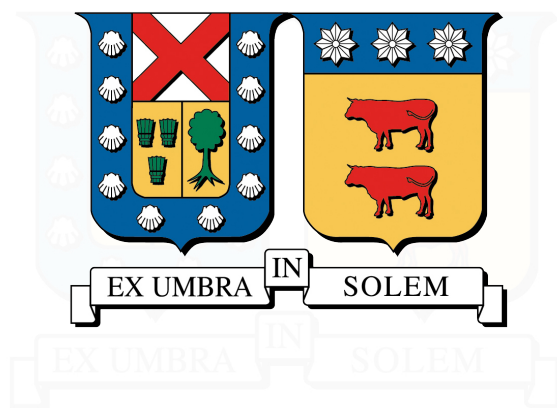


UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA E INFORMÁTICA
CONCEPCION - CHILE



Automatización de la Actualización de Registros GTFS de la Ciudad de Concepción

ABNER ABDIEL ASTETE HERNÁNDEZ

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO EN INFORMÁTICA

PROFESOR GUÍA : SR. ISRAEL FIGUEROA PEDREROS

4 de diciembre de 2023



Mi Dedicatoria . . .

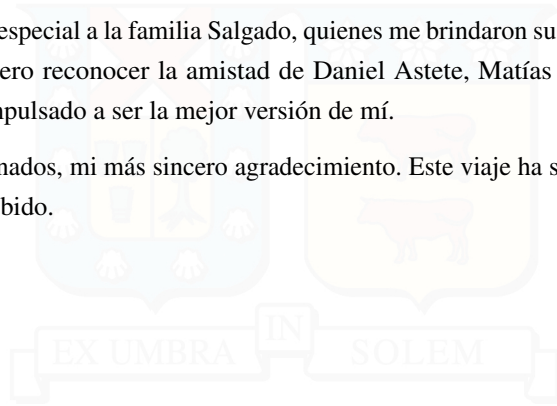
A mis padres que, a pesar de la distancia, siempre han estado conmigo.

AGRADECIMIENTOS

Quiero expresar mi agradecimiento a Dios, quien ha sido mi guía y compañero a lo largo de esta etapa. Agradezco de corazón a mis padres y hermanos, su amor ha sido muy importante para mí durante estos años.

Un agradecimiento especial a la familia Salgado, quienes me brindaron su cariño y abrieron las puertas de su hogar. También, quiero reconocer la amistad de Daniel Astete, Matías Salgado y Fabio Bottarelli, quienes siempre me han impulsado a ser la mejor versión de mí.

A todos los mencionados, mi más sincero agradecimiento. Este viaje ha sido posible gracias al apoyo, amor y amistad que he recibido.



RESUMEN

Este proyecto se presenta como una propuesta centrada en la actualización de la base de datos GTFS en el sistema de transporte de Concepción. Su objetivo central es desarrollar un sistema automatizado que, al recibir nuevos registros, se realice una actualización automática de la base de datos.

La implementación de este sistema ha demostrado resultados positivos al lograr un seguimiento claro de la información y establecer reglas para cargar nuevos datos en el sistema.

Palabras Clave. Actualización automática, base de datos, GTFS, sistema de transporte, Concepción.



ABSTRACT

This project is presented as a proposal focused on updating the GTFS database in the Concepción transportation system. Its main objective is to develop an automated system that, upon receiving new records, performs an automatic update of the database.

The implementation of this system has shown positive results by achieving a clear tracking of information and establishing rules for loading new data into the system.

Keywords. Automatic update, database, GTFS, transportation system, Concepción.



Índice de Contenidos

Índice de Contenidos	IX
Índice de Figuras	XI
Listings	XIII
Índice de Tablas	XV
Introducción	1
1. Problemática	5
1.1. Objetivo General	5
1.2. Objetivos Secundarios	5
1.3. Descripción de la Problemática	5
1.4. Estandares y Estructura de los archivos GTFS	6
1.5. Softwares Existentes	6
1.6. Problematicas tecnicas	6
1.7. Restricciones de Diseño	7
2. Marco Teórico	9
2.1. Secretaría Regional Ministerial (SEREMI):	9
2.2. Python:	9
2.2.1. Bibliotecas Relevantes:	10
2.3. Hypertext Transfer Protocol (HTTP):	10
2.4. General Transit Feed Specification (GTFS) Estático:	11
2.4.1. Agency.txt:	11
2.4.2. Stops.txt:	11
2.4.3. Routes.txt:	11
2.4.4. Trips.txt:	12
2.4.5. StopTimes.txt:	12
2.4.6. Calendar.txt:	12
2.4.7. Shapes.txt:	12
2.5. General Transit Feed Specification (GTFS) en Tiempo Real:	12
2.6. Application Programming Interface (API) y Endpoints:	13
2.7. PostgreSQL:	13
2.8. SQLAlchemy (ORM):	13
2.9. Pandas:	14
2.10. Fast API:	14
2.11. Django:	15
2.12. Docker:	15
2.13. Airflow:	15

3. Solución	17
3.1. Primer Componente: Creación de la Base de Datos para registros GTFS.	17
3.2. Segundo Servicio: Carga Registros GTFS.	22
3.2.1. Endpoint:	23
3.2.2. Enfoque de comparacion de registros:	23
3.2.3. Actualización del estado del ZIP:	26
3.2.4. Procesos de la Carga:	26
3.2.4.1. Identificación del último archivo ZIP ingresado al sistema:	26
3.2.4.2. Extraer ZIP:	27
3.2.4.3. Inspeccionar archivos TXT:	27
3.2.4.4. Transformar los TXT en dataframes:	28
3.2.4.5. Verificar si la Base de Datos tiene registros:	31
3.2.4.6. Lectura de registros en la base de datos:	32
3.2.4.7. Identificar los registros antiguos.	33
3.2.4.8. Identificar los registros nuevos:	36
3.2.4.9. Generar un nuevo ZIP:	39
3.2.4.10. Manejar errores en la carga:	39
4. Validación de la Solución	41
4.1. Ejecución del componente de Carga de datos GTFS:	41
5. Conclusiones - Trabajos Futuros	45
5.1. Conclusiones:	45
5.2. Trabajos Futuros	45
Bibliografía	47
A. Apendice A	49
A.1. Endpoint de la Carga:	49
A.2. Contenido archivos GTFS:	53

Índice de Figuras

1. Área de enfoque del proyecto.	3
2.1. Fast API.	15
2.2. Django.	16
2.3. Docker.	16
2.4. Airflow.	16
3.1. Base de Datos GTFS.	21
4.1. Panel para ejecucion de endpoint.	42
4.2. Primera Carga.	42
4.3. Datos de la Primera Carga ingresado a la base de Datos.	42
4.4. Segunda Carga.	42
4.5. Datos de la Segunda Carga ingresado a la base de Datos.	43



Listings

3.1. Conexión a la base de datos usando SQLAlchemy	18
3.2. Definición de la clase CargaGtfs	18
3.3. Clase ShapeReference	19
3.4. Clase Shapes	19
3.5. Clase Stop	20
3.6. Configuración de la base de datos	22
3.7. Definición de la clase 'CargaGtfs'	23
3.8. Comparacion de versiones en routes.txt	24
3.9. Comparacion de versiones en stop.txt	25
3.10. Actualizar estado de la Carga en la Base de Datos	26
3.11. Función para identificar el último ZIP en la base de datos.	26
3.12. Función para extraer archivos de un ZIP y actualizar el estado en la base de datos.	27
3.13. Función para inspeccionar archivos TXT y actualizar el estado en la base de datos.	28
3.14. Función para transformar el archivo 'agency.txt' a un DataFrame y verificar las columnas necesarias.	29
3.15. Función para verificar las columnas de un DataFrame y filtrarlo según las columnas necesarias.	29
3.16. Transformación del archivo stops.txt a un DataFrame y verificación de columnas.	30
3.17. Transformación del archivo shapes.txt a un DataFrame y generación de shapes _{reference}	31
3.18. Verificar si la base de datos tiene registros en varias tablas.	31
3.19. Cargar datos desde un DataFrame a una tabla de la base de datos.	32
3.20. Extraer datos de la base de datos a DataFrames.	32
3.21. Verificar y eliminar registros obsoletos.	33
3.22. Eliminar registros obsoletos de una tabla.	34
3.23. Eliminar registros obsoletos de la tabla 'stop _{times} '.	35
3.24. Verificar y agregar nuevos registros a las tablas correspondientes.	36
3.25. Agregar nuevos registros a una tabla en la base de datos.	37
3.26. Agregar nuevos registros a la tabla 'stops' en la base de datos.	38
3.27. Finalizar el proceso de carga y generar un nuevo archivo ZIP.	39
A.1. Endpoint de la Carga.	49



Índice de Tablas

A.1. Ejemplo de Datos del archivo: Agency.txt	53
A.2. Ejemplo de Datos del archivo: Calendar.txt	53
A.3. Ejemplo de Datos del archivo: Routes.txt	53
A.4. Ejemplo de Datos del archivo: Shapes.txt	54
A.5. Ejemplo de Datos del archivo: Stop-times.txt	54
A.6. Ejemplo de Datos del archivo: Stops.txt	54
A.7. Ejemplo de Datos del archivo: Trips.txt	54



Introducción

El sistema de transporte de la ciudad de Concepción se encuentra en pleno proceso de digitalización, con el objetivo de mejorar la eficiencia y la experiencia de los usuarios. Uno de los pilares fundamentales de esta transformación es la capacidad de proporcionar información en tiempo real sobre las rutas y horarios de las micros en los paraderos de la ciudad. Para lograr esto, es esencial contar con una base de datos actualizada de manera constante que alimente esta información.

Sin embargo, surge una problemática crucial en este contexto: la falta de una solución de código abierto y gratuita que automatice el proceso de actualización de esta base de datos a partir de la información proporcionada por los usuarios. La mayoría de los sistemas existentes requieren intervención manual para editar y mantener la información, lo que conlleva costos significativos en términos de tiempo y recursos humanos.

Esta limitación impide alcanzar la agilidad y la precisión necesarias para proporcionar información actualizada y confiable a los usuarios del sistema de transporte. Además, la falta de una trazabilidad eficiente de los datos ingresados y su estado de procesamiento puede generar incertidumbre en cuanto a la fiabilidad de la información disponible para el público.

En este contexto, este proyecto se enfoca en diseñar un sistema de automatización que no solo procese y actualice datos en tiempo real, sino que también se adapte a las particularidades locales. Esto implica la creación de adaptadores personalizados para incorporar fuentes de datos específicas de Concepción y garantizar que la información proporcionada sea precisa y relevante para los usuarios locales.

La propuesta consiste en el desarrollo de una automatización que permita procesar y actualizar la base de datos de forma automática cada vez que un usuario ingrese nueva información de datos GTFS. Esta automatización agregará los datos nuevos y eliminará los datos obsoletos, proporcionando una trazabilidad de todo el proceso.

La implementación de esta automatización no solo mejorará la precisión y la actualización de la información en el sistema de transporte de la ciudad de Concepción, sino que también reducirá los costos operativos y brindará una experiencia más eficiente a los usuarios.

Contexto del Proyecto:

Actualmente en la ciudad de Concepción se está trabajando en una propuesta de transformación digital para el sistema de transporte público del Gran Concepción. Parte de la propuesta es presentar una plataforma donde se disponibilizará la información existente permitiendo la gestión de la información.

La institución en Chile encargada de generar los archivos GTFS es la SEREMI. Uno de los propósitos de esta plataforma es validar que el archivo generado cumpla con el estándar y que la información se pueda acoplar a nuestro sistema, manteniendo así un registro de la información declarada por las autoridades.

De igual manera, el sistema gestionará a sus usuarios de la ciudad de Concepción. Cada usuario tendrá acceso restringido según la autorización correspondiente, indicando qué usuarios tienen acceso a todos los registros de los archivos GTFS o simplemente ver la información referente a un recorrido.

Este proyecto está centrado en presentar una propuesta para la carga de los nuevos archivos GTFS, manteniendo las relaciones de la base de datos y diagnosticando errores antes de ingresar los registros para mantener la consistencia entre los registros actuales y próximos.

Propósito del Proyecto:

Este proyecto no solo busca abordar la gestión de registros en archivos GTFS, sino que también se enfoca en aspectos como la trazabilidad de los datos, la identificación de nuevos registros y la eliminación de registros obsoletos.

La aplicación de este componente no solo beneficiará a la ciudad de Concepción, sino que también tiene un alcance más amplio, permitiendo a otros usuarios adaptarlo a sus propios proyectos de gestión de datos de transporte público. Además, este proyecto contribuirá al enriquecimiento del conocimiento en el campo de la gestión de datos de movilidad urbana, ofreciendo una solución valiosa para las necesidades actuales en este ámbito.

Este proyecto se propone como una herramienta para optimizar la administración de registros en archivos GTFS, con un enfoque inicial en Concepción y la perspectiva de una aplicación más amplia en el ámbito de la movilidad urbana, brindando un mayor control y trazabilidad de los datos en este contexto.

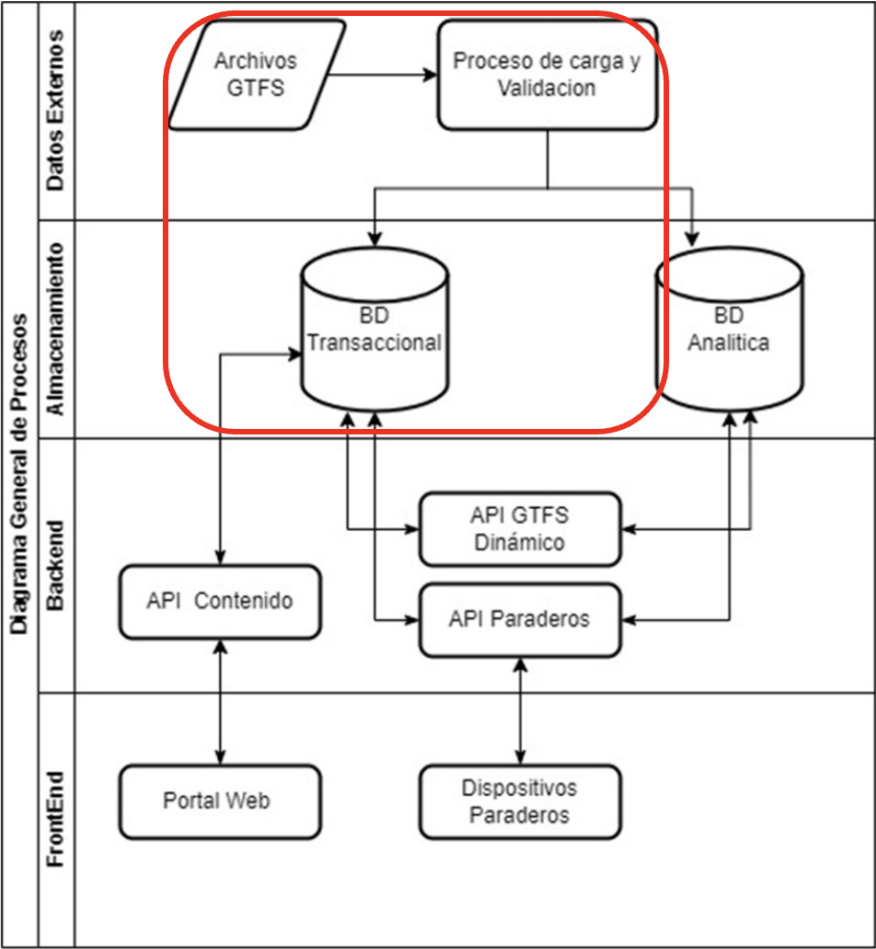


Figura 1: Área de enfoque del proyecto.



Capítulo 1

Problemática



1.1. Objetivo General

El objetivo general de este proyecto es diseñar y proponer una solución a la automatización para la gestión de registros en archivos GTFS, que permita la identificación automática de registros nuevos y obsoletos, así como el seguimiento de la trazabilidad de los datos.

Este sistema busca eliminar la dependencia de procesos manuales y proporcionar una solución que aumente la eficiencia en la gestión de datos de transporte público, con un enfoque inicial en la ciudad de Concepción y la capacidad de adaptarse a otros contextos y proyectos similares.

1.2. Objetivos Secundarios

- Implementación: Desarrollar un sistema de automatización para la gestión de registros en archivos GTFS.
- Documentación de los Endpoints: Preparar documentación que explique la API del sistema y su endpoint, facilitando su comprensión y uso por parte de los usuarios y desarrolladores.
- Trabajos Futuros: Proponer posibles mejoras y ampliaciones del sistema.

1.3. Descripción de la Problemática

La problemática principal radica en la necesidad de encontrar una propuesta que de solución a la automatización de la actualización de datos GTFS en el sistema de transporte público de Concepción.

1.4. Estándares y Estructura de los archivos GTFS

Un punto importante en la búsqueda de esta solución es la necesidad de implementar una validación de la estructura del archivo ZIP. Este paso es importante para asegurar que los usuarios ingresen archivos que cumplan con los requisitos especificados, garantizando así un procesamiento adecuado en la plataforma.

1.5. Softwares Existentes

La automatización de la actualización de datos GTFS se vuelve esencial para mantener informados a los usuarios sobre las rutas y horarios del sistema de transporte público. Durante la investigación de soluciones para esta problemática, evaluamos gestores GTFS existentes, como el Transit Data Manager de Moovit [Moovit \(2023\)](#), que se presentan como opciones para simplificar esta tarea.

Aunque estos gestores ofrecen eficiencia al acceder a datos GTFS, es importante considerar que su implementación implica la adquisición de licencias, lo que conlleva costos asociados y puede ser un factor limitante. Además, es relevante destacar que estos gestores son sistemas en sí y no permiten una integración fluida como componentes modulares en un sistema independiente, como el propuesto para la ciudad de Concepción. Esta falta de flexibilidad puede afectar la capacidad del sistema para adaptarse a las necesidades específicas de la ciudad y limitar su modularidad.

Las adaptaciones personalizadas y la configuración adicional son necesarias para asegurar que los datos recopilados sean relevantes y precisos para la ciudad.

Además, la dependencia de soluciones de terceros puede plantear preocupaciones sobre la autonomía y el control sobre la infraestructura y los datos críticos de transporte público. Esto puede ser un factor importante a considerar cuando se busca una solución que cumpla con los requisitos específicos de cada ciudad.

Por otro lado, existen editores de GTFS como GTFS Manager de Trillium [Transit \(2023\)](#), que están diseñados para crear y editar datos GTFS desde cero o modificar conjuntos de datos existentes. Estas herramientas permiten a las agencias de transporte y a los administradores de datos ajustar la información de transporte de acuerdo con sus necesidades locales. Sin embargo, a menudo se espera que los editores de GTFS sean operados manualmente y no suelen ofrecer capacidades avanzadas de automatización para la actualización periódica y en tiempo real de los datos.

Ninguna de estas categorías ofrece una solución modular o completa para la automatización de la actualización de datos GTFS.

1.6. Problemáticas técnicas

Para lograr una implementación de una solución personalizada, se deben superar tanto los desafíos técnicos como la adaptación de las herramientas disponibles a las necesidades específicas de la ciudad. Lo

principal es encontrar un equilibrio entre la distribución de datos y la capacidad de modificar esos datos para satisfacer las particularidades locales.

Una de las problemáticas clave que enfrentamos incluye la falta de un modelo de datos para los GTFS específico para Concepción. Esto requiere la creación de una estructura de datos que se adapte a las características locales y a las necesidades del sistema de transporte de la ciudad.

Otro aspecto crítico es la necesidad de identificar claramente las cargas que se ingresan a la base de datos. Esto implica la incorporación de registros con un campo específico que indique a qué carga pertenecen los datos ingresados, mejorando así la gestión y la supervisión de los recursos.

Una limitación técnica importante es que el archivo `shapes.txt` debería tener una clave primaria (PK), pero en la actualidad contiene datos repetidos. Esto presenta un desafío en términos de diseño de base de datos y gestión de la integridad de los datos.

Por último, garantizar la modularidad y la capacidad de integración para que la solución pueda interactuar de manera efectiva con otros componentes del sistema de transporte digitalizado.

1.7. Restricciones de Diseño

En este apartado, se exploran las restricciones de diseño que deben tenerse en cuenta para el proyecto. Entre estas restricciones se incluye la necesidad de que la aplicación sea web, que permitan interacciones mediante llamadas a través de endpoints. Esta especificación se traduce en una arquitectura que facilita la comunicación con otros sistemas y servicios, brindando una facilidad a los servicios que pueden aprovechar la funcionalidad a través de la API.

El requisito de utilizar un framework de Python, además de limitar las opciones, aporta ventajas como la coherencia en el desarrollo, la comunidad de apoyo y la velocidad de implementación gracias a la sintaxis clara y las bibliotecas disponibles. La orientación hacia endpoints web proporciona una flexibilidad adicional en la interconexión de sistemas, permitiendo una integración más eficiente con otras plataformas y servicios.

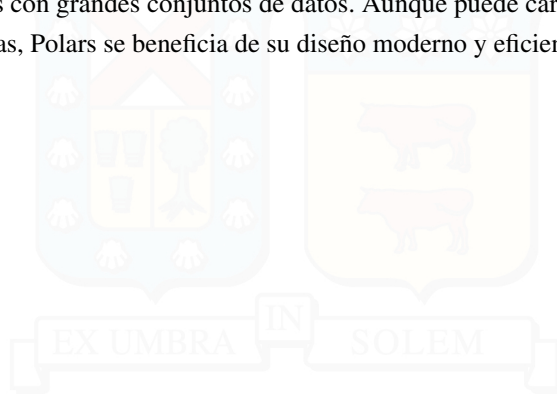
En cuanto al uso de PostgreSQL como base de datos, nos brinda una gestión robusta y confiable de los datos. Las restricciones establecidas no solo definen la infraestructura técnica del proyecto, sino que también orientan la aplicación hacia una arquitectura que favorece la interoperabilidad y la integración con otros componentes.

Fuera de estas restricciones, se plantea una alternativa que consiste en la implementación de Apache Airflow como orquestador de pipelines. Esta adición a la infraestructura podría proporcionar beneficios notables, como una visualización más detallada del estado de las cargas y una mayor capacidad de modularización. Con Airflow, se ganaría la capacidad de estructurar y monitorear las tareas de carga de manera más eficiente, facilitando así la identificación y resolución de problemas en el flujo de trabajo.

Sin embargo, es importante considerar las posibles consecuencias de esta alternativa. La implementación de Apache Airflow podría requerir tiempo y recursos adicionales, aumentando la complejidad del proyecto y añadiendo una dependencia adicional. La curva de aprendizaje y la configuración de Airflow también representan desafíos.

Se han contemplado medidas de diseño, entre ellas la elección entre las bibliotecas Pandas y Polars para el procesamiento de datos. Pandas, con su extensa documentación y una gran comunidad, aunque presenta una desventaja en términos de velocidad de procesamiento, especialmente con conjuntos de datos grandes debido a su implementación en Python.

Por otro lado, se ha considerado Polars, una biblioteca más reciente que promete un rendimiento más rápido, especialmente en operaciones con grandes conjuntos de datos. Aunque puede carecer de la misma cantidad de documentación que Pandas, Polars se beneficia de su diseño moderno y eficiente.



Capítulo 2

Marco Teórico

La presente sección tiene como objetivo proporcionar un sólido fundamento conceptual para nuestro proyecto. A lo largo de estas páginas, se exploran conceptos y antecedentes que sustentan la investigación y desarrollo del proyecto.

El marco teórico es esencial para comprender el contexto en el que se enmarca nuestro proyecto y para identificar las contribuciones que este aportará al campo.

2.1. Secretaría Regional Ministerial (SEREMI):

Las SEREMIs son divisiones del gobierno regional que se encargan de la administración y ejecución de políticas públicas a nivel regional en diversas áreas, como salud, educación, medio ambiente, transporte, entre otras. Cada región de Chile tiene su propia SEREMI en cada una de estas áreas, y su función es supervisar y coordinar la implementación de políticas y programas del gobierno central a nivel regional. Las SEREMIs desempeñan un papel importante en la descentralización y regionalización de la administración pública en Chile.

2.2. Python:

Python es un lenguaje de programación de alto nivel y de propósito general conocido por su simplicidad y legibilidad. Python es un lenguaje interpretado, lo que significa que el código se ejecuta línea por línea por un intérprete en lugar de ser compilado, lo que facilita la escritura y la depuración de programas.

Una de las características distintivas de Python es su amplia biblioteca estándar, que proporciona una gran variedad de módulos y funciones para tareas comunes, como manipulación de archivos, networking, desarrollo web, etc. Esto hace que Python sea altamente versátil y adecuado para una amplia gama de aplicaciones, desde desarrollo web y análisis de datos hasta automatización de tareas y desarrollo de aplicaciones científicas.

Además, Python es un lenguaje multiplataforma, lo que significa que puede ejecutarse en una variedad de sistemas operativos, como Windows, macOS y Linux.

2.2.1. Bibliotecas Relevantes:

```
anyio==3.7.1
asynpg==0.28.0
fastapi==0.101.0
pandas==1.5.3
psycpg2==2.9.7
SQLAlchemy==2.0.19
uvicorn==0.23.2
numpy
python-dateutil
pytz
blis
cymem
murmurhash
thinc==8.2.1
jinja2
starlette==0.27.0
pydantic==2.1.1
click
httpx==0.18.2
h11==0.14.0
websockets==10.0
asgiref==3.7.2
```



2.3. Hypertext Transfer Protocol (HTTP):

El Protocolo de Transferencia de Hipertexto (HTTP, por sus siglas en inglés) es el protocolo fundamental que rige la comunicación en la World Wide Web. Su función principal es permitir la transferencia de información, como páginas web, entre un servidor web y un cliente, que generalmente es un navegador web. HTTP opera bajo el modelo cliente-servidor, donde el cliente envía solicitudes al servidor para recuperar recursos web, como páginas HTML, imágenes, videos y otros tipos de datos.

HTTP se basa en una arquitectura simple de solicitud-respuesta. El cliente envía una solicitud HTTP al servidor, que incluye un método (como GET para recuperar datos) y una URL que apunta al recurso deseado.

Por otro lado tenemos el método POST que tiene la función de enviar datos para comparación o carga, es probable que estemos enviando un conjunto de datos más grande o complejo. El método POST permite

enviar datos en el cuerpo de la solicitud HTTP, facilitando la transmisión de datos extensos y estructurados. Si estamos realizando operaciones que modifican los datos en la base de datos.

El servidor procesa la solicitud y envía una respuesta HTTP al cliente, que incluye el estado de la operación (como éxito o error) y los datos solicitados. La comunicación entre el cliente y el servidor se realiza a través del protocolo TCP/IP, lo que garantiza una transferencia confiable de datos a través de Internet.

El TCP (Protocolo de Control de Transmisión) proporciona una comunicación confiable y orientada a la conexión entre dos dispositivos en una red, generalmente a través de Internet. Su principal función es garantizar que los datos enviados desde un extremo lleguen de manera íntegra y en el orden correcto al otro extremo, incluso en presencia de posibles pérdidas, duplicaciones o reordenamientos de paquetes de datos durante la transmisión.

2.4. General Transit Feed Specification (GTFS) Estático:

Las especificaciones generales del suministro de datos para el transporte público (GTFS), también conocidas como GTFS estáticas o transporte público estático, definen un formato común para los horarios de transporte público y la información geográfica relacionada. Los "Feeds" GTFS permiten que las empresas de transporte público publiquen sus datos y que los programadores escriban aplicaciones que consuman esos datos de manera interoperable, [GTFS \(2023\)](#).

Un Feed GTFS se compone de una serie de archivos de texto recopilados en un archivo ZIP. Cada archivo modela un aspecto específico de la información de transporte público.

Descripción breve de los archivos utilizados en el proyecto:

2.4.1. Agency.txt:

Define las empresas de transporte público que tienen un servicio representado en este conjunto de datos. Algunos ejemplos del contenido pueden verse en la tabla A.1.

2.4.2. Stops.txt:

Define las paradas en las que los vehículos recogen o dejan pasajeros. También indica las estaciones y las entradas de las estaciones. Algunos ejemplos del contenido pueden verse en la tabla A.6.

2.4.3. Routes.txt:

Rutas de transporte público. Una ruta es un grupo de viajes que se muestra a los pasajeros como un solo servicio. Algunos ejemplos del contenido pueden verse en la tabla A.3.

2.4.4. Trips.txt:

Viajes para cada ruta. Un viaje es una secuencia de dos o más paradas que ocurre durante un período específico. Algunos ejemplos del contenido pueden verse en la tabla A.7.

2.4.5. StopTimes.txt:

Proporciona las horas en las que un vehículo llega a una parada y sale de ella en cada viaje. Algunos ejemplos del contenido pueden verse en la tabla A.5.

2.4.6. Calendar.txt:

Indica las fechas de servicio especificadas mediante un horario semanal, con fechas de inicio y finalización. Algunos ejemplos del contenido pueden verse en la tabla A.2.

2.4.7. Shapes.txt:

Define las reglas para asignar las rutas de viaje de los vehículos, también conocido como alineamientos de rutas. Algunos ejemplos del contenido pueden verse en la tabla A.4.

Shapes describen la ruta que realiza un vehículo en un alineamiento de rutas y se definen en el archivo shapes.txt. Las shapes se asocian con viajes y se componen de una secuencia de puntos que el vehículo atraviesa en orden. No es necesario que las shapes intercepten la ubicación de las paradas de forma exacta, pero todas las paradas de un viaje deben encontrarse a una distancia cercana respecto de la forma de ese viaje, es decir, cerca de los segmentos rectos que conectan los puntos de la forma.

2.5. General Transit Feed Specification (GTFS) en Tiempo Real:

Proporcionar a los usuarios actualizaciones de los datos de transporte público en tiempo real mejora notablemente su experiencia en el servicio de transporte público que brindas. Brindar información actualizada sobre las horas actuales de llegada y salida permite que los usuarios planifiquen sus viajes sin problemas.

- Actualizaciones de viaje: Retrasos, cancelaciones, cambios de ruta.
- Alertas de servicio: Traslados de paradas, eventos imprevistos que afectan una estación, ruta o toda la red.
- Posiciones de vehículos: Información sobre los vehículos, incluida la ubicación y el nivel de embotellamiento.

2.6. Application Programming Interface (API) y Endpoints:

Una API es un conjunto de reglas y protocolos que permiten que diferentes aplicaciones se comuniquen y compartan datos entre sí de manera efectiva. Sirve como un intermediario que permite que dos aplicaciones se entiendan y trabajen juntas, incluso si están diseñadas para plataformas o lenguajes de programación diferentes. Las API definen cómo se deben solicitar y compartir datos, funciones o servicios específicos, lo que permite la integración y la interoperabilidad entre sistemas distintos.

Los endpoints son puntos de acceso específicos dentro de una API que representan recursos o funcionalidades particulares. Cada endpoint está asociado con una URL única y responde a solicitudes específicas que pueden ser de lectura, escritura, actualización o eliminación de datos. Por ejemplo, en una API de redes sociales, podría haber un endpoint para recuperar los últimos mensajes de un usuario y otro endpoint para publicar un nuevo mensaje. Los endpoints se utilizan para interactuar con recursos o ejecutar acciones específicas a través de la API.

2.7. PostgreSQL:

Sistema de gestión de bases de datos de código abierto que sigue el modelo de bases de datos relacionales. Este sistema permite la organización y el almacenamiento de datos en tablas estructuradas, donde cada tabla almacena información específica de manera ordenada. La flexibilidad de PostgreSQL radica en su capacidad para personalizar y extender estas tablas según las necesidades particulares de una aplicación o empresa.

Además, PostgreSQL cumple con los principios fundamentales de las bases de datos transaccionales al garantizar la Atomicidad, Consistencia, Aislamiento y Durabilidad (ACID) de las transacciones de datos. Esta característica es esencial en entornos donde la integridad y la confiabilidad de la información son cruciales.

La escalabilidad de PostgreSQL, tanto vertical como horizontal, lo hace adecuado para adaptarse a diferentes volúmenes de datos y niveles de carga de trabajo.

En el contexto de la seguridad de datos, PostgreSQL ofrece sólidos mecanismos de autenticación de usuarios y autorización basada en roles, proporcionando un control detallado sobre quién puede acceder y modificar los datos. Esto es de suma importancia en aplicaciones y entornos empresariales donde la privacidad y la confidencialidad son esenciales.

2.8. SQLAlchemy (ORM):

Es un Kit de herramientas Python SQL y el Object Relational Mapper (ORM) que brinda a los desarrolladores de aplicaciones la flexibilidad de SQL. Proporciona un conjunto completo de patrones de persistencia conocidos a nivel empresarial, diseñados para un acceso eficiente y de alto rendimiento a bases de datos.

SQLAlchemy es una biblioteca de Python que facilita la interacción con bases de datos relacionales. Funciona permitiendo a los desarrolladores definir modelos de datos en Python, que representan tablas y relaciones en la base de datos. Luego, a través de sesiones, se establece una conexión a la base de datos, y las operaciones como consultas y actualizaciones se escriben en Python, lo que hace que la interacción sea más legible y segura.

También se encarga de manejar transacciones de base de datos, garantizando que las operaciones se realicen de manera coherente y segura. En resumen, actúa como un intermediario eficiente entre una aplicación Python y una base de datos relacional, simplificando el desarrollo y la gestión de datos.

2.9. Pandas:

Pandas, una biblioteca de Python ampliamente utilizada en el ámbito del análisis de datos, también puede ser una herramienta valiosa para tareas de inserción y actualización de registros en una base de datos. En primer lugar, Pandas es esencial para la preparación de datos antes de la inserción. Ofrece funciones poderosas para limpiar y transformar datos, lo que facilita la corrección de errores, la eliminación de duplicados y la garantía de que los datos sean precisos y coherentes antes de ingresarlos en la base de datos.

Los DataFrames de Pandas son ideales para representar los datos que se insertarán o actualizarán en la base de datos. Los registros nuevos o actualizados se pueden organizar en DataFrames, lo que permite una revisión y verificación eficiente antes de la interacción con la base de datos. Además, Pandas ofrece métodos para conectarse a bases de datos SQL, como MySQL o PostgreSQL, a través de bibliotecas como SQLAlchemy, y para realizar operaciones de inserción o actualización utilizando los datos en el DataFrame.

También puede utilizarse para registrar y controlar las operaciones realizadas en la base de datos, lo que facilita el seguimiento de los cambios y la generación de registros de auditoría para mantener un registro de las modificaciones en los registros.

2.10. Fast API:

FastAPI es un marco moderno de desarrollo web en Python que se ha ganado rápidamente la atención de la comunidad debido a su velocidad, simplicidad y rendimiento. Diseñado para facilitar la creación de APIs de manera rápida y eficiente.

Una de las características destacadas de FastAPI es su enfoque en la creación de APIs con documentación automática. Gracias a la integración directa con Swagger UI y ReDoc, la documentación se genera automáticamente a partir de las anotaciones de tipo en el código, lo que facilita la comprensión y prueba de la API.



Figura 2.1: Fast API.

2.11. Django:

Django es un marco de desarrollo web de código abierto en Python que sigue el patrón Modelo-Vista-Plantilla (MVT). ofrece un conjunto completo de herramientas para el desarrollo eficiente. Incluye un ORM para interactuar con bases de datos, un administrador automático, sistema de plantillas, enrutamiento y vistas, y características de seguridad integradas. Django facilita el desarrollo rápido con recarga automática del servidor y soporte para tareas comunes como autenticación y gestión de sesiones. Su comunidad activa y extensa documentación hacen de Django una opción popular para proyectos web.

2.12. Docker:

[Docker \(2023\)](#) es una plataforma de código abierto que simplifica el desarrollo y despliegue de aplicaciones a través de contenedores. Estos contenedores encapsulan la aplicación y sus dependencias, asegurando portabilidad y consistencia en cualquier entorno compatible con Docker. Las imágenes de Docker, instantáneas inmutables que contienen todo lo necesario para ejecutar una aplicación, pueden ser compartidas y versionadas mediante Dockerfiles, scripts que definen los pasos de construcción.

2.13. Airflow:

Apache [Airflow \(2023\)](#) es una plataforma de código abierto diseñada para simplificar la automatización y gestión de flujos de trabajo complejos. Basada en el concepto de DAGs (Grafos Acíclicos Dirigidos), facilita la representación visual y la ejecución eficiente de procesos secuenciales.

Con capacidades de planificación y programación, Airflow permite la automatización de tareas en intervalos específicos o en respuesta a eventos externos. Su extensibilidad es destacable, ya que se integra fácilmente con una variedad de sistemas y servicios a través de conectores, abriendo posibilidades para interactuar con bases de datos, almacenamiento en la nube y otros servicios.

La plataforma proporciona un panel de control web que ofrece una visión detallada del estado y del historial de ejecución de las tareas, facilitando la monitorización y resolución de problemas. Además, la tolerancia a fallos y la gestión automática de reintentos mejoran la robustez de los flujos de trabajo.



Figura 2.2: Django.

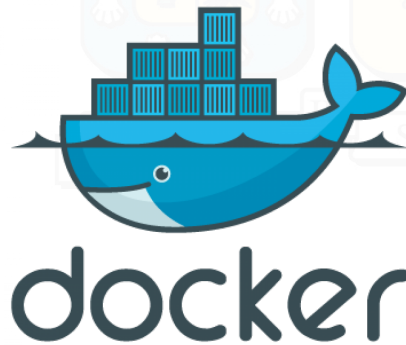


Figura 2.3: Docker.

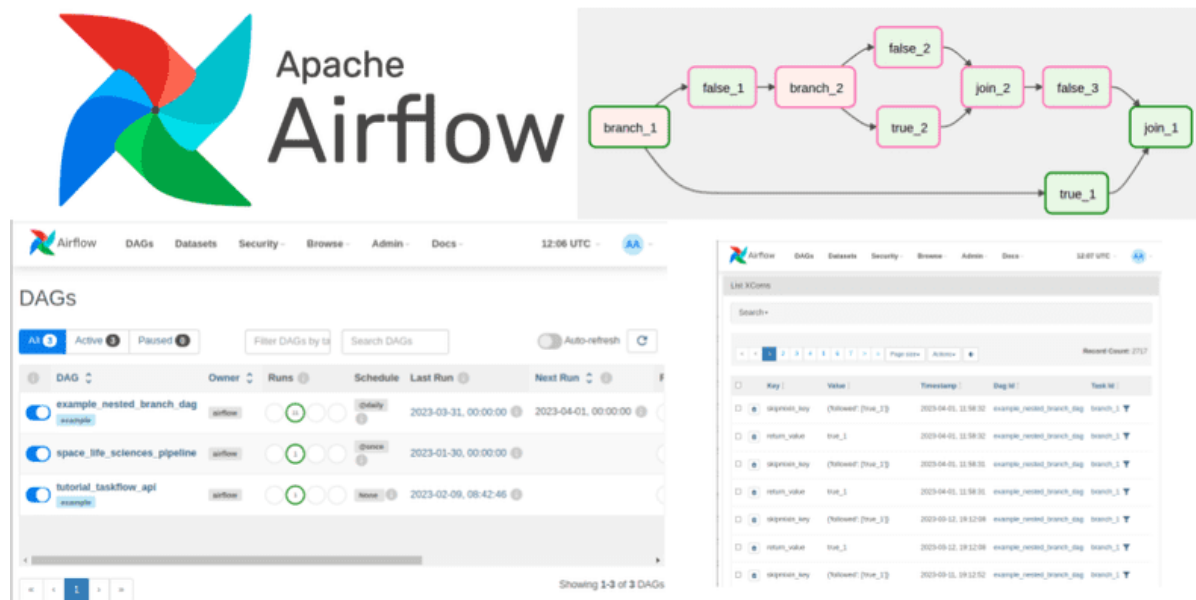


Figura 2.4: Airflow.

Capítulo 3

Solución

Este capítulo se centra en la justificación técnica de la propuesta para la automatización de datos GTFS. En el diseño de este proyecto, se consideraron diversas alternativas, y la elección de ciertos componentes y enfoques se basa en criterios específicos que responden a las necesidades y desafíos identificados en los capítulos anteriores.

La decisión de dividir el proyecto en dos componentes principales, uno dedicado a la creación de la base de datos GTFS y otro a la carga de registros, se fundamenta en el principio de modularidad. Al asignar funciones específicas a cada componente, se logra un sistema más flexible que permite realizar cambios o mejoras en una tarea sin afectar otras áreas. Esta modularidad facilita futuras actualizaciones y escalabilidad del sistema, ya que las modificaciones se pueden realizar de manera aislada.

3.1. Primer Componente: Creación de la Base de Datos para registros GTFS.

En la implementación de este componente, se optó por el framework Fast API y SQLAlchemy como ORM, con estas herramientas se realiza la conexión a la base de Datos y la creación de los modelos. [Tiangolo \(2023\)](#). Estas elecciones se fundamentan en su capacidad para proporcionar una estructura simple y una conexión sencilla a la base de datos. No obstante, es fundamental destacar que se presentó la propuesta de emplear DjangoRestFramework. [Tech \(2023\)](#)

La elección de no seleccionar DjangoRestFramework no implica que sea una opción desfavorable. Se eligió la solución mencionada debido a que FastAPI se centra en proporcionar solo lo esencial para la creación de APIs, evitando características adicionales que podrían no ser necesarias para nuestro proyecto en particular.

Este componente juega un papel importante en la creación y gestión de la base de datos GTFS. El modelo de base de datos seleccionado fue recomendado por el equipo de Google Transit [Inc. \(2023\)](#), a pesar de que los archivos GTFS no estaban inicialmente diseñados como un modelo entidad-relación. Sin embargo, para adaptarlo a las necesidades locales de Concepción, fue necesario realizar una variación en el modelo.

Listing 3.1: Conexión a la base de datos usando SQLAlchemy

```
1
2 engine =
3 create_engine("postgresql://postgres:5314806Jair@localhost:5432/fastApi_gtfs")
4
5 SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
6
7 Base = declarative_base()
```

En el marco de este diseño, se optó por la creación de dos tablas específicas para abordar necesidades particulares del proyecto

La tabla Carga-gtfs desempeña un papel importante en la trazabilidad de los datos. Aquí se almacena información como detalles sobre el usuario responsable de la carga y características del archivo ZIP utilizado. La asignación de la clave primaria 'id-carga-gtfs' como clave foránea en otras tablas cumple una función de indicar qué registros fueron ingresados durante una carga específica [Romero Munizaga \(2021\)](#). Este enfoque proporciona una estructura organizada que facilita la comprensión de los efectos a lo largo del tiempo.

Listing 3.2: Definición de la clase CargaGtfs

```
1
2 class CargaGtfs(Base):
3     __tablename__ = 'carga_gtfs'
4
5     id_carga_gtfs = Column(Integer, primary_key=True, autoincrement=True)
6     id_usuario = Column(String(255))
7     zip = Column(String(255))
8     fecha = Column(DateTime, server_default=func.now())
9     estado = Column(String(50))
10    mensaje = Column(String(255))
11
12    def __repr__(self):
13        return f"{self.id_usuario}, {self.zip}, {self.fecha}, {self.estado},\n        {self.mensaje}"
```

Para el desafío de la repetición de datos en el archivo GTFS 'shapes' en Concepción, se creó una nueva tabla que almacena los datos de 'Shapes' sin duplicados, estableciendo relaciones con la tabla 'trips'. Este enfoque garantiza la integridad de los datos al tiempo que aborda la duplicidad presente en el archivo GTFS de Shapes, mejorando así la coherencia del sistema en su conjunto.

**Listing 3.3:** Clase ShapeReference

```
1
2 class ShapeReference(Base):
3     __tablename__ = 'shape_reference'
4
5     id_carga_gtfs = Column(Integer, ForeignKey('carga_gtfs.id_carga_gtfs'),
6                             nullable=False)
7     shape_id = Column(String(50), primary_key=True)
8
9     carga_gtfs = relationship("CargaGtfs", back_populates="shape_references")
10
11     def __repr__(self):
12         return f"Shape ID: {self.shape_id}"
```

Listing 3.4: Clase Shapes

```
1
2 class Shapes(Base):
3     __tablename__ = 'shapes'
4
5     id_carga_gtfs = Column(Integer, ForeignKey('carga_gtfs.id_carga_gtfs'),
6                             nullable=False)
7     shape_id = Column(String(50), ForeignKey('shape_reference.shape_id'),
8                             primary_key=True)
9     shape_pt_lat = Column(String)
10    shape_pt_lon = Column(String)
11    shape_pt_sequence = Column(String, primary_key=True)
12
13    carga_gtfs = relationship("CargaGtfs", back_populates="shapes")
14    shape_reference = relationship("ShapeReference", back_populates="shapes")
15
16    def __repr__(self):
17        return f"Shape ID: {self.shape_id}, Lat: {self.shape_pt_lat}, Lon: {self.shape_pt_lon}"
```

En la tabla 'Stops' debido a la variabilidad encontrada en el campo 'id-stops', que presentaba partes de una fecha y un código sin un formato específico. Se propone abordar esta variabilidad con la creación de un 'id-stops-interno' que sigue una lógica incremental al ingresar nuevos registros en la base de datos. Esta elección se fundamenta en la necesidad de evitar conflictos futuros y garantizar la coherencia en la identificación de paradas almacenadas en la tabla.



Listing 3.5: Clase Stop

```
1
2 class Stop(Base):
3     __tablename__ = 'stop'
4
5     id_carga_gtfs = Column(Integer, ForeignKey('carga_gtfs.id_carga_gtfs'),
6         nullable=False)
7     stop_id_interno = Column(Integer)
8     stop_id = Column(String(50), primary_key=True)
9     stop_code = Column(String(50))
10    stop_name = Column(String(255))
11    stop_desc = Column(String(255))
12    stop_lat = Column(String(50))
13    stop_lon = Column(String(50))
14    zone_id = Column(String(50))
15    stop_url = Column(String(255))
16    location_type = Column(String(50))
17    parent_station = Column(String(50))
18    wheelchair_boarding = Column(String(50))
19
20    carga_gtfs = relationship("CargaGtfs", back_populates="stops")
21
22    def __repr__(self):
23        return self.stop_name
```

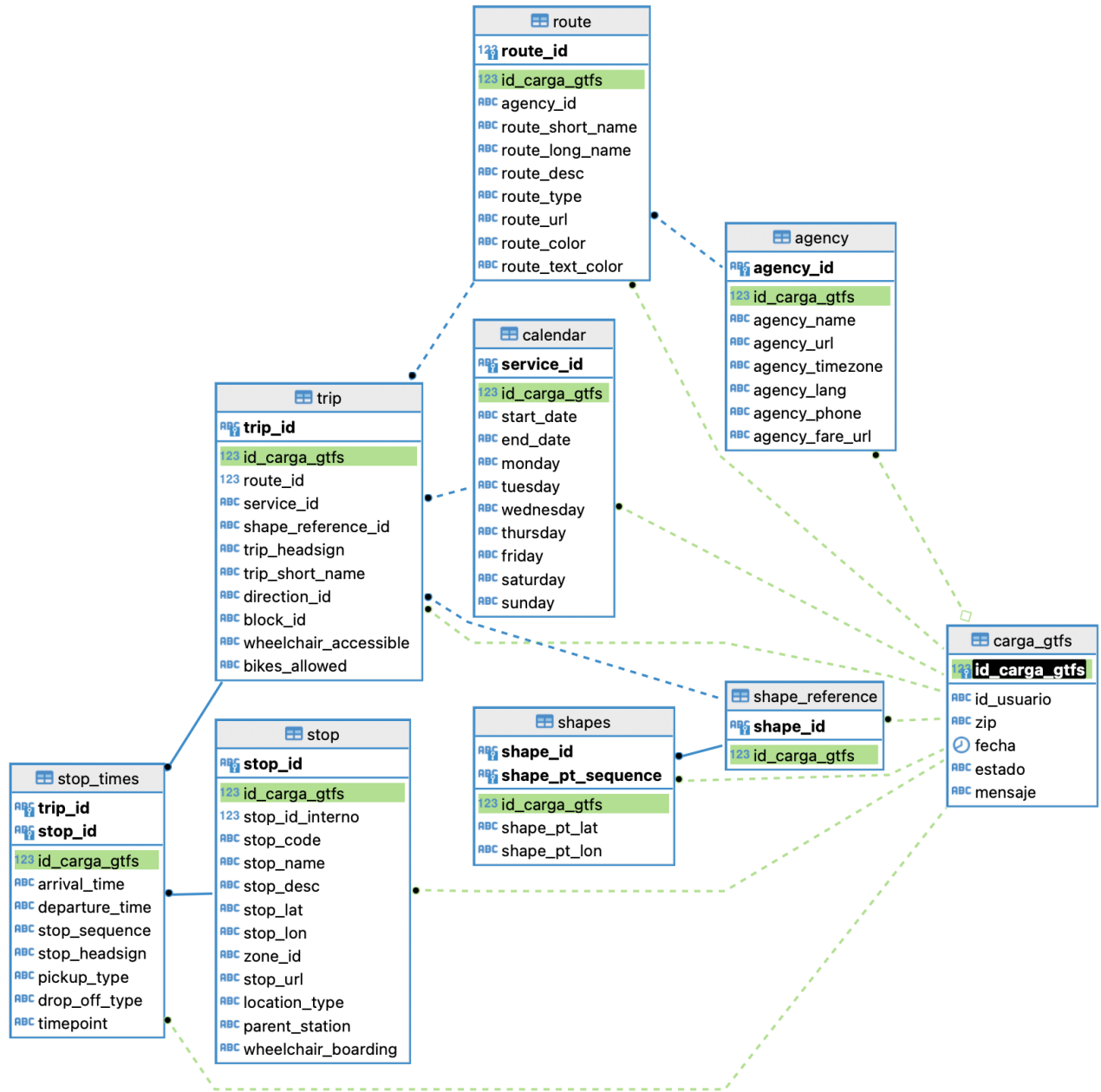


Figura 3.1: Base de Datos GTFS.

3.2. Segundo Servicio: Carga Registros GTFS.

El núcleo del proyecto se vincula directamente con la base de datos GTFS. Cada carga de datos se asocia con el nombre de usuario, el nombre del archivo ZIP, la hora de carga y detalles específicos del ZIP que se actualizan para indicar en qué parte del proceso se encuentra.

La información extraída del ZIP se almacena en la base de datos GTFS, y se genera un registro en la capa de trazabilidad para mantener un historial claro de las contribuciones de los usuarios.

Surge una pregunta clave sobre qué ocurre cuando el proceso falla por cualquier motivo, como un error en la inserción de datos en la base de datos. En caso de fallo, se implementa un mecanismo de restauración que revierte los cambios y restablece la base de datos a su estado inicial. Este enfoque podría garantizar integridad y coherencia de los datos, evitando posibles inconsistencias en caso de errores durante la carga.

Se implementará el uso de `DeferredReflection` para establecer la conexión con la base de datos. Esta elección resulta especialmente beneficiosa al interactuar con bases de datos preexistentes, ya que nos proporciona la capacidad de realizar la reflexión automática de las tablas cuando sea necesario, lo cual puede ser beneficioso porque reduce el tiempo de inicio de la aplicación, ya que no es necesario reflexionar todas las tablas de la base de datos inmediatamente al cargar el modelo.

Listing 3.6: Configuración de la base de datos

```
1
2 DATABASE_URL = "postgresql://postgres:5314806Jair@localhost:5432/fastApi_gtfs"
3 engine = create_engine(DATABASE_URL)
4
5 # Creamos base para los modelos utilizando DeferredReflection
6 Base = declarative_base(cls=DeferredReflection)
7 Base.metadata.bind = engine
```

Listing 3.7: Definición de la clase ‘CargaGtfs’

```
1
2 class CargaGtfs(Base):
3     __tablename__ = 'carga_gtfs'
4
5     id_carga_gtfs = Column(Integer, primary_key=True, autoincrement=True)
6     id_usuario = Column(String(255))
7     zip = Column(String(255))
8     fecha = Column(DateTime, server_default=func.now())
9     estado = Column(String(50))
10    mensaje = Column(String(255))
11
12    def __repr__(self):
13        return f"{self.id_usuario}, {self.zip}, {self.fecha}, {self.estado},
        {self.mensaje}"
```

3.2.1. Endpoint:

Este código define un endpoint utilizando FastAPI, un marco de trabajo moderno para construir APIs. El endpoint en se encuentra en la ruta /cargaz responde a solicitudes POST. Vemos el endpoint en la figura

Para que este endpoint pueda ejecutarse de manera adecuada, se requiere que exista una sesión de base de datos, la cual se obtiene mediante la función 'get-db' como dependencia. FastAPI utiliza el concepto de dependencias para gestionar la inyección de dependencias de manera sencilla y segura. En este caso, la dependencia 'get-db' proporciona una sesión de base de datos a la función carga.

El endpoint realiza una serie de operaciones, incluyendo la identificación y extracción de archivos ZIP, la inspección y transformación de archivos de texto, y la carga de datos resultante en la base de datos. Además, se implementa manejo de errores para abordar posibles problemas durante el proceso de carga.

3.2.2. Enfoque de comparacion de registros:

Se optó por utilizar la biblioteca Pandas para la implementación del enfoque de comparación de registros, aunque la elección no estuvo exenta de consideraciones y dificultades.

Un punto a tener en cuenta fue la curva de aprendizaje asociada con cada opción. La familiaridad con las herramientas disponibles y la rapidez con la que se podía integrar fueron aspectos importantes para el desarrollo del proyecto.

El enfoque propuesto para la comparación detallada de valores por fila en lugar de basarse únicamente en el identificador del registro se fundamenta en considerar que, aunque el identificador, apunta siempre al mismo registro, el contenido del registro puede cambiar con el tiempo. Por ejemplo, la coordenada de un paradero o el nombre de una calle pueden modificarse en versiones posteriores de los datos.

La decisión de realizar la comparación campo por campo y fila por fila se basa en la necesidad de capturar actualizaciones específicas en los datos, que podrían no ser evidentes al simplemente verificar el identificador del registro. Al realizar una comparación detallada, la solución propuesta proporciona una mayor profundidad en la evaluación de los cambios entre los dos conjuntos de datos. Esto permite identificar no solo los registros que son diferentes en términos de identificador, sino también aquellos que han experimentado cambios en sus valores individuales a lo largo del tiempo.

Listing 3.8: Comparacion de versiones en routes.txt

```

1 En la version 2023-11-07, el campo que cambio con respecto a otras versiones
  en el route_id 588 es:
2 Campo: route_long_name, Valor en 2023-07: San Remo 24F - Candelaria - San
  Remo 24M - Los Fresnos, Valor en 2023-11-07: San Remo 24F - Juegos
  Panamericanos - San Remo 24M - Juegos Panamericanos
3 Campo: route_long_name, Valor en 2023-09: San Remo 24F - Candelaria - San
  Remo 24M - Los Fresnos, Valor en 2023-11-07: San Remo 24F - Juegos
  Panamericanos - San Remo 24M - Juegos Panamericanos
4 -----
5 En la version 2023-11-07, el campo que cambio con respecto a otras versiones
  en el route_id 627 es:
6 Campo: route_short_name, Valor en 2023-07: 72WJ, Valor en 2023-11-07: 72JW
7 Campo: route_long_name, Valor en 2023-07: Pedro de Valdivia 72J - Lonco -
  Pedro de Valdivia 72W - Nonguen, Valor en 2023-11-07: Pedro de Valdivia
  72W - Lonco - Pedro de Valdivia 72J - Nonguen
8 Campo: route_short_name, Valor en 2023-09: 72WJ, Valor en 2023-11-07: 72JW
9 Campo: route_long_name, Valor en 2023-09: Pedro de Valdivia 72J - Lonco -
  Pedro de Valdivia 72W - Nonguen, Valor en 2023-11-07: Pedro de Valdivia
  72W - Lonco - Pedro de Valdivia 72J - Nonguen
10 -----
11 En la version 2023-11-07, el campo que cambio con respecto a otras versiones
  en el route_id 610 es:
12 Campo: route_short_name, Valor en 2023-07: 60XK, Valor en 2023-11-07: 60XL
13 Campo: route_long_name, Valor en 2023-07: Buses Tucapel 60K - Sta. Sabina -
  Buses Tucapel 60X - Hualpencillo, Valor en 2023-11-07: Buses Tucapel 60L
  - Sta. Sabina - Buses Tucapel 60X - Hualpencillo
14 Campo: route_short_name, Valor en 2023-09: 60XK, Valor en 2023-11-07: 60XL
15 Campo: route_long_name, Valor en 2023-09: Buses Tucapel 60K - Sta. Sabina -
  Buses Tucapel 60X - Hualpencillo, Valor en 2023-11-07: Buses Tucapel 60L
  - Sta. Sabina - Buses Tucapel 60X - Hualpencillo

```

Listing 3.9: Comparacion de versiones en stop.txt

```
1 En la version 2023-11-07, el campo que cambio con respecto a otras versiones
  en el ID 18-Jan es:
2 Campo: stop_lat, Valor en 2022-12: -36.61141, Valor en 2023-11-07:
  -36.611360704
3 Campo: stop_lon, Valor en 2022-12: -72.95565, Valor en 2023-11-07:
  -72.955620124
4 -----
5 En la version 2023-11-07, el campo que cambio con respecto a otras versiones
  en el ID 40960 es:
6 Campo: stop_lat, Valor en 2022-12: -36.92105601, Valor en 2023-11-07:
  -36.921426764
7 Campo: stop_lon, Valor en 2022-12: -73.0248702, Valor en 2023-11-07:
  -73.024736869
8 -----
9 En la versi n 2023-11-07, el campo que cambio con respecto a otras versiones
  en el ID 12-3oes:
10 Campo: stop_name, Valor en 2023-07: Freire esq Maipu, Valor en 2023-11-07:
  Freire esq. Maipu
11 Campo: stop_name, Valor en 2023-09: Freire esq Maipu, Valor en 2023-11-07:
  Freire esq. Maipu
12 -----
13 En la version 2023-11-07, el campo que cambio con respecto a otras versiones
  en el ID 90010 es:
14 Campo: stop_name, Valor en 2023-07: Los Robles esq Los Fresnos Poniente,
  Valor en 2023-11-07: Los Nogales esq. Las Pataguas
15 Campo: stop_lat, Valor en 2023-07: -36.84726772, Valor en 2023-11-07:
  -36.845843586
16 Campo: stop_lon, Valor en 2023-07: -73.10232218, Valor en 2023-11-07:
  -73.103425577
```

3.2.3. Actualización del estado del ZIP:

En el contexto de la implementación para la carga de datos desde archivos ZIP, cada registro correspondiente a los ZIPs ingresados en la base de datos posee dos campos: `Estado` y `Comentario`. Estos campos indican en que parte del proceso de carga nos encontramos, indicando si el proceso ha concluido con éxito o si se ha producido algún error.

Listing 3.10: Actualizar estado de la Carga en la Base de Datos

```
1
2 def actualizar_estado(id_carga_gtfs: int, estado: str, mensaje: str, db:
  Session):
3     zip = db.query(CargaGtfs).filter(CargaGtfs.id_carga_gtfs ==
      id_carga_gtfs).first()
4     zip.estado = estado
5     zip.mensaje = mensaje
6     db.commit()
```

3.2.4. Procesos de la Carga:

3.2.4.1. Identificación del último archivo ZIP ingresado al sistema:

Al realizar la identificación, se lleva a cabo una consulta en la base de datos que permite construir una instancia representativa de este nuevo registro.

Posteriormente, se procede a ubicar y reconocer el archivo ZIP en la ruta específica dentro del directorio de almacenamiento asignado. El procedimiento asegura la sincronización entre la información registrada en la base de datos y la ubicación física del archivo en el sistema de archivos.

Listing 3.11: Función para identificar el último ZIP en la base de datos.

```
1
2 def identificar_zip(db: Session):
3     zip = db.query(CargaGtfs).order_by(desc('fecha')).first()
4     return zip
```

3.2.4.2. Extraer ZIP:

Una vez identificado exitosamente el archivo ZIP, se extraen los archivos que contiene. Esta operación se lleva a cabo en una ruta predefinida, la cual fue especificada para garantizar un manejo estructurado de los datos extraídos.

Listing 3.12: Función para extraer archivos de un ZIP y actualizar el estado en la base de datos.

```
1
2 def extraer_zip(zip, zip_ruta, db):
3     with zipfile.ZipFile(zip_ruta, 'r') as archivo_zip:
4         archivo_zip.extractall('txts/')
5     actualizar_estado(id_carga_gtfs=zip.id_carga_gtfs, estado="Procesando",
        mensaje="Extrayendo archivos del ltimo ZIP ingresado.", db=db)
```

3.2.4.3. Inspeccionar archivos TXT:

En el proceso siguiente, se crea una instancia de una lista que identifica los archivos con extensión ".txt", los cuales son esenciales para la carga adecuada del sistema. Al examinar esta lista, si el número de archivos faltantes es mayor a cero, se actualiza el estado de la carga y proporcionamos información detallada sobre cuáles archivos específicos no se encontraron. En esta parte del proceso es donde se brinda la libertad de modificar los archivos esperados, dependiendo del modelo de datos que otras ciudades puedan tener, así ajustándose a sus necesidades.

Si, por otro lado, no hay archivos faltantes, en la siguiente etapa se inspecciona en busca de archivos sobrantes. En esta fase, se identifica cualquier archivo adicional que no esté incluido en la lista de archivos esenciales. Estos archivos sobrantes, que no contribuyen al proceso de carga, son eliminados de la ubicación en la que se extrajo el contenido del ZIP.

Listing 3.13: Función para inspeccionar archivos TXT y actualizar el estado en la base de datos.

```

1
2 def inspeccionar_txts(zip, almacen_txts:str, zip_ruta:str, db: Session):
3     archivos_en_txt = os.listdir(almacen_txts)
4     archivos_esperados = ['agency.txt', 'calendar.txt', 'routes.txt',
5         'stop_times.txt', 'stops.txt', 'trips.txt', 'shapes.txt']
6     archivos_faltantes = [archivo for archivo in archivos_esperados if archivo
7         not in archivos_en_txt]
8
9     if len(archivos_faltantes) > 0:
10         for archivo in archivos_en_txt:
11             archivo_ruta = os.path.join(almacen_txts, archivo)
12             os.remove(archivo_ruta)
13
14             actualizar_estado(id_carga_gtfs=zip.id_carga_gtfs, estado="Error",
15                 mensaje="Archivos Faltantes, por favor ingrese los siguientes
16                 archivos: "+", ".join(archivos_faltantes), db=db)
17             remove(zip_ruta)
18             raise ValueError("Archivos Faltantes, por favor ingrese los siguientes
19                 archivos: "+", ".join(archivos_faltantes))
20
21     for archivo in archivos_en_txt:
22         if archivo not in archivos_esperados:
23             os.remove(almacen_txts + archivo)
24
25     actualizar_estado(id_carga_gtfs=zip.id_carga_gtfs, estado="Procesando",
26         mensaje="Total de archivos esperados, siguiendo proceso...", db=db)

```

3.2.4.4. Transformar los TXT en dataframes:

En la siguiente fase, se instancia un DataFrame para cada archivo con extensión ".txt" utilizando la biblioteca Pandas. En caso de que surja algún error durante la lectura, el sistema se detendrá y se informará detalladamente sobre el error ocurrido.

Una vez se crean los DataFrames con éxito, se genera una lista que contiene las columnas esenciales para la carga adecuada de los registros. Si la cantidad de columnas faltantes es mayor a cero, se interrumpe la carga, se actualiza el estado e se indica cuáles son las columnas faltantes, especificando el DataFrame en cuestión. En esta parte del proceso es donde se brinda la libertad de modificar las columnas que otros archivos de diferentes ciudades puedan tener, así ajustándose a sus necesidades.

En el caso de tener columnas sobrantes, estas son eliminadas del DataFrame. En caso de éxito, se actualiza el estado, indicando que el DataFrame en particular cumple con las columnas necesarias para la carga.

A continuación, se agrega una nueva columna a la carga que almacena el ID del ZIP al que pertenece

cada registro del DataFrame. Esto se realiza para que, cuando los datos estén en la base de datos, podamos realizar consultas e identificar a qué carga pertenece cada registro.

Esto es un proceso por cada ".txt" pero por practicidad mostramos un ejemplo.

Listing 3.14: Función para transformar el archivo 'agency.txt' a un DataFrame y verificar las columnas necesarias.

```

1
2 def transformar_txts(zip, db: Session):
3     id_carga_gtfs = zip.id_carga_gtfs
4
5     try:
6         df_agency = pd.read_csv('txts/agency.txt', dtype=str)
7         columnas_necesarias_agency = ['agency_id', 'agency_name',
8                                       'agency_url', 'agency_timezone', 'agency_lang', 'agency_phone',
9                                       'agency_fare_url']
10    except Exception as e:
11        actualizar_estado(id_carga_gtfs=zip.id_carga_gtfs, estado="Error",
12                           mensaje="Se presentaron problemas al momento de transformar el
13                           archivo agency.txt a dataframe", db=db)
14
15    df_agency = verificar_columnas(df_agency, columnas_necesarias_agency,
16                                   'agency.txt', zip, db)
17    df_agency['id_carga_gtfs'] = id_carga_gtfs

```

Listing 3.15: Función para verificar las columnas de un DataFrame y filtrarlo según las columnas necesarias.

```

1
2 def verificar_columnas(dataframe, columnas_necesarias, archivo, zip, db):
3     columnas_sobrantes = [columna for columna in dataframe.columns if columna
4                           not in columnas_necesarias]
5     columnas_faltantes = [columna for columna in columnas_necesarias if
6                           columna not in dataframe.columns]
7
8     dataframe_filtrado = dataframe.drop(columnas_sobrantes, axis=1)
9
10    if len(columnas_faltantes) > 0:
11        actualizar_estado(id_carga_gtfs=zip.id_carga_gtfs, estado="Error",
12                           mensaje=f"Columnas faltantes en archivo {archivo}, por favor
13                           asegúrese de que el archivo contenga las siguientes columnas: {'',
14                           '.join(columnas_faltantes)}", db=db)
15        raise ValueError(f"Columnas faltantes en archivo {archivo}, por favor
16                           asegúrese de que el archivo contenga las siguientes columnas: {'',
17                           '.join(columnas_faltantes)}")
18    else:
19        actualizar_estado(id_carga_gtfs=zip.id_carga_gtfs,
20                           estado="Procesando", mensaje=f"Archivo '{archivo}' leido y

```

```

    comprobado que contenga todas las columnas necesarias, siguiendo
    proceso...", db=db)
13     return dataframe_filtrado

```

En el contexto específico de la tabla 'Stops', se implementa una columna adicional, denominada 'Stop-id-interno', esta se introduce para abordar el problema previo de falta de estandarización en el ID original.

Esto se logra asignando a cada fila en la tabla 'Stops' un valor único y secuencial en la columna 'stop-id-interno', comenzando desde uno y aumentando en incrementos de uno.

Listing 3.16: Transformación del archivo stops.txt a un DataFrame y verificación de columnas.

```

1
2 def transformar_txts(zip, db: Session):
3     id_carga_gtfs = zip.id_carga_gtfs
4
5     try:
6         df_stops = pd.read_csv('txts/stops.txt', dtype=str)
7         columnas_necesarias_stops =
8             ['stop_id', 'stop_code', 'stop_name', 'stop_desc',
9              'stop_lat', 'stop_lon', 'zone_id', 'stop_url', 'location_type',
10              'parent_station', 'wheelchair_boarding']
11     except Exception as e:
12         actualizar_estado(id_carga_gtfs=zip.id_carga_gtfs, estado="Error",
13                             mensaje="Se presentaron problemas al momento de transformar el
14                             archivo stops.txt a dataframe", db=db)
15
16     df_stops = verificar_columnas(df_stops, columnas_necesarias_stops,
17                                     'stops.txt', zip, db)
18     df_stops['id_carga_gtfs'] = id_carga_gtfs
19     df_stops['stop_id_interno'] = range(1, len(df_stops) + 1)

```

Se toma la decisión de crear un nuevo DataFrame a partir del archivo 'shapes.tx' con el propósito de agregar datos a la tabla correspondiente llamada 'shapes-reference'. Se lleva a cabo una selección del identificador (id) del archivo de texto original y se procede a eliminar cualquier duplicado presente.

Esta elección se basa en la necesidad de abordar la presencia de claves primarias duplicadas en la tabla 'shapes'. La estrategia implica la creación de una tabla intermedia, 'shapes-reference', que carece de duplicados en su llave primaria.

Listing 3.17: Transformación del archivo shapes.txt a un DataFrame y generación de shapes_reference.

```

1
2 def transformar_txts(zip, db: Session):
3     id_carga_gtfs = zip.id_carga_gtfs
4
5     try:
6         df_shapes = pd.read_csv('txts/shapes.txt', dtype=str)
7         columnas_necesarias_shapes =
8             ['shape_id', 'shape_pt_sequence', 'shape_pt_lat', 'shape_pt_lon']
9     except Exception as e:
10         actualizar_estado(id_carga_gtfs=zip.id_carga_gtfs, estado="Error",
11                             mensaje="Se presentaron problemas al momento de transformar el
12                                 archivo shapes.txt a dataframe", db=db)
13
14     df_shapes = verificar_columnas(df_shapes, columnas_necesarias_shapes,
15                                     'shapes.txt', zip, db)
16     df_shapes['id_carga_gtfs'] = id_carga_gtfs
17
18     # Se genera el dataframe correspondiente a la tabla shapes_reference.
19     df_shapes_reference =
20         df_shapes[['shape_id']].drop_duplicates().reset_index(drop=True)
21     df_shapes_reference['id_carga_gtfs'] = id_carga_gtfs

```

3.2.4.5. Verificar si la Base de Datos tiene registros:

Se instancia una lista que contiene los modelos de la base de datos, donde se realiza una consulta a cada una de ellas si tienen al menos un registro, se considera que la base de datos esta poblada. Retornando un True si la base de datos esta vacía.

Listing 3.18: Verificar si la base de datos tiene registros en varias tablas.

```

1
2 from models import Agency, Calendar, Route, ShapeReference, Shapes, StopTimes,
3     Stop, Trip
4
5 def verificar_registros_db(db: Session):
6     tablas = [Agency, Calendar, Route, ShapeReference, Shapes, StopTimes,
7               Stop, Trip]
8
9     for tabla in tablas:
10         tiene_registros =
11             db.query(exists().where(tabla.id_carga_gtfs.isnot(None))).scalar()
12         if tiene_registros:
13             return False # Si al menos una tabla tiene registros, la base de
14                             datos no est vac a

```

```

11
12     return True # Si ninguna tabla tiene registros, la base de datos est
        vac a

```

Si se indica que la base de datos se encuentra vacía. Se realiza una carga directa a la base de datos.

Listing 3.19: Cargar datos desde un DataFrame a una tabla de la base de datos.

```

1
2 def cargar_datos(df, zip, tabla, db, engine):
3     df.to_sql(tabla, engine, if_exists='append', index=False)
4     actualizar_estado(id_carga_gtfs=zip.id_carga_gtfs, estado="Procesando",
        mensaje=f"Archivo '{tabla}.txt' ha sido cargado a la tabla, siguiendo
        proceso...", db=db)

```

3.2.4.6. Lectura de registros en la base de datos:

La lectura de registros en la base de datos permite acceder a los registros existentes y almacenarlos en nuevos DataFrames.

Listing 3.20: Extraer datos de la base de datos a DataFrames.

```

1
2 def extraer_db(engine):
3     with engine.connect() as connection:
4         df_db_agency = pd.read_sql_table('agency', con=connection)
5         df_db_routes = pd.read_sql_table('route', con=connection)
6         df_db_calendar = pd.read_sql_table('calendar', con=connection)
7         df_db_stops = pd.read_sql_table('stop', con=connection)
8         df_db_shapes_reference = pd.read_sql_table('shape_reference',
            con=connection)
9         df_db_shapes = pd.read_sql_table('shapes', con=connection)
10        df_db_trips = pd.read_sql_table('trip', con=connection)
11        df_db_stop_times = pd.read_sql_table('stop_times', con=connection)
12
13    return df_db_agency, df_db_routes, df_db_calendar, df_db_stops,
        df_db_shapes_reference, df_db_shapes, df_db_trips, df_db_stop_times

```

3.2.4.7. Identificar los registros antiguos.

Para poder identificarlos es necesario tener el dataframe leído directamente de los archivos '.txt' y los extraídos de la base de datos.

Hay que mencionar en este punto que el orden en el cual nosotros eliminamos los registros es sumamente importante, ya que nos encontramos en una base de datos relacional, nosotros tenemos las tablas con referencias entre sí nos podría dar error si lo hacemos en el orden incorrecto.

Siendo el orden recomendado:

- Stop Times
- Trip.
- Shapes.
- Shape Reference.
- Calendar.
- Route.
- Agency.
- Stop.

Listing 3.21: Verificar y eliminar registros obsoletos.

```

1
2 def verificar_registros_obsoletos(zip, db, df_agency, df_calendar, df_routes,
3   df_stop_times, df_stops, df_trips, df_shapes, df_shapes_reference,
4   df_db_agency, df_db_routes, df_db_calendar, df_db_stops,
5   df_db_shapes_reference, df_db_shapes, df_db_trips, df_db_stop_times):
6
7     # Eliminar registros obsoletos
8     eliminar_registros_stop_time(zip, db, df_db_stop_times, df_stop_times,
9       'trip_id', 'stop_id', 'stop_times')
10    eliminar_registros(zip, db, df_db_trips, df_trips, 'trip_id', 'trip')
11    eliminar_registros(zip, db, df_db_shapes, df_shapes, 'shape_id', 'shapes')
12    eliminar_registros(zip, db, df_db_shapes_reference, df_shapes_reference,
13      'shape_id', 'shape_reference')
14    eliminar_registros(zip, db, df_db_calendar, df_calendar, 'service_id',
15      'calendar')
16    eliminar_registros(zip, db, df_db_routes, df_routes, 'route_id', 'route')
17    eliminar_registros(zip, db, df_db_agency, df_agency, 'agency_id', 'agency')
18    eliminar_registros(zip, db, df_db_stops, df_stops, 'stop_id', 'stop')

```

El primer paso para eliminar los registros antiguos, es eliminar tanto la columna del ID de la carga del dataframe de la base de datos como del dataframe leído del '.txt'. Pandas nos brinda la facilidad de poder comparar dos dataframes e identificar si hay registros idénticos en los dos dataframes o si se encuentra solo en uno.

Se utiliza el método merge para combinar los DataFrames 'df-db-sin-id' y 'df-sin-id' utilizando el método de unión externa ('outer') y se agrega una columna indicadora ('-merge') que muestra de qué DataFrame proviene cada fila. Team (2023)

Se filtran las filas de comparar que solo están presentes en el DataFrame df-db-sin-id (aquellas para las cuales -merge es 'left-only'). Estos registros representan los que están en la base de datos pero no en el nuevo DataFrame.

Seleccionamos el ID de cada elemento que queremos eliminar y realizamos una consulta a la base de datos donde los identifica y los elimina.

Si hay registros antiguos, se construye y ejecuta una consulta SQL DELETE para eliminar esos registros de la tabla especificada en tabla.

Listing 3.22: Eliminar registros obsoletos de una tabla.

```

1
2 def eliminar_registros(zip, db, df_db, df, id_tabla, tabla):
3     df_db_sin_id = df_db.drop(columns='id_carga_gtfs')
4     df_sin_id = df.drop(columns='id_carga_gtfs')
5     comparar = df_db_sin_id.merge(df_sin_id, indicator=True, how='outer')
6
7     # Filtrar los registros
8     registros_antiguos = comparar.loc[lambda x: x['_merge'] == 'left_only']
9     # Obtener una lista de los valores de la columna 'id_tabla' de los
       registros_antiguos
10
11     registros_a_eliminar = registros_antiguos[id_tabla].tolist()
12     if registros_a_eliminar:
13         query = text(f"DELETE FROM {tabla} WHERE {id_tabla} IN ({',
           '.join(map(repr, registros_a_eliminar))})")
14         db.execute(query)
15         actualizar_estado(id_carga_gtfs=zip.id_carga_gtfs,
           estado="Procesando", mensaje=f"Se eliminaron los registros
           obsoletos en la tabla '{tabla}'.", db=db)
16     else:
17         actualizar_estado(id_carga_gtfs=zip.id_carga_gtfs,
           estado="Procesando", mensaje=f"No se encontraron registros
           obsoletos para eliminar en la tabla '{tabla}'.", db=db)

```

Stop Times tiene un detalle particular, stop times tiene 2 llaves primarias. Por lo tanto para eliminar los registros la consulta de la base de datos se modifica y espera un argumento extra para el correcto funcionamiento.

Listing 3.23: Eliminar registros obsoletos de la tabla 'stop_{imes}'.

```
1
2 def eliminar_registros_stop_time(zip, db, df_db, df, pk1, pk2, tabla):
3     df_db_sin_id = df_db.drop(columns='id_carga_gtfs')
4     df_sin_id = df.drop(columns='id_carga_gtfs')
5
6     # Realizar una comparaci n entre los DataFrames sin la columna
7     'id_carga_gtfs'
8
9     comparar = df_db_sin_id.merge(df_sin_id, how='left', indicator=True)
10    # Filtrar los registros que est n en df_db pero no en df (left_only)
11    registros_antiguos = comparar.loc[lambda x: x['_merge'] == 'left_only']
12
13    if not registros_antiguos.empty:
14        # Crear una subconsulta para verificar la existencia de registros en
15        la tabla
16        subquery = (
17            f"SELECT 1 FROM {tabla} AS t "
18            f"WHERE t.{pk1} = {tabla}.{pk1} AND t.{pk2} = {tabla}.{pk2}"
19        )
20        query = text(f"DELETE FROM {tabla} WHERE EXISTS ({subquery})")
21        db.execute(query)
22        actualizar_estado(id_carga_gtfs=zip.id_carga_gtfs,
23                           estado="Procesando", mensaje=f"Se eliminaron los registros
24                           obsoletos de la tabla '{tabla}'.", db=db)
25    else:
26        actualizar_estado(id_carga_gtfs=zip.id_carga_gtfs,
27                           estado="Procesando", mensaje=f"No se encontraron registros
28                           obsoletos para eliminar en la tabla '{tabla}'.", db=db)
```

3.2.4.8. Identificar los registros nuevos:

Al igual que en proceso anterior, para poder identificarlos es necesario tener el dataframe leído directamente de los archivos '.txt' y los extraídos de la base de datos. Hay que mencionar en este punto que el orden en el cual nosotros agregamos los registros es sumamente importante, ya que nos encontramos en una base de datos relacional, nosotros tenemos las tablas con referencias entre sí nos podría dar error si lo hacemos en el orden incorrecto.

Siendo el orden recomendado:

- Stop.
- Agency.
- Route.
- Calendar.
- Shape Reference.
- Shapes.
- Trip.
- Stop Times.

Listing 3.24: Verificar y agregar nuevos registros a las tablas correspondientes.

```
1
2 def verificar_agregar_registros(zip, db, df_agency, df_calendar, df_routes,
3   df_stop_times, df_stops, df_trips, df_shapes, df_shapes_reference,
4   df_db_agency, df_db_routes, df_db_calendar, df_db_stops,
5   df_db_shapes_reference, df_db_shapes, df_db_trips, df_db_stop_times):
6   # Agregar nuevos registros
7   agregar_registros_stops(zip, db, df_db_stops, df_stops, 'stop')
8   agregar_registros(zip, db, df_db_agency, df_agency, 'agency')
9   agregar_registros(zip, db, df_db_routes, df_routes, 'route')
10  agregar_registros(zip, db, df_db_calendar, df_calendar, 'calendar')
11  agregar_registros(zip, db, df_db_shapes_reference, df_shapes_reference,
    'shape_reference')
    agregar_registros(zip, db, df_db_shapes, df_shapes, 'shapes')
    agregar_registros(zip, db, df_db_trips, df_trips, 'trip')
    agregar_registros(zip, db, df_db_stop_times, df_stop_times, 'stop_times')
```

El primer paso para agregar los registros nuevos, es eliminar tanto la columna del ID de la carga del dataframe de la base de datos como del dataframe leído del '.txt'. Pandas nos brinda la facilidad de poder comparar dos dataframes e identificar si hay registros idénticos en los dos dataframes o si se encuentra solo en uno.

Utilizando el método merge, se fusionan los DataFrames 'df-db-sin-id' y 'df-sin-id' mediante una unión externa ('outer'), incorporando una columna indicadora ('_merge') que indica el origen de cada fila. [Team \(2023\)](#)

Se filtran las filas de comparar que solo existen en el DataFrame 'df-sin-id' (aquellas para las cuales _merge es 'right-only'). Estos registros representan datos nuevos que no estaban presentes en la base de datos original. Se añade la columna 'id-carga-gtfs' del DataFrame df a los registros nuevos.

Se intenta insertar los registros nuevos en la tabla especificada en tabla mediante el método to_sql de Pandas.

Listing 3.25: Agregar nuevos registros a una tabla en la base de datos.

```

1
2 def agregar_registros(zip, db, df_db, df, tabla):
3     df_db_sin_id = df_db.drop(columns='id_carga_gtfs')
4     df_sin_id = df.drop(columns='id_carga_gtfs')
5
6     # Realizar una comparación entre los DataFrames
7     comparar = df_db_sin_id.merge(df_sin_id, indicator=True, how='outer')
8
9     # Filtrar los registros que están en df pero no en df_db (right-only)
10    registros_nuevos = comparar.loc[lambda x: x['_merge'] == 'right-only']
11    registros_nuevos.reset_index(drop=True, inplace=True)
12
13    if not registros_nuevos.empty:
14        # Asignar los valores de 'id_carga_gtfs' utilizando .loc
15        registros_nuevos.loc[:, 'id_carga_gtfs'] =
16            df.loc[registros_nuevos.index, 'id_carga_gtfs']
17        # Eliminar la columna '_merge'
18        registros_nuevos.drop(columns='_merge', inplace=True)
19        # Agregar los registros nuevos a la tabla en la base de datos
20        registros_nuevos.to_sql(tabla, engine, if_exists='append', index=False)
21        actualizar_estado(id_carga_gtfs=zip.id_carga_gtfs,
22                           estado="Procesando", mensaje=f"Se agregaron los registros nuevos
23                           en la tabla de '{tabla}'.", db=db)
24    else:
25        actualizar_estado(id_carga_gtfs=zip.id_carga_gtfs,
26                           estado="Procesando", mensaje=f"No se encontraron registros nuevos
27                           para agregar en la tabla '{tabla}'.", db=db)

```

Con la tabla 'Stops' se opera de una forma distinta ya que tiene una columna nueva con los ID incrementales.

La instancia con los registros nuevos, lo que se hace es hacer una consulta de cual es el máximo número que hay en la columna de 'id-interno' del dataframe de la base de datos y a los registros nuevos se le suma uno de forma secuencial, de esta forma se evitan colisiones entre registros. Y luego la consulta que ingresa registros a la base de datos.

Listing 3.26: Agregar nuevos registros a la tabla 'stops' en la base de datos.

```

1 def agregar_registros_stops(zip, db, df_db, df, tabla):
2     df_db_sin_id = df_db.drop(columns=['id_carga_gtfs', 'stop_id_interno'])
3     df_sin_id = df.drop(columns='id_carga_gtfs')
4
5     # Realizar una comparación entre los DataFrames
6     comparar = df_db_sin_id.merge(df_sin_id, on=list(df_db_sin_id.columns),
7                                   indicator=True, how='outer')
8
9     # Filtrar los registros que están en df pero no en df_db (right_only)
10    registros_nuevos = comparar.loc[lambda x: x['_merge'] == 'right_only']
11
12    # Asignar valores de 'id_carga_gtfs' utilizando .loc
13    registros_nuevos['id_carga_gtfs'] =
14        df.reset_index().loc[registros_nuevos.index]['id_carga_gtfs']
15
16    # Eliminar la columna '_merge'
17    registros_nuevos = registros_nuevos.drop(columns='_merge')
18
19    # Obtener el valor máximo existente de 'stop_id_interno'
20    max_existing_id = df_db['stop_id_interno'].max()
21
22    # Asignar nuevos valores de 'stop_id_interno'
23    registros_nuevos['stop_id_interno'] = range(max_existing_id + 1,
24                                                max_existing_id + 1 + len(registros_nuevos))
25
26    # Agregar los registros nuevos a la tabla en la base de datos
27    registros_nuevos.to_sql(tabla, engine, if_exists='append', index=False)
28
29    # Actualizar el estado
30    actualizar_estado(id_carga_gtfs=zip.id_carga_gtfs, estado="Procesando",
31                      mensaje=f"Se agregaron los registros nuevos en la tabla de
32                        '{tabla}'.", db=db)

```

3.2.4.9. Generar un nuevo ZIP:

Si el proceso de Carga ha sido un éxito se genera un nuevo ZIP, que contenga solo los elementos que fueron usados durante la carga, de esta forma podemos mantener un registro de como fue el archivo que fue utilizado en cierta carga.

Listing 3.27: Finalizar el proceso de carga y generar un nuevo archivo ZIP.

```
1 def terminar_carga(zip, db, zips_procesados, directorio_txts):
2     try:
3         # Eliminar archivos en el directorio_txts
4         for archivo in os.listdir(directorio_txts):
5             ruta_completa = os.path.join(directorio_txts, archivo)
6             try:
7                 if os.path.isfile(ruta_completa):
8                     os.unlink(ruta_completa)
9             except Exception as e:
10                print(f"No se pudo eliminar {ruta_completa}. Raz n: {e}")
11
12        # Crear un nuevo archivo ZIP procesado
13        shutil.make_archive(f'{zips_procesados}/{zip.zip}_procesado', 'zip',
14                           'txts')
15
16        # Actualizar el estado
17        actualizar_estado(id_carga_gtfs=zip.id_carga_gtfs, estado="Terminado",
18                          mensaje='Nuevo archivo ZIP generado, contiene nicamente archivos
19                          esperados', db=db)
20    except Exception as e:
21        # Manejar errores
22        actualizar_estado(id_carga_gtfs=zip.id_carga_gtfs, estado="Error",
23                          mensaje=f"Se presentaron problemas al terminar la carga: {e}",
24                          db=db)
```

3.2.4.10. Manejar errores en la carga:

En caso de que se produzca una excepción durante el proceso de carga, se implementa un protocolo de seguridad. Antes de iniciar la carga, se realiza una copia de seguridad de los datos existentes en la base de datos. Si ocurre una excepción durante la carga de nuevos datos, se garantiza que se ingrese la copia de seguridad para preservar la integridad de la base de datos. Además, se notifica el error, proporcionando información detallada sobre la naturaleza del problema encontrado durante el proceso de carga.



Capítulo 4

Validación de la Solución

4.1. Ejecución del componente de Carga de datos GTFS:

Cuando un registro se ingresa en el sistema, se activa el endpoint correspondiente. Este proceso hace que el sistema esté siempre actualizado y listo para responder a nuevas entradas, logrando una integración de los datos ingresados.

FastAPI proporciona una sección especialmente útil conocida como `"/docs"` que facilita la realización de llamadas a las APIs. Se ha aprovechado esta funcionalidad para probar y validar nuestro endpoint denominado `"/carga"` como vemos en el apartado A.1.

La primera carga a nuestra base de datos, transforma el estado inicial vacío a una totalmente poblada gracias al sistema.

Posteriormente, incorporamos un segundo archivo ZIP al sistema y se activa nuevamente el endpoint correspondiente. En este ciclo, se desencadena el proceso de comparación, donde se eliminan los registros obsoletos y se añaden los nuevos. Al finalizar esta operación se observa la asociación de cada nuevo registro agregado con su carga correspondiente.

default

GET / Root

POST /carga Carga

Parameters

No parameters

Servers

These operation-level options override the global server options.

/

Execute

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value Schema

"string"

Figura 4.1: Panel para ejecucion de endpoint.

	id_carga_gtfs [PK] integer	id_usuario character varying (255)	zip character varying (255)	fecha timestamp without time zone	estado character varying (50)	mensaje character varying (255)
1	1	1	Gran Concepción20231018	2023-11-29 15:32:53.303425	Sin Procesar	Esperando a ser procesado

Figura 4.2: Primera Carga.

	id_carga_gtfs integer	trip_id [PK] character varying (50)	arrival_time character varying (50)	departure_time character varying (50)	stop_id [PK] character varying (50)	stop_sequence character varying (50)	stop_headsign character varying (255)	pickup_type character varying (50)
1	1	00140749-b-c36b9237-d	23:16:00	23:16:00	16005188	33	[null]	[null]
2	1	00140749-b-c36b9237-d	22:49:00	22:49:00	36935	7	[null]	[null]
3	1	00140749-b-c36b9237-d	22:37:00	22:37:00	36938	1	[null]	[null]
4	1	00140749-b-c36b9237-d	22:49:00	22:49:00	37042	6	[null]	[null]
5	1	00140749-b-c36b9237-d	22:49:00	22:49:00	37043	8	[null]	[null]
6	1	00140749-b-c36b9237-d	22:50:00	22:50:00	37044	9	[null]	[null]
7	1	00140749-b-c36b9237-d	22:51:00	22:51:00	37045	10	[null]	[null]
8	1	00140749-b-c36b9237-d	22:55:00	22:55:00	37046	16	[null]	[null]
9	1	00140749-b-c36b9237-d	22:56:00	22:56:00	37047	17	[null]	[null]
10	1	00140749-b-c36b9237-d	22:57:00	22:57:00	37048	18	[null]	[null]
11	1	00140749-b-c36b9237-d	22:51:00	22:51:00	37099	11	[null]	[null]
12	1	00140749-b-c36b9237-d	22:53:00	22:53:00	37101	13	[null]	[null]
13	1	00140749-b-c36b9237-d	23:11:00	23:11:00	37104	31	[null]	[null]
14	1	00140749-b-c36b9237-d	23:06:00	23:06:00	37442	28	[null]	[null]

Total rows: 1000 of 2874403 Query complete 00:00:02.441 Ln 1, Col 1

Figura 4.3: Datos de la Primera Carga ingresado a la base de Datos.

	id_carga_gtfs [PK] integer	id_usuario character varying (255)	zip character varying (255)	fecha timestamp without time zone	estado character varying (50)	mensaje character varying (255)
1	1	1	Gran Concepción20231018	2023-11-29 15:37:34.208563	Terminado	Nuevo archivo ZIP generado, contiene únicamente archivos esperad...
2	2	1	Gran Concepción20231107	2023-11-29 15:44:39.896773	Sin Procesar	Esperando a ser procesado

Figura 4.4: Segunda Carga.



	id_carga_gtfs integer	trip_id [PK] character varying (50)	arrival_time character varying (50)	departure_time character varying (50)	stop_id [PK] character varying (50)	stop_sequence character varying (50)	stop_headsign character varying (255)	pickup_type character varying (50)
1	2	000104df-6-44615707-3	15:18:00	15:18:00	2-Jan	39	[null]	[null]
2	2	000104df-6-44615707-3	15:43:00	15:43:00	35547	67	[null]	[null]
3	2	000104df-6-44615707-3	15:42:00	15:42:00	35548	66	[null]	[null]
4	2	000104df-6-44615707-3	15:19:00	15:19:00	35690	41	[null]	[null]
5	2	000104df-6-44615707-3	15:10:00	15:10:00	35691	33	[null]	[null]
6	2	000104df-6-44615707-3	15:11:00	15:11:00	35693	34	[null]	[null]
7	2	000104df-6-44615707-3	15:12:00	15:12:00	35694	35	[null]	[null]
8	2	000104df-6-44615707-3	15:13:00	15:13:00	35695	36	[null]	[null]
9	2	000104df-6-44615707-3	15:14:00	15:14:00	35696	37	[null]	[null]
10	2	000104df-6-44615707-3	15:17:00	15:17:00	35697	38	[null]	[null]
11	2	000104df-6-44615707-3	15:21:00	15:21:00	35700	44	[null]	[null]
12	2	000104df-6-44615707-3	15:23:00	15:23:00	35703	45	[null]	[null]
13	2	000104df-6-44615707-3	15:23:00	15:23:00	35705	46	[null]	[null]
14	2	000104df-6-44615707-3	15:24:00	15:24:00	35706	47	[null]	[null]
15	2	000104df-6-44615707-3	15:25:00	15:25:00	35707	48	[null]	[null]

Total rows: 1000 of 2883203 Query complete 00:00:06.838 Ln 1, Col 1

Figura 4.5: Datos de la Segunda Carga ingresado a la base de Datos.



Capítulo 5

Conclusiones - Trabajos Futuros

5.1. Conclusiones:

La implementación del sistema de automatización para la gestión de registros en archivos GTFS ha mostrado resultados positivos, cumpliendo con los objetivos planteados.

La trazabilidad de los datos se ha logrado de manera efectiva, permitiendo un seguimiento claro de la procedencia y estado de cada registro. Esto nos da una visión detallada de los datos a lo largo del tiempo.

Se implanta un proceso que establece reglas específicas a los archivos ZIP y su contenido, asegurando que contengan la información necesaria para una carga efectiva en el sistema.

Además, se presenta una propuesta para un sistema de carga automatizada que se adapta fácilmente a las necesidades específicas de cada ciudad que requiera automatización de registros GTFS. La flexibilidad del sistema permite su modificación y personalización, convirtiéndolo en una herramienta para futuras implementaciones en distintos contextos.

Las herramientas como FastAPI, SQLAlchemy y Pandas, ofrecieron un buen resultado dentro del proyecto. Pero cabe señalar que el sistema, en su estado actual, realiza una transferencia completa de la información desde el disco a la memoria, lo que resulta en un uso significativo de recursos de memoria. Este enfoque, aunque efectivo para la comparación y procesamiento de datos, plantea consideraciones importantes sobre la optimización del consumo de memoria.

Esta propuesta no solo cumple con los requisitos actuales, sino que también sienta las bases para mejoras continuas y futuros proyectos. La optimización del uso de memoria será una consideración clave para garantizar un rendimiento eficiente en futuras versiones del sistema.

5.2. Trabajos Futuros

En esta sección, se presentan propuestas para futuros desarrollos que no solo fortalecerán la robustez del sistema actual, sino que también permitirán abordar desafíos emergentes y garantizar un mejor rendimiento

a medida que evolucione.

Como primera propuesta, se sugiere considerar contenerizar el componente, de esta manera creamos un entorno aislado que encapsula todas las dependencias y configuraciones necesarias. Esto no solo simplifica la integración con otros servicios, sino que también hace que nuestro proceso sea más portable y escalable. Utilizando herramientas como Docker, podemos asegurar una ejecución coherente independientemente del entorno en el que se despliegue.

Se propone explorar estrategias para optimizar el consumo de memoria durante el proceso. La eficiencia en el manejo de recursos para garantizar un mejor rendimiento, especialmente cuando tenemos grandes volúmenes de datos.

Además de estas propuestas, se sugiere fortalecer el sistema de extracción de archivos para garantizar un procesamiento sin problemas. En muchos casos, los desafíos pueden surgir cuando los archivos de entrada no siguen un patrón específico o cuando se ingresan formatos no compatibles. Para abordar esto, se propone implementar un filtro en el buzón donde se reciben los archivos comprimidos, de modo que solo se permita el ingreso de archivos con la extensión específica requerida (por ejemplo, archivos zip). Al restringir la entrada a un formato conocido y predefinido, se reduce la posibilidad de errores y se mejora la confiabilidad del sistema.

Esta medida no solo simplificará el proceso de extracción, sino que también contribuirá a la estandarización de los datos de entrada, lo que facilitará el desarrollo futuro y la implementación de nuevas funcionalidades. Además, al limitar la entrada a un formato específico, se facilita la implementación de validaciones adicionales para asegurar la integridad y autenticidad de los archivos recibidos.

Bibliografía

- Airflow, A. (2023). Apache airflow documentation. <https://airflow.apache.org/docs/apache-airflow/stable/index.html>. 2.13
- Docker (2023). Get started with docker. <https://docs.docker.com/get-started/>. 2.12
- GTFS (2023). Gtfs schedule reference. <https://gtfs.org/schedule/reference/>. 2.4
- Inc., A. (2023). Transporte público estático. <https://web.archive.org/web/diff/20230528025933/20221219194954/https://developers.google.com/transit/gtfs/reference?hl=es-419>. 3.1
- Moovit (2023). Moovit transit data manager. <https://moovit.com/maas-solutions/transit-data-manager/>. 1.5
- Romero Munizaga, J. (2021). Gestor de archivos gtfs para transapp. <https://repositorio.uchile.cl/handle/2250/181367>. 3.1
- Team, P. D. (2023). pandas.dataframe.merge. <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html>. 3.2.4.7, 3.2.4.8
- Tech, S. H. (2023). Django vs fastapi: A detailed comparison. https://medium.com/@ShortHills_Tech/django-vs-fast-api-a-detailed-comparison-df8d00f3c3b2. 3.1
- Tiangolo (2023). Fastapi tutorial - sql databases. <https://fastapi.tiangolo.com/tutorial/sql-databases/>. 3.1
- Transit, T. (2023). Gtfs manager. <https://trilliumtransit.com/gtfs/gtfs-manager/>. 1.5



Anexo A

Apendice A

A.1. Endpoint de la Carga:

Listing A.1: Endpoint de la Carga.

```

1  @carga_gtfs.post("/carga")
2  async def carga(db: Session = Depends(get_db)):
3      zip_info = identificar_zip(db=db)
4      zip_ruta = config["almacen_zips"] + zip_info.zip + ".zip"
5
6      extraer_zip(zip=zip_info, zip_ruta=zip_ruta, db=db)
7
8      inspeccionar_txts(zip=zip_info,
9          almacen_txts=config["almacen_txts"], zip_ruta=zip_ruta, db=db)
10     df_agency, df_calendar, df_routes, df_stop_times, df_stops,
11     df_trips, df_shapes, df_shapes_reference = transformar_txts(
12         zip=zip_info, db=db
13     )
14
15     (
16         db_inicial_agency,
17         db_inicial_routes,
18         db_inicial_calendar,
19         db_inicial_stops,
20         db_inicial_shapes_reference,
21         db_inicial_shapes,
22         db_inicial_trips,
23         db_inicial_stop_times,
24     ) = extraer_db(engine=engine)
25
26     bd_vacia = verificar_registros_db(db=db)

```

```

27     if bd_vacia:
28         primera_carga(
29             zip=zip_info,
30             db=db,
31             engine=engine,
32             df_agency=df_agency,
33             df_calendar=df_calendar,
34             df_routes=df_routes,
35             df_stop_times=df_stop_times,
36             df_stops=df_stops,
37             df_trips=df_trips,
38             df_shapes=df_shapes,
39             df_shapes_reference=df_shapes_reference,
40         )
41
42     terminar_carga(zip=zip_info, db=db,
43                   zips_procesados=config["zips_procesados"],
44                   directorio_txts=config["almacen_txts"])
45     return "Termino correctamente"
46 else:
47     try:
48         (
49             df_db_agency,
50             df_db_routes,
51             df_db_calendar,
52             df_db_stops,
53             df_db_shapes_reference,
54             df_db_shapes,
55             df_db_trips,
56             df_db_stop_times,
57         ) = extraer_db(engine=engine)
58
59     verificar_registros_obsoletos(
60         zip=zip_info,
61         df_agency=df_agency,
62         df_calendar=df_calendar,
63         df_routes=df_routes,
64         df_stop_times=df_stop_times,
65         df_stops=df_stops,
66         df_trips=df_trips,
67         df_shapes=df_shapes,
68         df_shapes_reference=df_shapes_reference,
69         df_db_agency=df_db_agency,
70         df_db_routes=df_db_routes,
71         df_db_calendar=df_db_calendar,
72         df_db_stops=df_db_stops,
73         df_db_shapes_reference=df_db_shapes_reference,

```

```

72         df_db_shapes=df_db_shapes,
73         df_db_trips=df_db_trips,
74         df_db_stop_times=df_db_stop_times,
75         db=db,
76     )
77
78     (
79         df_db_agency,
80         df_db_routes,
81         df_db_calendar,
82         df_db_stops,
83         df_db_shapes_reference,
84         df_db_shapes,
85         df_db_trips,
86         df_db_stop_times,
87     ) = extraer_db(engine=engine)
88
89     verificar_agregar_registros(
90         zip=zip_info,
91         df_agency=df_agency,
92         df_calendar=df_calendar,
93         df_routes=df_routes,
94         df_stop_times=df_stop_times,
95         df_stops=df_stops,
96         df_trips=df_trips,
97         df_shapes=df_shapes,
98         df_shapes_reference=df_shapes_reference,
99         df_db_agency=df_db_agency,
100        df_db_routes=df_db_routes,
101        df_db_calendar=df_db_calendar,
102        df_db_stops=df_db_stops,
103        df_db_shapes_reference=df_db_shapes_reference,
104        df_db_shapes=df_db_shapes,
105        df_db_trips=df_db_trips,
106        df_db_stop_times=df_db_stop_times,
107        db=db,
108    )
109
110    terminar_carga(zip=zip_info, db=db,
111                   zips_procesados=config["zips_procesados"],
112                   directorio_txts=config["almacen_txts"])
113
114    return "La carga termino con exito."
115
116 except SQLAlchemyError as e:
117
118     restaurar_db(db)

```

```
117
118     primera_carga(
119         zip=zip_info,
120         db=db,
121         engine=engine,
122         df_agency=db_inicial_agency,
123         df_calendar=db_inicial_calendar,
124         df_routes=db_inicial_routes,
125         df_stop_times=db_inicial_stop_times,
126         df_stops=db_inicial_stops,
127         df_trips=db_inicial_trips,
128         df_shapes=db_inicial_shapes,
129         df_shapes_reference=db_inicial_shapes_reference,
130     )
131
132     carga_error(zip=zip_info, db=db,
133                 zips_procesados=config["zips_procesados"],
134                 directorio_txts=config["almacen_txts"])
135     raise HTTPException(status_code=500, detail=f"Error
136     durante la carga: {str(e)}")
137
138 except Exception as e:
139     restaurar_db(db)
140
141     primera_carga(
142         zip=zip_info,
143         db=db,
144         engine=engine,
145         df_agency=db_inicial_agency,
146         df_calendar=db_inicial_calendar,
147         df_routes=db_inicial_routes,
148         df_stop_times=db_inicial_stop_times,
149         df_stops=db_inicial_stops,
150         df_trips=db_inicial_trips,
151         df_shapes=db_inicial_shapes,
152         df_shapes_reference=db_inicial_shapes_reference,
153     )
154
155     carga_error(zip=zip_info, db=db,
156                 zips_procesados=config["zips_procesados"],
157                 directorio_txts=config["almacen_txts"])
158     raise HTTPException(status_code=500, detail=f"Error
159     durante la carga: {str(e)}")
```

A.2. Contenido archivos GTFS:

Tabla A.1: Ejemplo de Datos del archivo: Agency.txt

agency_id	agency_name	agency_url	agency_timezone	agency_lang	agency_phone
DTPR	División de Transporte Público Regional	http://www.dtp.r.gob.cl/	America/Santiago	es	+562 2421 3580

Tabla A.2: Ejemplo de Datos del archivo: Calendar.txt

service_id	start_date	end_date	monday	tuesday	wednesday	thursday	friday	saturday	sunday
S	20210101	20311231	0	0	0	0	0	1	0
L	20210101	20311231	1	1	1	1	1	0	0
D	20210101	20311231	0	0	0	0	0	0	1

Tabla A.3: Ejemplo de Datos del archivo: Routes.txt

route_id	agency_id	route_short_name	route_long_name
606	DTPR	44EU	Flota Centauro 44U - Hualpencillo - Flota Centauro 44E - Lomas-Mall
540	DTPR	40I	Tomé - Concepción
567	DTPR	17OX	Expresos Chiguayante 17X - Leonera - Expresos Chiguayante 17O - Nonguén
602	DTPR	42FW	Buses Hualpencillo 42W - 4 Esquinas - Buses Hualpencillo 42F - Concepción
725	DTPR	70I	Las Bahías Nonguén - Las Bahías 70I - Centinela

Tabla A.4: Ejemplo de Datos del archivo: Shapes.txt

shape_id	shape_pt_lat	shape_pt_lon	shape_pt_sequence	shape_dist_traveled
1211667534	-36.779165	-73.099251	1	0.0
1211667534	-36.779309	-73.098583	2	61.610190285531615
1211667534	-36.778807	-73.098382	3	120.2303660554362
1211667534	-36.77881	-73.09837	4	121.34996122033635
1211667534	-36.77853	-73.09823	5	154.88849614102278

Tabla A.5: Ejemplo de Datos del archivo: Stop-times.txt

trip_id	arrival_time	departure_time	stop_id	stop_sequence
15713-44615707-3	05:00:00	05:00:00	38477	1
15713-44615707-3	05:01:00	05:01:00	38508	2
15713-44615707-3	05:02:00	05:02:00	38511	3
15713-44615707-3	05:03:00	05:03:00	38482	4
15713-44615707-3	05:03:00	05:03:00	38440	5

Tabla A.6: Ejemplo de Datos del archivo: Stops.txt

stop_id	stop_name	stop_lat	stop_lon
37905	Quillota Baker	-36.77786882	-73.09451695
42606	Suecia esq Las Golondrinas	-36.78079052	-73.09633281
42607	Suecia esq Escocia Poniente	36.78231792	-73.09673246
42608	Suecia esq Finlandia	-36.78452841	-73.09672978
2-Apr	Padre Hurtado esq Av. Chacabuco	-36.834242078	-73.057396389

Tabla A.7: Ejemplo de Datos del archivo: Trips.txt

route_id	service_id	trip_id	trip_headsign	direction_id	block_id	shape_id
550	S	15713-44615707-3	Vía Láctea 10O - Pobl. Porvenir	1		1211667428
550	S	15663-44615707-3	Vía Láctea 10B - Lobos Viejos	0		1211667427
550	S	15728-44615707-3	Vía Láctea 10O - Pobl. Porvenir	1		1211667428
550	S	15678-44615707-3	Vía Láctea 10B - Lobos Viejos	0		1211667427
550	S	15743-44615707-3	Vía Láctea 10O - Pobl. Porvenir	1		1211667428