name")[0].firstChild.data
            except AttributeError:
                name = "not name provided by the provider"
        else:
            name = "not name provided by the provider"
        if len(geography.getElementsByTagName("shortname")) != 0:
            try:
                short_name = geography.getElementsByTagName("shortname")[
                    0].firstChild.data
            except AttributeError:
                short_name = "not shortname provided by the provider"
        else:
            short_name = "not shortname provided by the provider"
        '''GET THE GEOGRAPHY ID '''
        response = requests.get(
            'http://host:port/inti/geographies/'+geography_id)
        if response.status_code == 404:
            post_data = {"id": geography_id,
                         "longitude": longitude,
                         "latitude": latitude,
                         "un_code": un_code,
                         "un_region_code": un_region_code,
                         "un_subregion_code": un_subregion_code,
                         "name": name,
                         "short_name": short_name}
            response = requests.post(
                "http://host:port/inti/geographies/", data=post_data)
            content = response.content
    return
```

Fig. B. 16: Función geography()

5.2.4.6 Función: *unit*

La función accede al directorio ingresado como parámetro e inicia su proceso de lectura en base al funcionamiento de un archivo *.xml*, se obtienen los atributos necesarios para almacenar en la base de datos según la estructura del archivo. Una vez obtenidas las variables se procede a realizar un proceso de validación por medio del método GET sobre la API, donde se consulta si ya existe el identificador único de esta tabla. Si la respuesta es que no existe dicho identificador, se agrega en la base de datos por medio del método POST; o, caso contrario, se omite el

ingresar la tupla. A continuación, en la *Fig. B. 17* se presenta el código correspondiente al proceso mencionado.

```
def unit(route):
    xmlReader = minidom.parse(route)
    units = xmlReader.getElementsByTagName("unit")
    for unit in units:
        unit_id = unit.getAttribute("id")
        if len(unit.getElementsByTagName("name")) != 0:
            try:
                unit_name = unit.getElementsByTagName(
                    "name")[0].firstChild.data
            except AttributeError:
                unit_name = "not name provided by the provider"
        else:
            unit_name = "not name provided by the provider"
        if len(unit.getElementsByTagName("comment")) != 0:
            try:
                unit_comment = unit.getElementsByTagName("comment")[
                    0].firstChild.data
            except AttributeError:
                unit_comment = "not comment provided by the provider"
        else:
            unit_comment = "not comment provided by the provider"
        '''GET THE UNIT ID '''
        response = requests.get(
            'http://host:port/inti/units/'+unit_id)
        if response.status_code == 404:
            post_data = {"id": unit_id,
                         "name": unit_name,
                         "comment": unit_comment}
            response = requests.post(
                "http://host:port/inti/units/", data=post_data)
            content = response.content
    return
```

Fig. B. 17: Función unit()

5.2.4.7 Función: *intermediateExchange*

La función accede al directorio ingresado como parámetro e inicia su proceso de lectura en base al funcionamiento de un archivo *.xml.* Se obtienen los atributos necesarios para almacenar en la base de datos según la estructura del archivo. Una vez obtenidas las variables se procede a realizar un proceso de validación por medio del método GET sobre la API, donde se consulta si ya existe el identificador único de esta tabla. Si la respuesta es que no existe dicho identificador, se agrega en la base de datos por medio del método POST; o, caso contrario, se omite el ingresar la tupla. A continuación, en la *Fig. B. 18* se presenta el código

correspondiente al proceso mencionado.

```
def intermediateExchange(route):
    print("intermediateExchange")
    xmlReader = minidom.parse(route)
    intermediateExchanges = xmlReader.getElementsByTagName(
        "intermediateExchange")
    for intermediateExchange in intermediateExchanges:
        id = intermediateExchange.getAttribute("id")
        unit_id = intermediateExchange.getAttribute("unitId")
        if len(intermediateExchange.getElementsByTagName("name")) != 0:
            try:
                name = intermediateExchange.getElementsByTagName("name")[
                    0].firstChild.data
            except AttributeError:
                name = "not name provided by the provider"
        else:
            name = "not name provided by the provider"
        '''GET THE INTERMEDIATE_EXCHANGES ID  '''
        response = requests.get(
            'http://host:port/inti/intermediate_exchanges/'+id)
        if response.status_code == 404:
            post_data = {"id": id,
                         "unit_id": unit_id,
                         "name": name}
            response = requests.post(
                "http://host:port/inti/intermediate_exchanges/", data=post_data)
            content = response.content
    return
```

Fig. B. 18: Función intermediateExchange()

5.2.4.8 Función: *system_model*

La función accede al directorio ingresado como parámetro e inicia su proceso de lectura en base al funcionamiento de un archivo *.xml*. Se obtienen los atributos necesarios para almacenar en la base de datos según la estructura del archivo. Una vez obtenidas las variables se procede a realizar un proceso de validación por medio del método GET sobre la API, donde se consulta si ya existe el identificador único de esta tabla. Si la respuesta es que no existe dicho identificador, se agrega en la base de datos por medio del método POST; o, caso contrario, se omite el ingresar la tupla. A continuación, en la *Fig. B. 19* se presenta el código correspondiente al proceso mencionado.

```
def system_model(route):
    xmlReader = minidom.parse(route)
```

```
    systemModels = xmlReader.getElementsByTagName("systemModel")
    for systemModel in systemModels:
        system_model_id = systemModel.getAttribute("id")
        if len(systemModel.getElementsByTagName("name")) != 0:
            try:
                system_model_name = systemModel.getElementsByTagName("name")[
                    0].firstChild.data
            except AttributeError:
                system_model_name = "not name provided by the provider"
        else:
            system_model_name = "not name provided by the provider"
        if len(systemModel.getElementsByTagName("shortname")) != 0:
            try:
                system_model_short_name =

systemModel.getElementsByTagName("shortname")[0].firstChild.data
            except AttributeError:
                system_model_short_name = "not shortname provided by the provider"
        else:
            system_model_short_name = "not shortname provided by the provider"
        '''GET THE SYSTEM_MODELS ID  '''
        response = requests.get(
            'http://host:port/inti/system_models/'+system_model_id)
        if response.status_code == 404:
            post_data = {"id": system_model_id,
                         "name": system_model_name,
                         "short_name": system_model_short_name}
            response = requests.post(
                "http://host:port/inti/system_models/", data=post_data)
            content = response.content
    return
```

Fig. B. 19: Función system_model()

5.2.4.9 Función: *property*

La función accede al directorio ingresado como parámetro e inicia su proceso de lectura en base al funcionamiento de un archivo *.xml*. Se obtienen los atributos necesarios para almacenar en la base de datos según la estructura del archivo. En este caso se analizó que no todos los archivos contenían uno de los atributos necesarios, por lo que se procedió a asignar un identificador neutro en este caso a la variable *unit_id* para que no se quede vacío. Una vez obtenidas las variables se procede a realizar un proceso de validación por medio del método GET sobre la API, donde se consulta si ya existe el identificador único de esta tabla. Si la respuesta es que no existe dicho identificador, se agrega en la base de datos por medio del método POST; o, caso contrario, se omite el ingresar la tupla. A continuación, en la *Fig. B. 20* se presenta el código correspondiente al proceso

mencionado.

```
def property(route):
    xmlReader = minidom.parse(route)
    properties = xmlReader.getElementsByTagName("property")
    for property in properties:
        property_id = property.getAttribute("id")
        default_variable_name = property.getAttribute("defaultVariableName")
        unit_id = property.getAttribute("unitId")
        if len(property.getAttribute("unitId")) == 0:
            unit_id = "00000000-0000-0000-0000-000000000000"
        if len(property.getElementsByTagName("name")) != 0:
            try:
                property_name = property.getElementsByTagName("name")[
                    0].firstChild.data
            except AttributeError:
                property_name = "not name provided by the provider"
        else:
            property_name = "not name provided by the provider"
        '''GET THE PROPERTIES ID '''
        response = requests.get(
            'http://host:port/inti/properties/'+property_id)
        if response.status_code == 404:
            post_data = {"id": property_id,
                         "unit_id": unit_id,
                         "default_variable_name": default_variable_name,
                         "name": property_name}
            response = requests.post(
                "http://host:port/inti/properties/", data=post_data)
            content = response.content
    return
```

Fig. B. 20: Función property()

5.2.4.10 Función: *activityIndexEntry* & Función: *search_version*

Para la función *activityIndexEntry* se piden dos variables de entrada, una variable con el directorio *md* como se ha trabajado con las funciones hasta el momento y una segunda variable que maneje la versión que ingresó el usuario desde el formulario. Por lo que, primero actuará la función *search_version* que recibe de parámetro el dato de la versión ingresado desde el formulario y es asignado a su correspondiente identificador. A continuación, en la *Fig. B. 21* se presenta el código correspondiente al proceso mencionado.

```
def search_version(nombre_version):
    id_version = ""
    if nombre_version == 'ecoinvent 3.6 cut off':
        id_version = '18'
    elif nombre_version == 'ecoinvent 3.6 apos':
        id_version = '19'
```

```
    elif nombre_version == 'ecoinvent 3.6 consequential':
        id_version = '20'
    elif nombre_version == 'ecoinvent 3.7 cut off':
        id_version = '21'
    elif nombre_version == 'ecoinvent 3.7 apos':
        id_version = '22'
    elif nombre_version == 'ecoinvent 3.7 consequential':
        id_version = '23'
    elif nombre_version == 'ecoinvent 3.7.1 cut off':
        id_version = '24'
    elif nombre_version == 'ecoinvent 3.7.1 apos':
        id_version = '25'
    elif nombre_version == 'ecoinvent 3.7.1 consequential':
        id_version = '26'
    return id_version
```

Fig. B. 21: Función search_version()

Ahora, la función accede al directorio ingresado como parámetro e inicia su proceso de lectura en base al funcionamiento de un archivo *.xml*. Se obtienen los atributos necesarios para almacenar en la base de datos según la estructura del archivo. Una vez obtenidas las variables se procede a realizar un proceso de validación por medio del método POST sobre la API, donde se realiza una consulta personalizada sobre ciertas variables para conocer si una tupla en particular ya existe en esa tabla. Si la respuesta es que no existe, dicha tupla será ingresada con el método POST en la base de datos en dos tablas diferentes para que cumpla con la correspondencia adecuada. Caso contrario, si la tupla consultada sí existe, se vuelve a realizar una consulta personalizada con el método POST para comprobar que otra tupla de datos no esté ingresada en otra tabla. Si no lo está, se ingresan los datos con el método POST en una de las tablas para no generar duplicidad de datos. A continuación, en la *Fig.B.22* se presenta el código correspondiente al proceso mencionado.

```
def activityIndexEntry(route, version):
    xmlReader = minidom.parse(route)
    activityIndexEntry = xmlReader.getElementsByTagName("activityIndexEntry")
    for activityIndex in activityIndexEntry:
        activity_index_id = activityIndex.getAttribute("id")
        activity_name_id = activityIndex.getAttribute("activityNameId")
        geography_id = activityIndex.getAttribute("geographyId")
        start_date = activityIndex.getAttribute("startDate")
        end_date = activityIndex.getAttribute("endDate")
        special_activity_type = activityIndex.getAttribute("specialActivityType")
        system_model_id = activityIndex.getAttribute("systemModelId")
        '''VALIDATION TO KNOW IF A SPECIFIC TUPLE HAS ALREADY BEEN ENTERED'''
        post_data = {"activity_index_id": activity_index_id,
                     "geography_id": geography_id,
                     "start_date": start_date,
```

```
                    "end_date": end_date,
                    "special_activity_type": special_activity_type,
                    "system_model_id": system_model_id}
        response = requests.post(
            "http://host:port/inti/request_activityIndex/", data=post_data)
        al = response.json()
        content = al.get('response')
        if content == '0':
            '''INSERT IN THE TABLE activity_index'''
            post_data = {"id": activity_index_id,
                         "start_date": start_date,
                         "end_date": end_date,
                         "special_activity_type": special_activity_type,
                         "geography": geography_id,
                         "system_model": system_model_id}
            response = requests.post("http://host:port/inti/activities_index/",
                  data=post_data)
            '''INSERT IN THE TABLE version_name_index'''
            post_data = {"activity_index": activity_index_id,
                         "activity_name": activity_name_id,
                         "version": version}
            response = requests.post("http://host:port/inti/
                  version_name_indexes/", data=post_data)
        else:
            post_data = {"activity_index_id": activity_index_id,
                     "activity_name_id": activity_name_id,
                     "version_id": version}
            response = requests.post(
                "http://host:port/inti/request_versionNameIndex/",
                  data=post_data)
            al = response.json()
            content = al.get('response')
            if content == '0':
                '''INSERT IN THE TABLE version_name_index'''
                post_data = {"activity_index": activity_index_id,
                         "activity_name": activity_name_id,
                         "version": version}
                response = requests.post("http://host:port/inti/
                      version_name_indexes/", data=post_data)
    return
```

Fig. B. 22: Función activityIndexEntry()

5.2.4.11 Función: *leerActividadGenerica* & Función: *ordenamientoBurbuja*

En la *Fig. B. 23* se presenta el código correspondiente al proceso que menciona a continuación. La función accede al directorio ingresado como parámetro, antes de iniciar el proceso de lectura. Primero se deben renombrar los archivos debido a que están en formato *.spold* y es un formato que Python no lee. Estos archivos están nombrados en base a $actividad\_flujoReferencia$ por lo que se debe romper esta estructura y renombrar a los archivos tomando como nombre lo que está antes del carácter especial "_" y renombrarlos a un formato *.xml*. Luego, inicia su proceso de lectura en base al funcionamiento de un archivo *.xml*. Se obtienen

los atributos necesarios para almacenar en la base de datos según la estructura del archivo. Una vez obtenidas las variables se procede a realizar los procesos de validación. El primer proceso es insertar en la base de datos los sinónimos para la actividad correspondiente. Están agregados en un elemento del formato .xml por lo que se debe recorrer la estructura e insertar de uno en uno en la tabla pertinente. Luego, se siguen obteniendo los atributos necesarios para almacenar en la base de datos, al analizar la estructura de los archivos, no todos los archivos contaban con las variables *includedActivitiesStart* e *includedActivitiesEnd*, por lo que se procedió a formular una variable llamada *included_processes* que cuenta con la concatenación de ambas variables.

Continúa el proceso de lectura de los atributos necesarios para almacenar en la base de datos, al analizar el atributo *generalComment*, sus elementos estaban desordenados para cada archivo, por que lo que se procedió a formar una matriz con la información que se obtiene y se procedió a ordenarla por medio del método Burbuja y así formar la variable ya ordenada. En la *Fig. B. 24* se presenta el código correspondiente al proceso mencionado.

Continúa el proceso de lectura de los atributos necesarios (*Fig. B. 23*) para almacenar en la base de datos, procediendo con un proceso de validación por medio del método POST sobre la API. Donde se realiza una consulta personalizada sobre una variable para conocer si una tupla en particular ya existe en esa tabla. Si es el caso, se procede a insertar cierta tupla con el método POST en la base de datos en la tabla pertinente para que cumpla con la correspondencia adecuada. Luego de obtener demás atributos, se procede con la siguiente consulta personalizada con el método POST para conocer el último identificador ingresado a la tabla *data_generator_and_publication* y en base a esa respuesta poder ingresar con el método POST en otra tabla una tupla que contendrá el identificador consultado previamente. Se continúa obteniendo los atributos del archivo y se procede con la siguiente consulta personalizada con el método POST para conocer el último identificador ingresado a la tabla *activity* y en base a esa respuesta poder ingresar con el método POST en otra tabla una tupla que

contendrá el identificador consultado previamente.

Finalmente, se obtienen los últimos atributos necesarios del archivo y se realiza una consulta personalizada con el método POST para conocer el identificador único de una persona, y en base a esa respuesta insertar una tupla con el método POST en la base de datos.

```
def leerActividadGenerica(route, version):
    print("renombrando archivos")
    for archivo in os.listdir(route):
        cadena_archivo = archivo
        cortar = cadena_archivo.split('_')
        os.rename(route + cadena_archivo, route + str(cortar[0]) + ".xml")
    print("COMIENZA EL PROCESO")
    for archivo in os.listdir(route):
        print(os.path.join(archivo))
        xmlReader = minidom.parse(route + archivo)
        activityDescriptions = xmlReader.getElementsByTagName("activityDescription")
        i = 0
        general_comment = ""
        for activityDescription in activityDescriptions:
            for activity in activityDescription.getElementsByTagName("activity"):
                activity_index_id = activity.getAttribute("id")
                '''INSERT SYNONYMS OF ACTIVITIES'''
                if len(activity.getElementsByTagName("synonym")) != 0:
                    try:
                        j = 0
                        for syno in activity.getElementsByTagName("synonym"):
                            aux = activity.getElementsByTagName("synonym")
                                  [j].firstChild.nodeValue
                            post_data = {"activity": activity_index_id,
                                         "synonym": aux}
                            response = requests.post("http://host:port/inti/
                                  synonyms/", data=post_data)
                            j += 1
                    except AttributeError:
                        aux = "not synonym provided"
                else:
                    aux = "not synonym provided"
                '''FINISH INSERT SYNONYMS'''
                if len(activity.getElementsByTagName("allocationComment")) != 0:
                    try:
                        allocation_comment = property.getElementsByTagName
                              ("allocationComment")[0].firstChild.data
                    except AttributeError:
                        allocation_comment = "not allocationComment provided by the
                             provider"
                else:
                    allocation_comment = "not allocationComment provided by the
                          provider"
                included_processes = ""
                """VALIDATE WHEN DON'T HAVE AN START -check-"""
                if len(activity.getElementsByTagName("includedActivitiesStart")) != 0:
                    try:
                        included_processes = included_processes +
                             "includedActivitiesStart"+activity.getElementsByTagName
```

```
                        ("includedActivitiesStart")[0].firstChild.data
                except AttributeError:
                    included_processes = included_processes + ""
            """VALIDATE WHEN DON'T HAVE AN END -check-"""
            if len(activity.getElementsByTagName("includedActivitiesEnd")) != 0:
                try:
                    included_processes = included_processes +
                          "includedActivitiesEnd" + activity.getElementsByTagName
                          ("includedActivitiesEnd")[0].firstChild.data
                except AttributeError:
                    included_processes = included_processes + ""
            '''ORDENAMIENTO DE generalComment -check-'''
            for generalComment in activity.getElementsByTagName("generalComment"):
                matriz = []
                if len(generalComment.getElementsByTagName("text")) != 0:
                    for text in generalComment.getElementsByTagName("text"):
                        filas = len(generalComment.getElementsByTagName("text"))
                        columnas = 2
                        for j in range(columnas):
                            if (j == 0):
                                a = str(text.getAttribute("index"))
                            if (j == 1):
                                try:
                                    b = str(generalComment.getElementsByTagName
                                          ("text")[i].firstChild.data)
                                except AttributeError:
                                    b = ""
                        i += 1
                        array = np.array([a, b])
                        matriz.append(array)
                    ordenamientoBurbuja(matriz, len(matriz))
                    for q in range(0, filas):
                        general_comment = general_comment + "\n" + matriz[q][1]
                else:
                    general_comment = ""
    administrativeInformations = xmlReader.getElementsByTagName
            ("administrativeInformation")
    for administrativeInformation in administrativeInformations:
        for dataGeneratorAndPublication in
              administrativeInformation.getElementsByTagName
              ("dataGeneratorAndPublication"):
            personName = dataGeneratorAndPublication.getAttribute("personName")
            source_id = dataGeneratorAndPublication.getAttribute
                  ("publishedSourceId")
            if len(dataGeneratorAndPublication.getAttribute("publishedSourceId"))
                  == 0:
                source_id = "00000000-0000-0000-0000-000000000000"
            is_copyright_protected = dataGeneratorAndPublication.getAttribute
                  ("isCopyrightProtected")
            '''GET THE ID OF THE PERSON BASED ON THE NAME GIVEN THE DIFFERENT ID'S
                  FOR THE VERSIONS'''
            post_data = {"name": personName}
            response = requests.post(
                "http://host:port/inti/request_Persons/", data=post_data)
            al = response.json()
            content = al.get('response')
            content1 = al.get('response1')
            post_data = {"person": content1,
                     "source": source_id,
                     "is_copyright_protected": is_copyright_protected}
            response = requests.post("http://host:port/inti/
```

```
                data_generator_publication/", data=post_data)
        for timePeriod in activityDescription.getElementsByTagName("timePeriod"):
            is_data_valid_for_entire_period = timePeriod.getAttribute
                  ("isDataValidForEntirePeriod")
        for technology in activityDescription.getElementsByTagName("technology"):
            if len(technology.getElementsByTagName("comment")) != 0:
                if len(technology.getElementsByTagName("text")) != 0:
                    try:
                        for comment in technology.getElementsByTagName("comment"):
                            comment_technology = comment.getElementsByTagName
                                  ("text")[0].firstChild.data
                    except AttributeError:
                        comment_technology = comment_technology + ""
            else:
                comment_technology = "NO COMMENT TECHNOLOGY"
        '''GET THE LAST ID OF DATA_GENERATOR_AND_PUBLICATION ENTERED IN THE
              DATABASE'''
        post_data = {"name": '1'}
        response = requests.post(
                "http://host:port/inti/request_DataGeneratorAndPublication/",
                data=post_data)
        al = response.json()
        content = al.get('response')
        content1 = al.get('response1')
        if content == '0':
            post_data = {"activity_index_id": activity_index_id,
                         "allocation_comment": allocation_comment,
                         "general_comment": general_comment,
                         "included_processes": included_processes,
                         "comment_technology": comment_technology,
                         "is_data_valid_for_entire_period": content1,
                         "data_generator_and_publication_id": version,
                         "version_id": source_id}
            response = requests.post("http://host:port/inti/activities/",
                  data=post_data)
    print("flowDatas = xmlReader.getElementsByTagName(flowData)")
    i = 0  # contador para comprobar la cantidad que se ingresa en la tabla
    flowDatas = xmlReader.getElementsByTagName("flowData")
    for flowData in flowDatas:
        for intermediateExchange in flowData.getElementsByTagName
              ("intermediateExchange"):
            intermediate_exchange_id = intermediateExchange.getAttribute
                  ("intermediateExchangeId")
            variable_name = intermediateExchange.getAttribute("variableName")
            if len(intermediateExchange.getElementsByTagName("inputGroup")) != 0:
                try:
                    input_group = intermediateExchange.getElementsByTagName
                          ("inputGroup")[0].firstChild.data
                except AttributeError:
                    input_group = "No Group"
            else:
                input_group = "No Group"
            if len(intermediateExchange.getElementsByTagName("outputGroup")) != 0:
                try:
                    output_group = intermediateExchange.getElementsByTagName
                          ("outputGroup")[0].firstChild.data
                except AttributeError:
                    output_group = "No Group"
            else:
                output_group = "No Group"
            i += 1
```

```
                '''GET THE LAST ID OF ACTIVITY ENTERED IN THE DATABASE'''
                post_data = {"allocation_comment": '1'}
                response = requests.post(
                        "http://host:port/inti/request_Activity/",
                        data=post_data)
                al = response.json()
                content = al.get('response')
                if content == '0':
                    content1 = al.get('response1')
                    post_data = {"activity_id": content1,
                            "intermediate_exchange_id": intermediate_exchange_id,
                            "variable_name": variable_name,
                            "input_group": input_group,
                            "output_group": output_group}
                    response = requests.post("http://host:port/inti/
                          activities_intermediate_exchange/", data=post_data)
        modellingAndValidations = xmlReader.getElementsByTagName
              ("modellingAndValidation")
        for modellingAndValidation in modellingAndValidations:
            for review in modellingAndValidation.getElementsByTagName("review"):
                reviewerName = review.getAttribute("reviewerName")
                '''GET THE ID OF THE PERSON BASED ON THE NAME GIVEN THE DIFFERENT ID'S
                      FOR THE VERSIONS'''
                post_data = {"name": reviewerName}
                response = requests.post(
                        "http://host:port/inti/request_ActivityPerson/",
                        data=post_data)
                al = response.json()
                content = al.get('response')
                if content == '0':
                    content1 = al.get('response1')
                    content2 = al.get('respuesta2')
                    post_data = {"person_id": content1,
                         "activity_id": content2}
                    response = requests.post("http://host:port/inti/
                          activities_intermediate_exchange/", data=post_data)
    return
```

Fig. B. 23: Función leerActividadGenerica()

```
def ordenamientoBurbuja(matriz,tam):
    for i in range(1,tam):
        for j in range(0,tam-i):
            if(int(matriz[j][0]) > int(matriz[j+1][0])):
                n = matriz[j+1][0]
                o = matriz[j+1][1]
                matriz[j+1][0] = matriz[j][0]
                matriz[j+1][1] = matriz[j][1]
                matriz[j][0] = n
                matriz[j][1] = o
```

Fig. B. 24: Función ordenamientoBurbuja()

Una vez detallados estos métodos, un usuario puede conocer la funcionalidad de cada método y de ser necesario podrá acceder a ellos para modificarlos o adaptarlos conforme se necesite.

6 Descripción del manual técnico de la estructuración de la API.

El proyecto Django Rest Framework puede ser encontrado en el siguiente link: https://github.com/kmolina20/IntiDRF-UI.git y debe ser inicializado como un proyecto Django Rest Framework

6.1 Configuración del proyecto

Dentro del directorio IntiDRF-UI/app está el archivo *settings.py* en el cual deben estar las aplicaciones instaladas para que funcionen correctamente el proyecto, como se puede ver en la *Fig. B. 25*.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'rest_framework.authtoken',
    'corsheaders',
    'IntiApp.apps.IntiappConfig',
    'drf_yasg',
    'django_filters',
    'usuario.apps.UsuarioConfig',
]
```

Fig. B. 25: Installed_apps dentro de settings.py

Así mismo, en el mismo directorio, encontramos el archivo *urls.py*, donde debemos configurar el path de origen que contendrá las urls de la aplicación, como se puede ver en la *Fig. B. 26*:

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('inti/', include(('IntiApp.urls','inti'))),
    path('inti/', include('IntiApp.routers')),
    path('accounts/login/', Login.as_view(), name="login"),
    path('logout/', login_required(LogoutUser), name="logout"),
]
```

Fig. B. 26: urlpatterns dentro de urls.py

6.2 Desarrollo del proyecto

Dentro del directorio, nos encontramos con diferentes subcarpetas, como se

puede ver en la *Fig. B. 27*. En la carpeta */IntiApp* se encuentran las carpetas necesarias para la configuración interna del prototipo. En la carpeta */static* se encuentran las carpetas necesarias para los estilos del html. En la carpeta */templates* se encuentran los archivos .html necesarios para la interfaz de usuario del proyecto. Finalmente, en la carpeta */usuario* se encuentra la configuración para el login del prototipo.

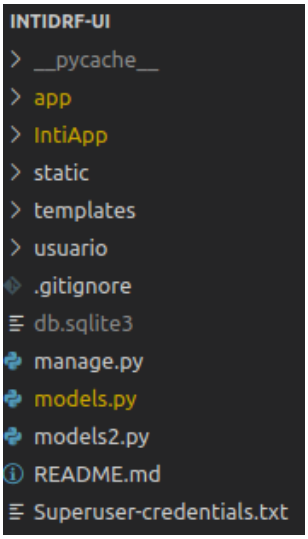
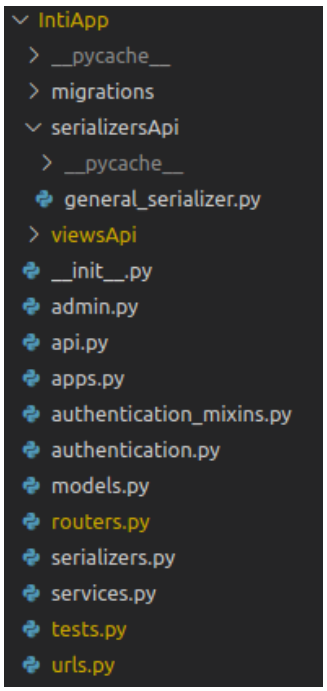


Fig. B. 27: Arquitectura del directorio /Core/

6.2.1 IntiApp

En el directorio IntiDRF-UI/IntiApp, se encuentran una serie de archivos y directorios como se puede ver en la *Fig. B. 28. a*. Y, a su vez, a manera detallada en la *Fig. B. 28. b,* se puede analizar los archivos que están dentro del directorio IntiDRF-UI/IntiApp/viewsApi.

a



b

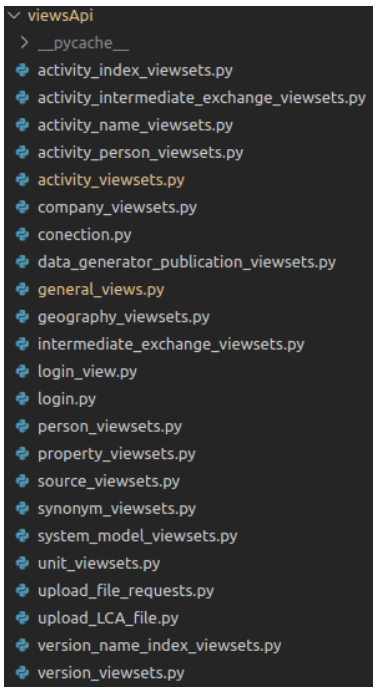


Fig. B. 28: Arquitectura del directorio IntiDRF-UI/IntiApp.

En el directorio *IntiDRF-UI/IntiApp*, se encuentran varios archivos, entre ellos se encuentra *urls.py*, el cual muestra el extracto de código de la Fig. B. 29, que define las URLs que tienen el método login configurado. Este apartado sirve para el control de confidencialidad con respecto a la información que se muestra en el prototipo.

```
urlpatterns=[
    re_path(r'^swagger(?P<format>\.json|\.yaml)$',
        schema_view.without_ui(cache_timeout=0), name='schema-json'),
    path(r'swagger/', schema_view.with_ui('swagger', cache_timeout=0),
        name='schema-swagger-ui'),
    path(r'redoc/', schema_view.with_ui('redoc', cache_timeout=0),
        name='schema-redoc'),
    path('api_generate_token',views.obtain_auth_token, name = 'Generate Toke'),
    path('home/',login_required(general_views.index),name='index'),
    path('alike_flows/',login_required(general_views.alike_flows),name='alike_flows'),
    path('activities_version/',login_required(general_views.activities_version),
        name='activities_version'),
    path('similar_names_activities/',login_required(general_views.similar_names_
        activities),name='similar_names_activities'),
    path('geographies_activities/',login_required(general_views.geographies_
        activities),name='geographies_activities'),
    path('flows/',login_required(general_views.flows),name='flows'),
    path('intermediate_exchange/',login_required(general_views.intermediate
        _exchange),name='intermediate_exchange'),
    path('upload/', login_required(upload_LCA_file.upload_files), name =
        "upload_files"),
    path('query/', login_required(upload_LCA_file.query_files), name = "query_files"),
    path('accounts/login/', LoginView.as_view(template_name='login.html'),
        name="login"),
    path('logout/',logout_then_login, name = 'logout'),
```

```
     #UPLOAD FILE PETITIONS (only for the client .zip)
    path('request_activityIndex/',
        upload_file_requests.RequestActivityIndex.as_view(),name='activityIndex
        Petitions'),
    path('request_versionNameIndex/',
        upload_file_requests.RequestVersionNameIndex.as_view(),name='versionNameIndex
        Petitions'),
    path('request_Persons/', upload_file_requests.RequestPersons.as_view(),
         name='person Petitions'),
    path('request_DataGeneratorAndPublication/',upload_file_requests.
        RequestDataGeneratorAndPublication.as_view(),name='DataGeneratorAndPublication
        Petitions'),
    path('request_Activity/', upload_file_requests.RequestActivity.as_view(),
        name='activity Petitions'),
    path('request_ActivityPerson/',upload_file_requests.RequestActivityPerson.
        as_view(),name='activityPerson Petitions'),
]
```

Fig. B. 29: Extracto de código del archivo IntiDRF-UI/IntiApp/urls.py

También está el archivo *models.py* en el cual están los modelos de la base de datos. Como se puede ver en la *Fig. B. 30*, un extracto del código donde están los modelos de todas las tablas con sus respectivos atributos, claves primarias y claves foráneas, además del nombre de la tabla que hacen referencia.

```
from django.db import models

class AcitivityIntermediateExchange(models.Model):
    activity = models.ForeignKey('Activity', models.DO_NOTHING)
    intermediate_exchange = models.ForeignKey('IntermediateExchange',
models.DO_NOTHING)
    variable_name = models.CharField(max_length=1000, blank=True, null=True)
    id = models.BigAutoField(primary_key=True)
    input_group = models.CharField(max_length=10, blank=True, null=True)
    output_group = models.CharField(max_length=10, blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'acitivity_intermediate_exchange'
.
.
.
```

Fig. B. 30: Extracto de código del archivo IntiDRF-UI/IntiApp/models.py

Así mismo, está el archivo *routers.py* en el cual están definidas las URLs estándar según las tablas de la base de datos, como se puede ver en la *Fig.B.30*. Este archivo define las URLs que se muestran en la vista API Root.

```
router.register(r'activities',activity_viewsets.ActivityViewSet, basename= 'activity')
router.register(r'versions',version_viewsets.VersionViewSet, basename= 'version')
router.register(r'companies',company_viewsets.CompanyViewSet, basename='company')
router.register(r'people',person_viewsets.PersonViewSet, basename='person')
router.register(r'geographies',geography_viewsets.GeographyViewSet,
     basename='geography')
router.register(r'units',unit_viewsets.UnitViewSet, basename='unit')
router.register(r'activities_intermediate_exchange',activity_intermediate_exchange_vie
     wsets.ActivityIntermediateExchangeViewSet, basename= 'activity intermediate
```

```
     exchange')
router.register(r'activities_index',activity_index_viewsets.ActivityIndexViewSet,
     basename= 'activity index')
router.register(r'activities_name',activity_name_viewsets.ActivityNameViewSet,
     basename= 'activity name')
router.register(r'activities_person',activity_person_viewsets.ActivityPersonViewSet,
     basename= 'activity person')
router.register(r'data_generator_publications',data_generator_publication_viewsets.Dat
     aGeneratorAndPublicationViewSet, basename= 'data generator and publications')
router.register(r'intermediate_exchanges',intermediate_exchange_viewsets.IntermediateE
     xchangeViewSet, basename= 'intermediate exchange')
router.register(r'properties',property_viewsets.PropertyViewSet, basename= 'property')
router.register(r'sources',source_viewsets.SourceViewSet, basename= 'source')
router.register(r'synonyms',synonym_viewsets.SynonymViewSet, basename= 'synonym')
router.register(r'system_models',system_model_viewsets.SystemModelViewSet, basename=
     'system model')
router.register(r'version_name_indexes',version_name_index_viewsets.VersionNameIndexVi
     ewSet, basename= 'version name index')
```

Fig. B. 30: Extracto de código del archivo IntiDRF-UI/IntiApp/routers.py

Ahora, al analizar el directorio *IntiDRF-UI/IntiApp/serializersApi* se encuentra el archivo *general_serializer.py* el cual, como se puede ver en el código de la Fig. B. 31, hace referencia al nombre del modelo de la tabla que se corresponde con el archivo *models.py* que se obtuvo a partir de la base de datos como indicó la *Fig. B. 30.* Y de ser el caso y contener claves foráneas, hace referencia al serializador que corresponde del mismo archivo *general_serializer.py.*

```
from IntiApp.models import Version, Activity, Company, Person, Geography, Unit,
    ActivityIntermediateExchange, ActivityIndex, ActivityName, ActivityPerson,
    DataGeneratorAndPublication, IntermediateExchange, Property, Source, Synonym,
    SystemModel, VersionNameIndex, RequestActivityIndex, RequestIdName, RequestName,
    RequestId, RequestVersionNameIndex
from rest_framework import serializers

class VersionSerializer(serializers.ModelSerializer):
    class Meta:
        model = Version
        fields = '__all__'

class CompanySerializer(serializers.ModelSerializer):
    class Meta:
        model = Company
        fields = '__all__'

class PersonSerializer(serializers.ModelSerializer):
    company = CompanySerializer()

    class Meta:
        model = Person
        fields = '__all__'
.
.
.
```

Fig. B. 31: Extracto de código del archivo models.py

Finalmente, al analizar el directorio *IntiDRF-UI/IntiApp/viewsApi* se encuentra una serie de archivos que corresponden a cada tabla de la base de datos. El

motivo de esto, es que para cada tabla se debe crear una vista y por eso en la *Fig. B. 30*, se detallaron las URLs a las cuales corresponden estas vistas. A manera de ejemplificación se mostrará una de ellas.

El código del archivo *activity_index_viewsets.py* se puede ver en la *Fig.B.32*, la cual referencia la tabla *activity_index* de la base de datos. Analizando el código de la clase *ActivityIndexViewSet*, se nota como referencia al método correspondiente de la misma tabla, pero del archivo *general_serializer*. Ahora, los métodos que contiene esta clase está enfocado en cumplir la función CRUD, para cada tabla de la base de datos, por ello tienen la codificación que muestra a continuación.

```
from rest_framework import viewsets, status
from rest_framework.response import Response
from rest_framework.authentication import TokenAuthentication


from IntiApp.serializersApi import general_serializer

class ActivityIndexViewSet(viewsets.ModelViewSet):
    serializer_class = general_serializer.ActivityIndexSerializer
    authentication_classes = (TokenAuthentication,)

    def get_queryset(self, pk=None):
        if pk is None:
            return self.get_serializer().Meta.model.objects.all()
        else:
            return self.get_serializer().Meta.model.objects.filter(id = pk).first()

    def create(self, request):
        serializer = self.serializer_class(data = request.data)
        if serializer.is_valid():
            serializer.save()
            return Response({'message': 'Activity Index created successfully'},status=
status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    def update(self, request, pk=None):
        if self.get_queryset(pk):
            activity_index_serializer = self.serializer_class(self.get_queryset(pk),
data = request.data)
            if activity_index_serializer.is_valid():
                activity_index_serializer.save()
                return Response(activity_index_serializer.data, status=
status.HTTP_200_OK)
            return Response(activity_index_serializer.errors, status=
status.HTTP_400_BAD_REQUEST)


    def destroy(self, request, pk=None):
        activityIndex = self.get_queryset().filter(id = pk).first()
        if activityIndex:
            activityIndex.delete()
            return Response({'message':'Activity Index removed successfully'}, status=
status.HTTP_200_OK)
        return Response({'message':'No Activity Index found with this ID'}, status=
status.HTTP_400_BAD_REQUEST)
```

Fig. B. 32: Extracto de código del archivo activity_index_viewsets.py

Continuando con este directorio, en estas clases viewset se encuentra también la configuración de las consultas personalizadas, de igual manera, como ejemplo se dará detalle de una función. En este caso, en la clase *activity_viewsets.py*, como se puede ver en el código de la *Fig. B. 32*, es un extracto de código de la clase *activity_viewsets.* Esta función así como las demás desarrolladas, implementan el código de la consulta personalizada específica.

```
@action(detail=True)
    def flows(self, request, pk=None):
        v_id = request.GET.get('version', '') # version = ecoinvent 3.6 cut off
        data = {}
        data = []
        cursor1 = conexion.cursor()
        if len(v_id)==0:
            select = "SELECT IE.ID, IE.NAME, v.version, an.id, an.ACTIVITY_NAME, ai.id
                FROM ACTIVITY_INTERMEDIATE_EXCHANGE AIE, INTERMEDIATE_EXCHANGE IE,
                ACTIVITY A, ACTIVITY_NAME AN, VERSION_NAME_INDEX VNI, ACTIVITY_INDEX
                AI, VERSION V WHERE AIE.OUTPUT_GROUP = '0' AND
                AIE.INTERMEDIATE_EXCHANGE_ID = IE.ID AND AIE.ACTIVITY_ID = A.ID AND
                A.ACTIVITY_INDEX_ID = AI.ID AND VNI.ACTIVITY_INDEX_ID = AI.ID AND
                VNI.ACTIVITY_NAME_ID = AN.ID AND VNI.VERSION_ID = V.ID AND
                a.id='"+pk+"'"
        else:
            select = "SELECT IE.ID, IE.NAME, v.version, an.id, an.ACTIVITY_NAME ai.id
                FROM ACTIVITY_INTERMEDIATE_EXCHANGE AIE, INTERMEDIATE_EXCHANGE IE,
                ACTIVITY A, ACTIVITY_NAME AN, VERSION_NAME_INDEX VNI, ACTIVITY_INDEX
                AI, VERSION V WHERE AIE.OUTPUT_GROUP = '0' AND
                AIE.INTERMEDIATE_EXCHANGE_ID = IE.ID AND AIE.ACTIVITY_ID = A.ID AND
                A.ACTIVITY_INDEX_ID = AI.ID AND VNI.ACTIVITY_INDEX_ID = AI.ID AND
                VNI.ACTIVITY_NAME_ID = AN.ID AND VNI.VERSION_ID = V.ID AND V.version =
                '"+v_id+"' and a.id='"+pk+"'"
        cursor1.execute(select)
        select = cursor1.fetchall()
        for row in select:
            data.append({
                "intermediate exchange id": str(row[0]),
                "intermediate exchange name": str(row[1]),
                "version": str(row[2]),
                "activity name id": str(row[3]),
                "activity name": str(row[4]),
                "activity index id": str(row[5])
            })
        s1 = json.dumps(data)
        d1 = json.loads(s1)
        d2=self.paginate_queryset(d1)
        if len(select) == 0:
            return Response({'response': 'no data found'})
        else:
            return Response(d2)
```

Fig. B. 31: Extracto de código del archivo models.py

La y las funciones de las consultas personalizadas cumplen el siguiente proceso. Se crea una lista donde se van a almacenar las tablas consultadas. Estas tuplas se generan a partir de una consulta sql, cabe mencionar que las consultas sql son diferentes para cada función pues dependen de lo que deseen responder. En este caso, la consulta sql obtiene todos los flujos de referencia asociados con

una actividad, filtrado por el id de la actividad y/o por el nombre de la versión. Una vez se obtiene la consulta y se ingresan los datos en el objeto, se construye un archivo JSON a partir del método *.dumps* y el método *.loads*. En este caso el JSON respuesta también se lo procesa por el método *paginate_queryset*, para evitar el lag en la carga de datos y así finalmente se envía la respuesta al html que corresponde la función.

Anexo 3: Carta Soporte del Proyecto de Titulación.

UNIVERSITY OF CALIFORNIA, SANTA BARBARA

BERKELEY ▯ DAVIS ▯ IRVINE ▯ MERCED ▯ LOS ANGELES ▯ RIVERSIDE ▯ SAN DIEGO ▯ SAN FRANCISCO

SANTA BARBARA ▯ SANTA CRUZ

INSTITUTE FOR SOCIAL, BEHAVIORAL AND ECONOMIC RESEARCH
2201 North Hall SANTA BARBARA, CA 93106-2150
PHONE: 805.893.2548 FAX: 805.893.7995 EMAIL: INFO@ISBER.UCSB.EDU

http://www.isber.ucsb.edu/

Brandon Kuczenski, PhD.
Associate Researcher
Institute for Social, Behavioral, and Economic Research
University of California, Santa Barbara

March 3, 2022

To the Thesis Review Committee:

I am writing this letter in support of the Master's candidates Kamila Nicole Molina Orellana and Estuardo Mateo Quizhpi Peralta, to whom I have been an external advisor for approximately the past two years.

My background is in industrial ecology, and I specialize in methodological challenges facing life cycle assessment (LCA), which is one of the principal techniques used by industrial ecology researchers. The practice of LCA is greatly constrained by its dependence on large databases which are lacking in both transparency and technical support. There is a specific need for improved Web infrastructure to improve access to metadata surrounding these LCA data resources. This problem is made more challenging by the fact that LCA itself is a highly demanding methodology, and the design of information services to support it require an understanding of both LCA and information technology.

I was recruited by a faculty member at the University of Cuenca, Paul F. Vanegas Peña, to help develop a research project that could serve as a Master's thesis project for the students and also fill some gaps in the LCA technology landscape. The chosen project was to develop a metadata resource for the preeminent life cycle inventory database, known as ecoinvent. In particular, the desired data resource would enable a user to:

(i) browse and search information about many different versions of the ecoinvent database;
(ii) retrieve datasets that share properties in common with a specified dataset; and
(iii) obtain "migration" information that would permit a user to link datasets of one database version to equivalent datasets from prior and subsequent versions.

The task required educating the students on significant elements of LCA methodology, even while they were learning computer engineering. In carrying out the project, the students successfully designed a database schema to capture the necessary information, extracted and loaded the metadata into the database, developed a REST-based application programming interface, and implemented a simple "dashboard" web application. Their contribution does materially advance the practice of LCA by making it easier for users to

migrate complex models between ecoinvent versions. These accomplishments were secured despite several challenges, including the highly technical nature of the ecoinvent source data, complex aspects of LCA methodology, and a language barrier presented by my very rudimentary grasp of the Spanish language. Of course, challenges caused by the COVID pandemic also pervaded the work for the duration of our collaboration. In all cases I found the students to be eager to rise to the challenge. It has been my great pleasure to work with them, and I hope for the public deployment of their work product so that I can make use of it myself.

Warm regards,

Brandon Kuczenski