Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática.

Interfaz de usuario WEB para la accesibilidad de los diferentes desarrollos basados en Optimización e Inteligencia Artificial.

WEB user interface for accessibility of various developments based on Optimization and Artificial Intelligence.

Nicolas Daniel Vegas Rodríguez

San Cristóbal de La Laguna, 07 de Mayo de 2023.

D. Moreno Pérez José Andrés, con N.I.F. 42.935.437-A catedrático de Ciencia de la Computación e Inteligencia Artificial adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

D. León Rodríguez Ginés, con N.I.F. 78.408.712-X responsable del departamento de Big Data de Transportes Interurbanos de Tenerife (TITSA), como cotutor.

C E R T I F I C A (N)

Que la presente memoria titulada:

”Interfaz de usuario WEB para la accesibilidad de los diferentes desarrollos basados en Optimización e Inteligencia Artificial”.

Ha sido realizada bajo su dirección por **D. Nicolas Daniel Vegas Rodríguez**, con N.I.F. 54.865.345-H.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 26 de mayo de 2023.

Agradecimientos

En primer lugar quiero agradecer a mi familia por su amor, apoyo y comprensión durante todo mi proceso de formación académica. En especial, quiero agradecer a mi madre y hermano por ser mi mayor soporte. Gracias a su apoyo incondicional, he podido dedicar tiempo y esfuerzo a este proyecto, y lograr completarlo con éxito. También, agradecer a mi tutor José Andrés Moreno Pérez y co-tutor Ginés León Rodríguez así como a todo el departamento de big data de TITSA por guiarme a lo largo del desarrollo de este proyecto. Por último, agradecer a todos los compañeros que he conocido a lo largo de mi carrera en la Universidad y en las prácticas externas.

A todos ellos, muchas gracias de corazón.

Licencia

Resumen

El objetivo principal de este trabajo final de grado es desarrollar una **herramienta web** que permita ejecutar **algoritmos** de **optimización** para la gestión del **transporte** de forma amigable y accesible para mejorar la **experiencia de usuario.**

Los **datos** en el ámbito del transporte son de vital importancia, ya que afectan a todas las decisiones en la **gestión de las rutas** y vehículos. La interfaz visual debe ser capaz de recoger un fichero de entrada y ejecutar el algoritmo de optimización configurado por el usuario.

Además, se busca facilitar la **escalabilidad** de la herramienta para incorporar nuevos algoritmos en el futuro. Por último, se debe permitir la descarga de la **solución** en un fichero CSV y la **representación de los resultados.**

Palabras clave: transporte, datos, gestión de rutas, experiencia de usuario, optimización, algoritmos, desarrollo web, escalabilidad, solución, representación de resultados.

Abstract

The main objective of this final degree project is to develop a web tool that allows for the execution of optimization algorithms for **transportation** management in a **user-friendly** and accessible manner to enhance the user experience.

Data in the transportation field is crucial, as it affects all decisions in **route** and vehicle management. The visual interface should be capable of collecting an input file and executing the **optimization** algorithm configured by the user.

Furthermore, the aim is to facilitate the **scalability** of the tool for the incorporation of new **algorithms** in the future. The tool should allow for the download of the **solution** in a CSV file and the **representation of results** through the developed tool.

Keywords: transportation, data, route management, user-friendly, optimization, algorithms, web tool, scalability, solution, representation of results.

Índice General

Índice de figuras

Índice de tablas

Capítulo 1: Introducción

1.1. Definición del problema

En el ámbito del transporte, la gestión de datos es esencial para la toma de decisiones en la planificación de rutas y la asignación de recursos. En el caso específico de Tenerife, la población ha experimentado un crecimiento notable en los últimos años, lo que plantea desafíos adicionales para las empresas de transporte, un claro ejemplo se puede ver en el siguiente gráfico:

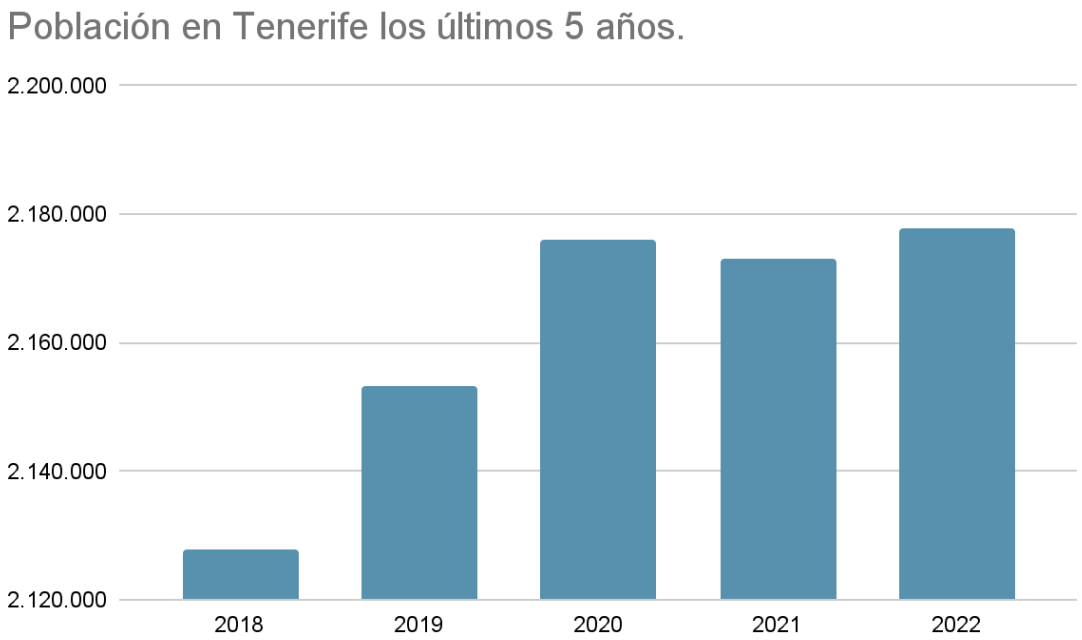


Figura 1: Gráfico de población en Tenerife en los últimos 5 años.
Fuente: Instituto Canario de Estadística (ISTAC), [URL].

Debido al crecimiento de la población, como se evidencia en la figura 1, las empresas de transporte se encuentran en la necesidad de desarrollar algoritmos más sofisticados y precisos. Para ello, sus departamentos de Big Data se dedican a analizar continuamente nuevos datos con el objetivo de generar mejores resultados.

Sin embargo, es importante destacar que los resultados obtenidos a través de estos algoritmos no sólo son relevantes para los departamentos especializados en matemáticas e informática, sino que también deben ser comprensibles para otros departamentos, como el de marketing. Esto implica que cada algoritmo debe contar con visualizaciones claras y concisas que puedan ser comprendidas por personas que no poseen un conocimiento profundo en matemáticas o informática.

1.2. Justificación

Para que los distintos departamentos puedan comprender y estudiar la implementación de las soluciones informáticas desarrolladas por el departamento de titsa, surge la necesidad de desarrollar distintas visualizaciones para cada algoritmo.

Sin embargo, a la hora de desarrollar las visualizaciones, a medida que aumentan los algoritmos disponibles, también tenemos la necesidad de centralizar dichos algoritmos y sus visualizaciones, de esta manera mejoramos la eficiencia del trabajo realizado por el departamento de Big Data, permitiendo que, mediante una herramienta y con determinadas instrucciones, los distintos departamentos puedan generar las visualizaciones correspondientes de los algoritmos deseados.

Es por todo lo previamente mencionado, que esté presente proyecto consiste en realizar una interfaz visual que permita la visualización de los resultados generados por los distintos algoritmos, adicionalmente, se plantea la necesidad de centralizar la ejecución de los algoritmos, permitiendo así, un desarrollo efectivo y sencillo de los distintos algoritmos.

1.3. Objetivo

El objetivo de este proyecto es desarrollar una herramienta web que permita a los usuarios ejecutar diferentes algoritmos de optimización de manera amigable y escalable para mejorar la gestión de la empresa de transporte. La herramienta debe ser capaz de recopilar uno o varios ficheros de entrada, configurar los parámetros del algoritmo, ejecutarlo y descargar la solución en uno o varios ficheros CSV.

Además, la escalabilidad de la herramienta es un aspecto clave a considerar. El transporte es un sector en constante evolución, con nuevos desafíos y cambios en los requerimientos. Por lo tanto, la herramienta debe tener la capacidad de adaptarse e incorporar nuevos algoritmos de optimización en el futuro, sin requerir cambios significativos en la infraestructura existente.

Capítulo 2: Estudio Previo

2.1. Estado del arte

En esta sección, se revisarán los elementos clave relacionados con el desarrollo de visualizadores de resultados de algoritmos y examinaremos los enfoques actuales utilizados en aplicaciones similares. Se llevará a cabo un análisis exhaustivo de diversas herramientas y tecnologías empleadas en la visualización de datos, con especial atención en soluciones que garanticen una representación efectiva de los resultados de los algoritmos desarrollados específicamente para la empresa TITSA.

Además, se abordará el estado del arte de un algoritmo que he desarrollado personalmente y que forma parte del componente backend de la solución propuesta.

Esta revisión exhaustiva permitirá evaluar las herramientas más apropiadas y los enfoques más efectivos para el desarrollo del visualizador de resultados. Además, se analizarán las ventajas y desventajas de cada opción, considerando las necesidades y requisitos específicos de la empresa TITSA.

2.1.1. Visualizadores de soluciones

En el ámbito de la visualización de resultados, existen diversas herramientas y tecnologías que permiten representar de manera efectiva y comprensible los resultados obtenidos de algoritmos y análisis. A continuación, se presentan algunos visualizadores de resultados utilizados en la industria y la investigación:

2.1.1.1. Power Bi

Power BI es una plataforma de visualización de datos desarrollada por Microsoft. Proporciona una amplia gama de capacidades para la creación de informes interactivos y paneles de control dinámicos. Con Power BI, los usuarios pueden conectarse a múltiples fuentes de datos, realizar transformaciones y análisis, y diseñar visualizaciones personalizadas. Además, ofrece capacidades de colaboración y compartición de informes en línea.

Figura 2: Logo de Power BI.

Ventajas	Desventajas
Permite crear objetos visuales con Python qué es el lenguaje utilizado por el departamento de Big Data de TITSA.	El programa requiere una curva de aprendizaje considerable sobre todo para personas que no tienen conocimientos informáticos.
Permite centralizar las visualizaciones en informes de Power Bi y compartirlos para los demás departamentos.	Se pierde modularidad debido a que la ejecución de los algoritmos siempre estaría ligada al contexto de la visualización de Power bi.
Permite realizar transformaciones de datos sobre los archivos necesarios desde la propia interfaz de la aplicación.	Es más costoso tanto en tiempo como en dinero que desarrollar un visualizador propio con las tecnologías utilizadas por el departamento (Python Dash).

Tabla 2.1.1.1. Ventajas y desventajas de Power Bi.

2.1.1.2. Tableau

Tableau es otra herramienta líder en visualización de datos. Ofrece una interfaz intuitiva y flexible para crear visualizaciones interactivas y dashboards personalizados. Tableau permite la conexión a diversas fuentes de datos y proporciona una amplia gama de opciones de visualización, desde gráficos básicos hasta mapas geoespaciales avanzados. También ofrece funcionalidades de análisis y exploración de datos, lo que permite descubrir patrones y tendencias ocultas en los resultados.

Figura 3: Logo de Tableau.

Ventajas	Desventajas
Integración con python.	Es costoso en cuanto a recursos de infraestructura si queremos alojarlo en nuestra organización.
Amplia comunidad y recursos.	Requiere un período de aprendizaje considerable y costoso en dinero.
Interactividad y exploración de datos.	Limitaciones con algunas bibliotecas de python.

Tabla 2.1.1.2. Ventajas y desventajas de Tableau.

2.1.1.3. Conclusiones sobre visualizadores comerciales.

Estos visualizadores de resultados son solo algunos ejemplos de las muchas herramientas disponibles en el mercado. Cada uno tiene sus propias fortalezas y debilidades, y la elección de la herramienta adecuada dependerá de los requisitos específicos del proyecto y las preferencias del usuario. Es importante evaluar cuidadosamente las características, la facilidad de uso y las capacidades de visualización de cada herramienta antes de seleccionar la más adecuada para el desarrollo del visualizador de resultados en el contexto de la empresa TITSA.

2.1.2. Algoritmo de conductores

El Bus Driver Scheduling Problem (BDSP) es un desafío de optimización combinatoria que implica asignar conductores de autobuses a vehículos con rutas predefinidas. El objetivo principal es optimizar los costos operativos y mejorar la calidad del trabajo de los empleados, teniendo en cuenta objetivos como minimizar el número de cambios de vehículo y maximizar la eficiencia de la asignación de conductores.

En el ámbito académico, se han realizado importantes contribuciones para abordar este problema. En el repositorio de la Universidad Pompeu Fabra [1], se encuentra disponible un documento que describe el modelado del problema de programación de conductores. Este documento proporciona una base teórica sólida y presenta en detalle los elementos clave del problema, como las restricciones y las variables a considerar en el proceso de asignación de conductores.

Además, se ha propuesto un nuevo modelo matemático y una solución aproximada utilizando el enfoque GRASP (Greedy Randomized Adaptive Search Procedure) para el problema de programación de conductores de autobuses. Este trabajo se presenta en el artículo Metaheuristic algorithms for the bus driver scheduling problem with complex break constraints [2].

Esta investigación ofrece una nueva perspectiva y un enfoque innovador para abordar el problema, brindando una solución aproximada eficiente basada en técnicas heurísticas y métodos de búsqueda adaptativos.

2.1.3. Trabajos de fin de grado sobre visualizadores de resultados.

La necesidad de visualizadores de soluciones en distintas empresas no es nada nuevo. En los últimos años, se han realizado diversos Trabajos de Fin de Grado en la Universidad de La Laguna que abordan la creación de visualizadores en diferentes ámbitos. Algunos ejemplos relevantes del año 2022 son los siguientes:

- Tratamiento y visualización de datos abiertos sobre la demografía en España [3].
- Interfaz para el modelado de meta-heurísticas [4].
- Procesamiento y visualización de datos abiertos sobre el empleo en Canarias [5].
- Despliegue y puesta en marcha de un servicio para la resolución de problemas de optimización [6].

En estos trabajos finales de grado podemos ver cómo se han analizado los visualizadores actuales del mercado y se han hecho propuestas personalizadas teniendo en cuenta los requisitos definidos por cada proyecto.

2.2. Arquitectura del proyecto

La escalabilidad del prototipo es una preocupación crucial que debe tenerse en cuenta. Para abordar esta cuestión, se han evaluado diversas arquitecturas y se ha decidido utilizar la arquitectura hexagonal. Esta elección se basa en su capacidad para garantizar la resiliencia del sistema frente a cambios y para facilitar la incorporación de nuevos algoritmos, asegurando así que la aplicación pueda crecer y adaptarse a las necesidades futuras de los usuarios.

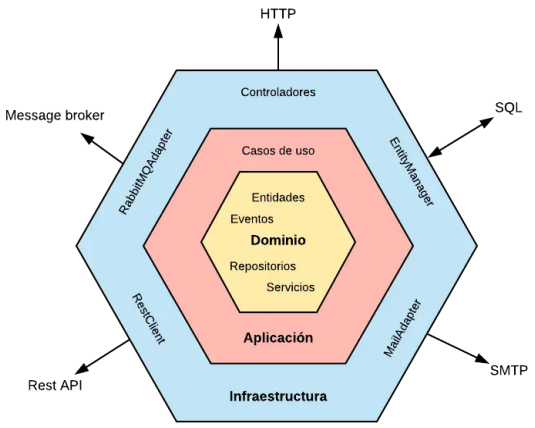


Figura 4: Arquitectura hexagonal.

2.2.1. Frontend

En el desarrollo del visualizador de resultados para la empresa TITSA, se ha optado por utilizar el lenguaje de programación Python y el framework Dash. Esta elección se basa en la ya existente utilización de Python y Dash en el departamento de Big Data de TITSA, lo que facilitará la integración y colaboración con otros proyectos existentes.

2.2.1.1. Python Dash

Figura 5: Logo de Python Dash.

Python Dash es un framework de código abierto que permite desarrollar rápidamente aplicaciones web interactivas y visualizaciones de datos utilizando Python. Con Dash, los desarrolladores pueden crear interfaces de usuario dinámicas y atractivas sin tener que preocuparse por el diseño y la implementación compleja del frontend. Basado en la biblioteca Flask y con el uso de componentes de React.js, Dash combina la flexibilidad y potencia de Python para el procesamiento de datos y la lógica del backend con una experiencia de usuario moderna y amigable en el frontend.

Una de las ventajas clave de Dash es su capacidad para crear visualizaciones interactivas de datos. Al combinar Dash con bibliotecas de visualización populares como Plotly, se puede generar una amplia gama de gráficos, tablas y otros elementos visuales interactivos. Esto hace que Dash sea especialmente adecuado para proyectos que requieren análisis y exploración de datos en tiempo real.

Dash ofrece una amplia variedad de componentes predefinidos, como botones, deslizadores, cuadros de texto y tablas, que facilitan la creación de interfaces de usuario personalizadas y funcionales. Además, se puede extender Dash con componentes personalizados y estilizarlos utilizando CSS para adaptarlos a los requisitos de diseño específicos del proyecto.

2.2.1.2. Plotly

Para las visualizaciones, se utilizará la biblioteca Plotly, que proporciona una amplia gama de gráficos interactivos y elegantes. Plotly ofrece capacidades de visualización altamente personalizables, lo que permitirá representar los resultados de manera efectiva y atractiva.

Figura 6: Logo de Plotly.

2.2.1.3. Geopandas

Además, se utilizará la biblioteca Geopandas para las visualizaciones geoespaciales. Geopandas es una extensión de la biblioteca Pandas que permite trabajar con datos geoespaciales, como mapas y geometrías. Esto será especialmente útil para representar información relacionada con las rutas y ubicaciones geográficas en el contexto del transporte en Tenerife.

Figura 7: Logo de GeoPandas.

2.2.1.4. Conclusiones sobre las tecnologías

El uso de Python, Dash, Plotly y Geopandas proporcionará una combinación poderosa y versátil para el desarrollo del visualizador de resultados. Estas tecnologías permitirán crear visualizaciones interactivas, personalizadas y compatibles con los datos específicos de la empresa TITSA.

Es importante destacar que la elección de estas herramientas se basa en su idoneidad para el proyecto y en su uso previo por parte del departamento de Big Data de TITSA. Esto garantizará una mayor eficiencia en el desarrollo y una integración fluida con los sistemas y flujos de trabajo existentes.

En resumen, el desarrollo del visualizador de resultados se llevará a cabo utilizando Python y el framework Dash, junto con las bibliotecas Plotly y Geopandas,

para garantizar visualizaciones efectivas y una representación adecuada de los datos relacionados con el transporte en Tenerife.

2.2.2. Backend

El backend, o lado del servidor, es una parte fundamental de cualquier aplicación web. Es responsable de procesar las solicitudes del usuario, realizar operaciones lógicas y acceder a la base de datos o a otros recursos necesarios para proporcionar una respuesta adecuada. En el contexto de nuestro proyecto de visualización de resultados, el backend jugará un papel crucial en la gestión de los algoritmos y en la comunicación con el frontend.

2.2.2.1. Flask

El framework Flask será utilizado para desarrollar el servidor API, que actuará como intermediario entre el frontend y el backend. Flask es una opción popular debido a su simplicidad y flexibilidad. Proporciona una forma sencilla de definir rutas y endpoints, gestionar las solicitudes HTTP y las respuestas correspondientes, y configurar la lógica de negocio necesaria.

Con Flask, podremos implementar las funcionalidades necesarias para recibir las solicitudes del frontend, procesarlas y enviar las respuestas correspondientes. Además, Flask permite la fácil integración de complementos y extensiones que pueden mejorar la funcionalidad del servidor API. Esto nos permitirá manejar las solicitudes de manera eficiente y proporcionar una experiencia fluida para los usuarios de la aplicación.

Figura 8: Logo de Flask.

2.2.2.2. Pandas

La biblioteca Python Pandas desempeñará un papel fundamental en la gestión de los algoritmos en el backend. Pandas es una biblioteca de manipulación y análisis de datos que proporciona estructuras de datos flexibles y eficientes, así como herramientas para realizar operaciones complejas en conjuntos de datos.

Utilizaremos Pandas para realizar operaciones de procesamiento de datos, como filtrado, agrupación y transformación de los resultados obtenidos de los algoritmos. Además, Pandas nos permitirá realizar cálculos estadísticos y realizar análisis avanzados de los datos generados por los algoritmos. Con su amplia funcionalidad, Pandas nos proporcionará las herramientas necesarias para gestionar y analizar los resultados de manera eficiente en el backend.

Figura 9: Logo de Pandas.

2.2.2.3. Conclusiones sobre las tecnologías

En conclusión, el backend será el encargado de gestionar los algoritmos y establecer una comunicación efectiva con el frontend. Flask, como framework de desarrollo web, nos permitirá crear el servidor API y gestionar las solicitudes y respuestas. Por su parte, Pandas, como biblioteca de manipulación y análisis de datos, nos proporcionará las herramientas necesarias para gestionar y analizar eficientemente los resultados de los algoritmos en el backend.

2.3. Manejo de versiones y módulos en python

El manejo de versiones en Python se refiere al control y seguimiento de las diferentes versiones de los paquetes y librerías utilizadas en un proyecto. Cada paquete puede tener múltiples versiones, y el uso de una versión específica puede ser crucial para garantizar la compatibilidad con el código existente y el correcto funcionamiento de la aplicación. Además, las actualizaciones de los paquetes pueden incluir correcciones de errores, mejoras de rendimiento o nuevas funcionalidades, por lo que es importante poder gestionar de forma adecuada las versiones utilizadas en un proyecto.

Para asegurar una gestión óptima de las versiones en nuestro proyecto, emplearemos dos herramientas fundamentales: pyenv, encargada de controlar las versiones de Python correspondientes, y pipenv, que nos permitirá administrar los paquetes y sus respectivas versiones.

Figura 10: Logo de Pyenv.

Adicionalmente, con el objetivo de simplificar el despliegue de nuestra aplicación, consideraremos la utilización de Docker. Esta potente herramienta permitirá a los desarrolladores implementar diversos servidores en contenedores, lo que facilitará significativamente el proceso de despliegue.

Figura 11: Logo de Docker.

Capítulo 3: Desarrollo de la aplicación

3.1. Desarrollando en Dash

Al desarrollar aplicaciones con Python Dash, es crucial comprender su funcionamiento, y para lograrlo, es fundamental leer la documentación disponible. En el caso específico de Python Dash, cuenta con una documentación muy completa que ofrece numerosos ejemplos y un foro donde los desarrolladores comparten sus conocimientos.

Durante el desarrollo de esta herramienta, fue necesario comprender varios conceptos fundamentales del framework, como los callbacks. En Python Dash, los callbacks se utilizan para gestionar las modificaciones realizadas en los diferentes componentes de la aplicación. Estos callbacks están constantemente monitoreando una entrada específica y, en caso de que esta cambie, se activa el callback correspondiente para ejecutar la lógica definida por el desarrollador.

Es importante tener en cuenta una particularidad de Python Dash: los callbacks se cargan al inicio de la aplicación y pueden surgir problemas si hacen referencia a identificadores que no existen. Afortunadamente, existen diferentes soluciones para abordar esta situación. Una de ellas consiste en activar el indicador "prevent_initial_call" en el callback, lo cual evita que se llame al inicio de la aplicación. Otra solución es utilizar "pattern-matching-callbacks", de modo que el callback solo se activa cuando se produce un cambio en un identificador que cumple cierto patrón.

Además resulta interesante destacar la funcionalidad de "dcc.Store", que se utiliza para almacenar información entre callbacks. Esto es recomendable en lugar de utilizar variables globales en la aplicación, ya que dcc.Store ofrece una forma más controlada y estructurada de gestionar el intercambio de datos entre los distintos callbacks.

Por último, para crear una herramienta multipágina con diferentes finalidades, es necesario investigar la funcionalidad de “multi-pages“ y cómo definir los layouts correspondientes.

3.2. Backend

En el desarrollo de nuestro proyecto, hemos implementado un backend que se encarga de centralizar las ejecuciones de los algoritmos y proporcionar una interfaz de programación de aplicaciones (API) para su posterior visualización en la interfaz de usuario. En esta sección, presentaremos el desarrollo de nuestro backend, destacando las tecnologías utilizadas y los diferentes endpoints que hemos creado.

Para implementar nuestra API, hemos optado por utilizar el framework Flask de Python. Flask es conocido por su facilidad de uso y su capacidad para desarrollar aplicaciones web rápidas y eficientes. Aprovechando la flexibilidad y las características de Flask, hemos creado una API bien estructurada que permite interactuar con nuestra aplicación de manera sencilla y segura.

Nuestra API consta de diferentes rutas y endpoints que ofrecen funcionalidades específicas. A continuación, detallaremos algunas de las principales rutas que hemos implementado:

- Ruta “**/available_algorithms**”: Tiene la responsabilidad de proporcionar información sobre los algoritmos actualmente disponibles en el servidor.
- Ruta “**/process_files**”: Lleva a cabo el procesamiento de los archivos proporcionados por el frontend. Esta ruta admite archivos con extensiones "csv" y en formato xlsx.
- Ruta “/**configuration_file**”: Se encarga de responder con un archivo de configuración que contiene los detalles de un algoritmo específico, basado en su nombre.
- Ruta “**/algorithm_performance**”: Esta ruta recibe en su cuerpo el nombre del algoritmo y los archivos asociados a él. Su objetivo es ejecutar el algoritmo correspondiente y devolver una lista de archivos en formato json que contienen los resultados obtenidos.
- En caso de intentar acceder a cualquier otra ruta se devolverá un código de estatus 501 con el mensaje: “Not Implemented”.

```python
@app.errorhandler(404)
def page_not_found(error):
    return default()

app.add_url_rule("/process_files", "available_algorithms", available_algorithms, methods=["GET"])
app.add_url_rule("/description", "description", description, methods=["GET"])
app.add_url_rule("/process_files", "process_files", process_files, methods=["POST"])
app.add_url_rule("/configuration_file", "configuration_file", configuration_file, methods=["GET"])
app.add_url_rule("/algorithm_performance", "algorithm_performance", algorithm_performance,methods=["POST"])
```

Figura 12: Rutas definidas en el backend.

Ahora, profundizaremos en la arquitectura de nuestro backend, que se basa en la implementación de la arquitectura hexagonal, también conocida como la arquitectura de puertos y adaptadores. Esta arquitectura nos permite lograr una separación clara y eficiente de las diferentes capas de nuestra aplicación, facilitando la modularidad, el mantenimiento y la escalabilidad.

3.2.1. Arquitectura

3.2.1.1. Dominio

En la capa principal de nuestro dominio, se encuentran las entidades y servicios fundamentales que definen las reglas de negocio y comportamientos esenciales de nuestra aplicación.

En nuestro proyecto, hemos desarrollado una clase llamada "Algorithm" que desempeña un papel crucial en la gestión de la ejecución de los algoritmos y el formateo de los resultados que nuestra aplicación debe devolver. Esta clase se encarga de coordinar y ejecutar los algoritmos específicos, así como de agregar información relevante, como la hora de ejecución y el archivo del que proviene la ejecución del algoritmo, a los diferentes marcos de datos generados.

Además de la clase "Algorithm", hemos creado ficheros de configuración específicos para cada algoritmo definido en nuestra aplicación. Estos ficheros contienen detalles generales que los marcos de datos de cada algoritmo deben cumplir. Al definir estas configuraciones, aseguramos la coherencia y consistencia en la estructura y contenido de los marcos de datos generados por cada algoritmo.

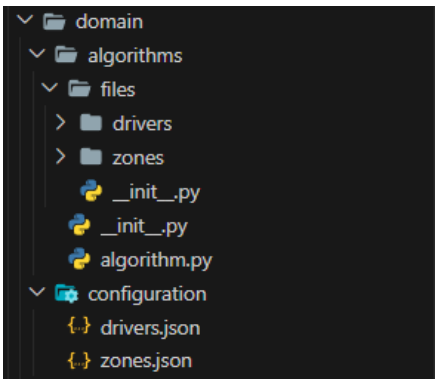


Figura 13: Dominio del backend.

3.2.1.2. Aplicación

En la capa de aplicación, se encuentran los casos de uso que representan la lógica de negocio fundamental de nuestra aplicación. En el contexto de nuestro servidor, hemos definido casos de uso específicos que nos permiten obtener la información necesaria sobre un algoritmo y, posteriormente, ejecutarlo utilizando los archivos correspondientes.

Estos casos de uso son responsables de coordinar las diferentes acciones y operaciones necesarias para obtener los datos requeridos para la ejecución de un algoritmo. Esto puede implicar la interacción con otras capas, como la capa de dominio, donde se definen las reglas de negocio y se gestionan las entidades y servicios principales.

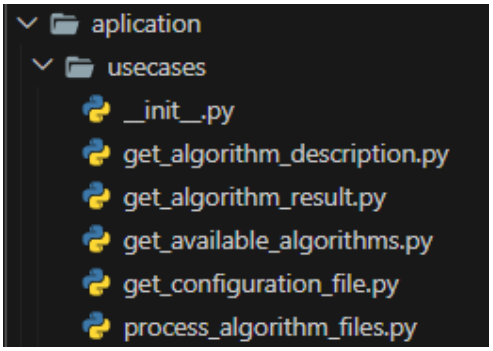


Figura 14: Casos de uso del backend.

Esta capa de aplicación es esencial para garantizar que la lógica de negocio de nuestra aplicación se ejecute de manera coherente y eficiente. Los casos de uso nos permiten separar las preocupaciones y organizar de manera clara y estructurada las diferentes acciones que deben llevarse a cabo para lograr los objetivos deseados.

3.2.1.3. Infraestructura

En esta capa, nos enfocamos en la configuración e implementación del marco de trabajo seleccionado, que en nuestro caso es Python Flask. Definimos el servidor, estableciendo parámetros como el puerto de escucha, la dirección de host y cualquier otra configuración relevante. Además, definimos las rutas que serán accesibles desde el frontend o desde otras partes de la aplicación.

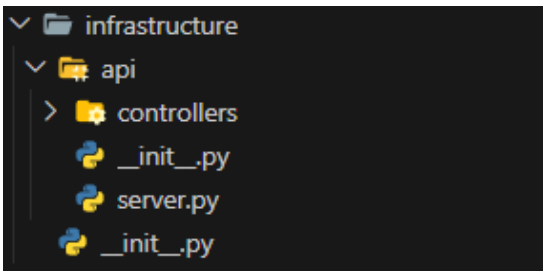


Figura 15: Contenido de la capa infraestructura.

Esta capa nos permite también definir distintos adaptadores si añadimos alguna base de datos o de dependencias externas, dichos adaptadores tendrán contratos que deben de cumplir en la capa de dominio, en el caso de python, esto se logra mediante el uso de clases abstractas.

3.2.2. Pruebas y validaciones

En el ámbito del desarrollo de software, es fundamental garantizar la calidad y confiabilidad de las aplicaciones que creamos. Una metodología que ha demostrado ser efectiva en este sentido es el desarrollo dirigido por pruebas (Test-Driven Development, TDD). En el marco del TDD, el enfoque principal se centra en escribir pruebas automatizadas antes de implementar el código de producción.

3.2.2.1. Desarrollo dirigido por pruebas

El desarrollo dirigido por pruebas (Test-Driven Development, TDD) es una metodología de desarrollo de software que se centra en escribir pruebas automatizadas antes de implementar el código de producción. El objetivo principal del TDD es mejorar la calidad del software y fomentar la refactorización continua a medida que se agregan nuevas funcionalidades.

Existen dos enfoques principales en el desarrollo dirigido por pruebas: el enfoque Inside-out y el enfoque Outside-in. Estos enfoques difieren en la forma en que se escriben las pruebas y se desarrolla el código.

1. **Inside-out:** Se basa en escribir pruebas unitarias desde el dominio de la aplicación hasta la infraestructura.
2. **Outside-in:** Al contrario del enfoque inside-out, se basa en escribir pruebas de extremo a extremo, probando el controlador de infraestructura de nuestra aplicación y llegar hasta el dominio de nuestra aplicación.

El servidor backend fue desarrollado desde un enfoque inside-out donde se probaron los servicios y luego los controladores de nuestra aplicación ya que el dominio es un poco más abstracto ya que depende de los algoritmos que existan en el mismo.

3.2.2.2.Tipos de pruebas

Pruebas Unitarias	Pruebas de integración	Pruebas de extremo a extremo
Son una técnica utilizada para probar el funcionamiento correcto de los objetos unitarios de nuestro dominio.	Son aquellas que comprueban el funcionamiento entre varias piezas de la aplicación y cómo se relacionan entre ellas,	Son aquellas que comprueban el funcionamiento desde extremo a extremo de nuestros controladores.

Tabla 3.2.2.2: Tipos de prueba en el backend.

En cuanto a las pruebas unitarias y la de integración hay diferentes debates que apuntan a la existencia de pruebas llamadas “pruebas sociales”, esto se debe a que si las pruebas no tienen factores externos como bases de datos, las pruebas unitarias “sociales” son las que prueban los casos de uso de tu aplicación y las de integración son aquellas que prueban los controladores.

Debido a esto, se ha optado por crear carpetas de pruebas donde las unitarias son referidas a los casos de uso de nuestra capa de aplicación y las de integración son referidas a los controladores.

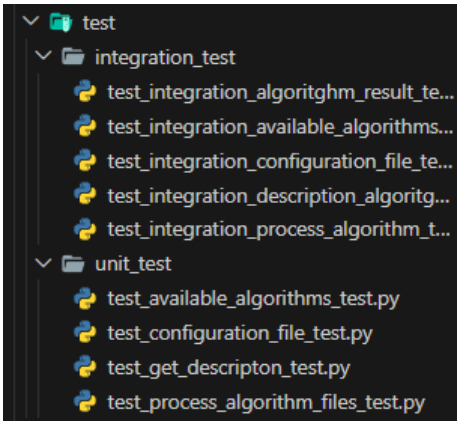


Figura 16: Carpeta de pruebas del backend.

3.2.2.3. Falseado de comportamientos

Un aspecto importante en el ámbito de las pruebas es la utilización de los llamados "dobles de prueba", que consisten en simular comportamientos del sistema para probar casos que no son posibles de verificar debido a diversos factores. Estos dobles de prueba son ampliamente utilizados para probar nuestra aplicación de manera aislada cuando existen dependencias externas.

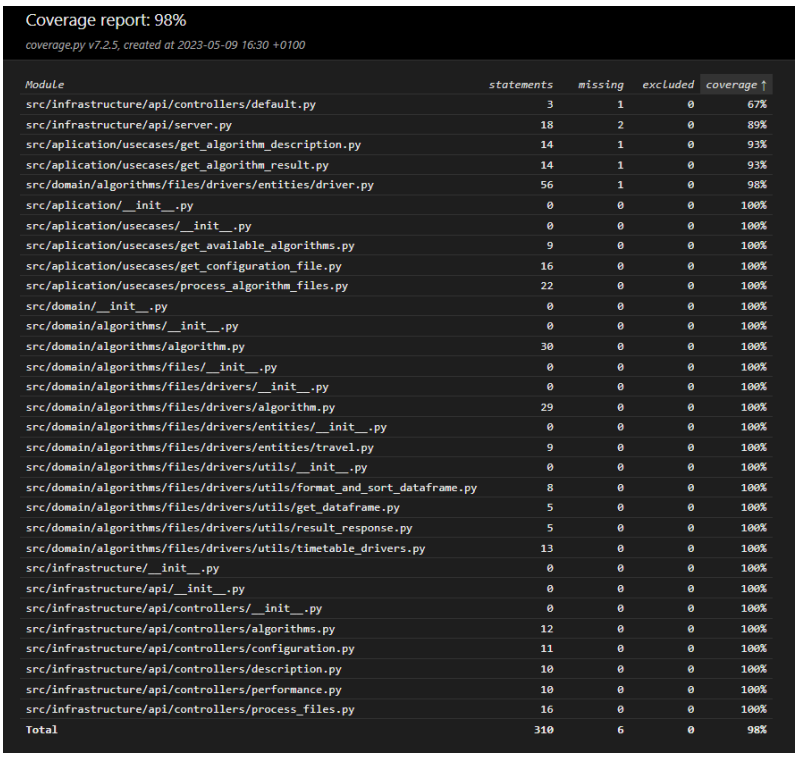
En nuestro proyecto, hemos empleado diversos dobles de prueba para poner a prueba casos que no se pueden comprobar con el estado actual del sistema. Un ejemplo concreto es simular que el servidor no tenga algoritmos disponibles, lo cual nos permite evaluar cómo se comporta la aplicación en ese escenario particular.

```python
def test_not_found_available_algorithms_from_client(self):
    with app.test_client() as client:
        with patch(
            "aplication.usecases.get_available_algorithms.get_available_algorithms",
            return_value=[],
        ):
            response = client.get("/available_algorithms")
            assert response.status_code == 404
```

Figura 17: Ejemplo de “doble de prueba” en el backend.

Finalmente, al aplicar todas estas técnicas y utilizar una herramienta que evalúa el cubrimiento de pruebas, podemos determinar los flujos de ejecución que han sido cubiertos en nuestra aplicación. En el caso específico de Python, podemos utilizar una herramienta llamada Python Coverage, que nos permite medir el cubrimiento de nuestro código y generar informes en formato HTML que muestran de manera visual qué partes del código han sido ejecutadas durante las pruebas.

Python Coverage nos brinda una visión detallada de la cobertura de línea, mostrando qué líneas de código han sido ejecutadas y cuáles no durante las pruebas. Esto nos permite identificar rápidamente las áreas de nuestro código que necesitan una mayor atención en términos de pruebas adicionales o mejoras.



Coverage report: 98%

coverage.py v7.2.5, created at 2023-05-09 16:30 +0100

Module	statements	missing	excluded	coverage ↑
src/infrastructure/api/controllers/default.py	3	1	0	67%
src/infrastructure/api/server.py	18	2	0	89%
src/aplication/usecases/get_algorithm_description.py	14	1	0	93%
src/aplication/usecases/get_algorithm_result.py	14	1	0	93%
src/domain/algorithms/files/drivers/entities/driver.py	56	1	0	98%
src/aplication/__init__.py	0	0	0	100%
src/aplication/usecases/__init__.py	0	0	0	100%
src/aplication/usecases/get_available_algorithms.py	9	0	0	100%
src/aplication/usecases/get_configuration_file.py	16	0	0	100%
src/aplication/usecases/process_algorithm_files.py	22	0	0	100%
src/domain/__init__.py	0	0	0	100%
src/domain/algorithms/__init__.py	0	0	0	100%
src/domain/algorithms/algorithm.py	30	0	0	100%
src/domain/algorithms/files/__init__.py	0	0	0	100%
src/domain/algorithms/files/drivers/__init__.py	0	0	0	100%
src/domain/algorithms/files/drivers/algorithm.py	29	0	0	100%
src/domain/algorithms/files/drivers/entities/__init__.py	0	0	0	100%
src/domain/algorithms/files/drivers/entities/travel.py	9	0	0	100%
src/domain/algorithms/files/drivers/utils/__init__.py	0	0	0	100%
src/domain/algorithms/files/drivers/utils/format_and_sort_dataframe.py	8	0	0	100%
src/domain/algorithms/files/drivers/utils/get_dataframe.py	5	0	0	100%
src/domain/algorithms/files/drivers/utils/result_response.py	5	0	0	100%
src/domain/algorithms/files/drivers/utils/timetable_drivers.py	13	0	0	100%
src/infrastructure/__init__.py	0	0	0	100%
src/infrastructure/api/__init__.py	0	0	0	100%
src/infrastructure/api/controllers/__init__.py	0	0	0	100%
src/infrastructure/api/controllers/algorithms.py	12	0	0	100%
src/infrastructure/api/controllers/configuration.py	11	0	0	100%
src/infrastructure/api/controllers/description.py	10	0	0	100%
src/infrastructure/api/controllers/performance.py	10	0	0	100%
src/infrastructure/api/controllers/process_files.py	16	0	0	100%
Total	310	6	0	98%

Figura 18: Informe html del cubrimiento de pruebas.

3.3. Frontend

Hemos utilizado Python Dash para desarrollar el frontend de nuestro proyecto, aprovechando sus capacidades para crear una interfaz de usuario dinámica y visualmente atractiva. Esta elección nos ha permitido centralizar y presentar las visualizaciones de los algoritmos de optimización ejecutados en el backend, alineándose con las necesidades del departamento de Big Data de Titsa.

En esta sección, profundizaremos en el desarrollo del frontend, destacando las funcionalidades clave implementadas y las decisiones estratégicas adoptadas para mejorar la experiencia del usuario. Analizaremos los componentes y visualizaciones que hemos creado, así como su interacción con el backend para obtener los datos requeridos. Todo ello, en consonancia con la arquitectura hexagonal que hemos empleado para facilitar la incorporación de nuevas visualizaciones.

3.3.1. Arquitectura

3.3.1.1. Dominio

El principal objetivo de nuestro servidor frontend es visualizar los resultados de los algoritmos. No obstante, también se busca que proporcione retroalimentación constante sobre los requisitos necesarios para llamar a dichos algoritmos. Para lograr esto, el servidor realizará solicitudes al backend para obtener información sobre el archivo de configuración del algoritmo que se va a ejecutar. Además, se encargará de validar que los archivos cargados cumplan con el formato requerido.

En lo que respecta a las visualizaciones estáticas, por defecto se mostrarán tablas que reflejen los resultados y marcos de trabajo correspondientes a la ejecución del algoritmo. No obstante, es posible personalizar estas visualizaciones estáticas de acuerdo con cada algoritmo en particular. Esta personalización se logra al incluir el comportamiento deseado en un fichero con el nombre del algoritmo en la carpeta "visualizer", lo cual permitirá adaptar la presentación de los resultados a las características específicas de cada algoritmo.

Asimismo, también es posible aplicar restricciones propias del algoritmo añadiendo las comprobaciones deseadas a la carpeta “check” con el nombre del algoritmo que se quiera comprobar .

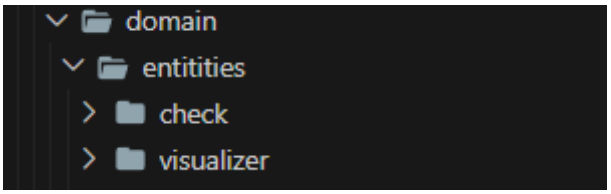


Figura 19: Dominio del frontend.

3.3.1.2. Aplicación

En la capa de aplicación del frontend, se encuentran los casos de uso encargados de establecer la lógica para seleccionar las comprobaciones y visualizaciones requeridas, basándose en el nombre del algoritmo seleccionado por el usuario. Asimismo, se incluirán funciones auxiliares que serán de utilidad para el correcto funcionamiento de la lógica de nuestra aplicación.

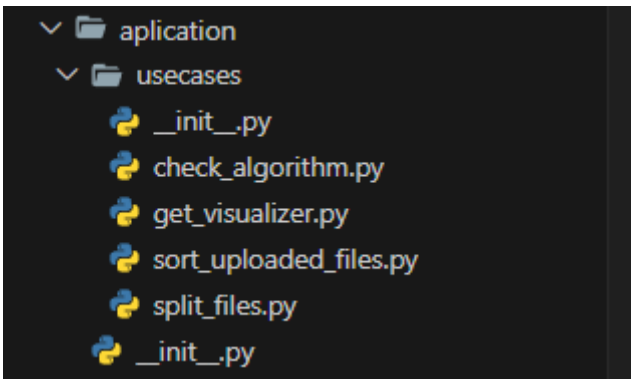


Figura 20: Capa de aplicación del frontend.

En la Figura 20 se representa gráficamente la capa de aplicación del frontend, destacando su importancia dentro de la arquitectura hexagonal. Es fundamental tener presente que esta arquitectura fomenta la separación de preocupaciones y la independencia de las capas. Por lo tanto, en esta capa de aplicación nos enfocaremos únicamente en definir la lógica necesaria para nuestras visualizaciones y comprobaciones, siguiendo los principios de la arquitectura hexagonal.

3.3.1.3. Infraestructura

En la capa de infraestructura de nuestro servidor frontend, nos ocuparemos de todos los aspectos relacionados con el framework utilizado, en este caso, Python Dash. Esta capa tiene la responsabilidad de definir las páginas y componentes de nuestra aplicación, así como manejar los distintos "callbacks" tanto para los componentes como para las visualizaciones dinámicas.

Además, en esta capa también se establecerán los estilos de nuestra interfaz mediante el uso de CSS. Utilizaremos diferentes técnicas, como el uso de unidades relativas, para garantizar la responsividad del sitio web. Esto nos permitirá adaptar la apariencia y el diseño de la aplicación a diferentes dispositivos y tamaños de pantalla.

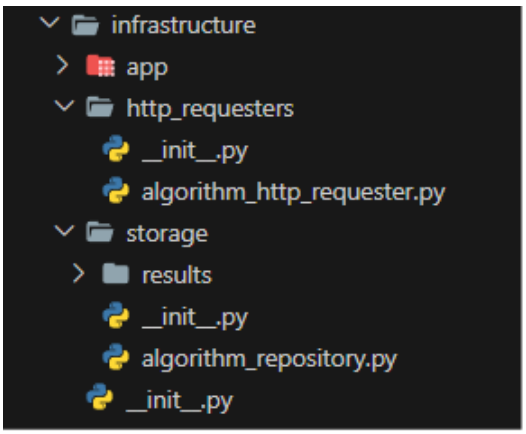


Figura 21: Capa de infraestructura del frontend.

Por último, definiremos la conexión con nuestro backend a través de una clase HTTP requester y el almacenamiento en local de nuestros resultados de la ejecución de los algoritmos mediante el patrón Repository.

En resumen, la capa de infraestructura del servidor frontend se enfoca en la implementación técnica de la aplicación, definiendo la estructura, las conexiones con el exterior, el almacenado de los resultados, los componentes y los estilos necesarios para ofrecer una interfaz funcional y atractiva a los usuarios

3.3.2. Flujo de conexión con el backend

Una vez que hemos configurado nuestros servidores frontend y backend, es importante establecer un flujo de comunicación claro para cada ruta a la que accederemos.

El primer paso en este flujo es realizar una consulta desde el servidor frontend para determinar los algoritmos disponibles en la aplicación. Esta información se mostrará en un menú desplegable (dropdown) para que el usuario pueda seleccionar el algoritmo deseado.

Una vez que el usuario ha seleccionado un algoritmo, el servidor frontend accederá a una ruta específica en el servidor backend para obtener la descripción detallada de dicho algoritmo. Esta descripción proporcionará al usuario información relevante sobre su funcionamiento y requisitos.

A continuación, el servidor frontend solicitará al servidor backend el archivo de configuración del algoritmo seleccionado. Este archivo contiene instrucciones

precisas sobre el formato y los detalles de los archivos que el usuario debe subir para ejecutar el algoritmo correctamente.

Una vez que el usuario ha cargado los archivos requeridos según las indicaciones del archivo de configuración, el servidor frontend permitirá al usuario iniciar la ejecución del algoritmo haciendo clic en un botón designado. Este botón desencadenará la ejecución del algoritmo en el servidor backend y mostrará la visualización definida específicamente para ese algoritmo. De esta manera, el usuario podrá visualizar los resultados de manera clara y comprensible.

En resumen, el flujo de comunicación entre los servidores frontend y backend implica la obtención de los algoritmos disponibles, la descripción del algoritmo seleccionado, el archivo de configuración y la ejecución del algoritmo con la visualización correspondiente.

3.4. Código Sostenible

En nuestro proyecto, hemos prestado especial atención a garantizar que nuestro código sea sostenible a largo plazo. Para lograr esto, hemos aplicado diversas metodologías y buenas prácticas que promueven la legibilidad, la mantenibilidad y la escalabilidad del código. En esta sección, destacaremos algunas de las estrategias que hemos seguido:

1. **Nombramiento Significativo de Variables:** Hemos adoptado la práctica de utilizar nombres de variables descriptivos y significativos en todo nuestro código. Esto nos permite comprender fácilmente el propósito y el contexto de cada variable, lo que facilita la lectura y el mantenimiento del código. Además, hemos evitado el uso de nombres ambiguos o abreviaturas confusas, favoreciendo la claridad y la comprensión para cualquier persona que trabaje en el proyecto.

2. **Modularidad:** Hemos estructurado nuestro código en módulos lógicos y coherentes, siguiendo el principio de responsabilidad única. Cada módulo se enfoca en una funcionalidad específica y se encarga de realizar tareas bien definidas. Esto nos permite tener un código más modular y desacoplado, lo

que facilita la reutilización, la prueba y los futuros cambios o mejoras en el proyecto.

3. **Enfoque de Inside-Out en el Desarrollo Dirigido por Pruebas:** Hemos adoptado el enfoque de desarrollo dirigido por pruebas (Test-Driven Development, TDD) para garantizar la calidad y la robustez de nuestro código. Con este enfoque, primero escribimos las pruebas unitarias
4. **Gestión de Dependencias y Control de Versiones:** Hemos utilizado herramientas y técnicas para gestionar de manera efectiva las dependencias del proyecto y controlar las versiones del código. Hemos utilizado administradores de paquetes como pipenv para gestionar las dependencias externas y garantizar la consistencia entre los entornos de desarrollo y producción. Asimismo, hemos utilizado sistemas de control de versiones como Git y plataformas como GitHub para realizar un seguimiento de los cambios.

En conjunto, estas estrategias y prácticas han contribuido a crear un código sostenible, permitiéndonos mantener y escalar nuestro proyecto de manera eficiente a medida que evoluciona con el tiempo. Al seguir estas metodologías, hemos asegurado la calidad, la eficiencia y la confiabilidad de nuestro código, proporcionando una base sólida para el desarrollo continuo y el éxito de nuestro proyecto.

3.5. Dockerizando la solución.

Docker se ha convertido en una herramienta fundamental para desarrolladores y administradores de sistemas, ya que permite crear, desplegar y ejecutar aplicaciones de manera eficiente y reproducible en entornos aislados. En el contexto de Python, Docker ofrece numerosas ventajas y simplifica el proceso de configuración y gestión de entornos de desarrollo.

3.5.1. Entornos virtuales en Docker

Aunque los entornos virtuales son una práctica común para aislar y gestionar las dependencias en entornos de desarrollo Python tradicionales, no se recomienda utilizarlos directamente en máquinas Docker.

La razón principal es que Docker proporciona un enfoque más completo y eficiente para la gestión de entornos a través de la creación de imágenes. En lugar de crear un entorno virtual dentro de un contenedor Docker, se puede crear una imagen Docker específica para cada aplicación o proyecto. Esto asegura que todas las dependencias, bibliotecas y configuraciones necesarias estén encapsuladas en la imagen, lo que facilita la replicación y el despliegue consistente en diferentes entornos.

En nuestro proyecto, hemos adoptado una práctica eficiente utilizando la funcionalidad multistage de Docker para instalar las dependencias del Pipfile.lock en nuestros contenedores. Esta elección se basa en el uso de Pipenv, lo que nos permite prescindir de la necesidad de utilizar entornos virtuales dentro de nuestras máquinas Docker.

3.5.2. Docker en cada servidor

Una de las principales razones para dockerizar servidores por separado, como el frontend y el backend, es la escalabilidad. Docker ofrece la capacidad de escalar horizontalmente cada componente de una aplicación de manera independiente y eficiente. Al separar el frontend y el backend en contenedores Docker individuales, podemos ajustar y escalar cada uno según sus necesidades específicas sin afectar a los demás componentes. Esto nos brinda flexibilidad para asignar recursos de manera óptima y garantizar un rendimiento óptimo en función de la carga y la demanda de cada parte de la aplicación.

3.5.3. Docker compose

Docker Compose es una herramienta que facilita la orquestación y la gestión de múltiples contenedores Docker. Con Compose, podemos definir y configurar fácilmente todos los servicios y contenedores necesarios para nuestra aplicación en un archivo YAML. Esto nos permite especificar la configuración de cada contenedor, las dependencias entre ellos y otros detalles relevantes. Docker Compose simplifica la administración de múltiples contenedores y nos brinda una forma sencilla de gestionar todo el entorno de nuestra aplicación.

La única consideración que tenemos que tener es a la url de nuestro backend a la hora de definir nuestro docker-compose ya que dependerá de la red que definamos por lo que nos conectaremos a dicho backend mediante una url establecida por una variable de entorno en el HTTP Requester correspondiente.

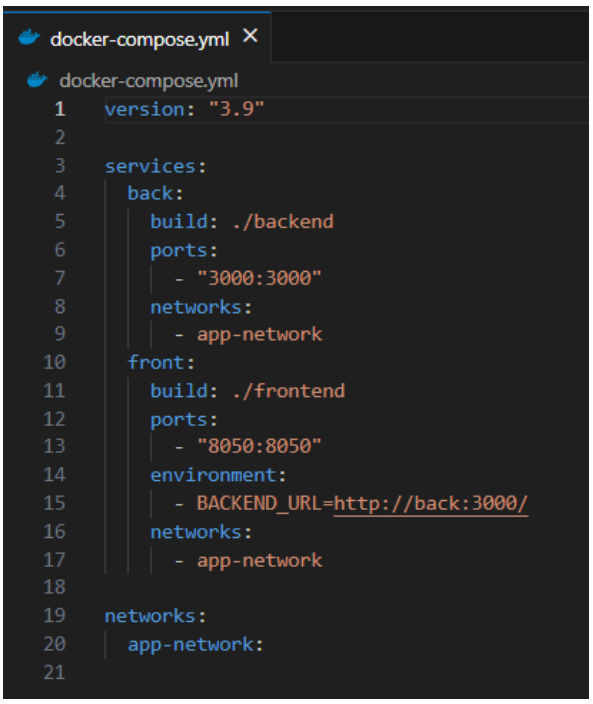


Figura 22: Archivo docker-compose.yml.

Capítulo 4: Arquitectura hexagonal

4.3. Introducción

La arquitectura hexagonal, fue propuesta por Alistair Cockburn como un enfoque para diseñar sistemas de software que sean flexibles, mantenibles y orientados a los dominios del negocio. Se basa en el principio de separación de intereses y busca establecer una clara división entre las reglas de negocio de una aplicación y los detalles de implementación técnicos.

El concepto central de la arquitectura hexagonal es la separación por capas de nuestro sistema. La capa más interna representa el núcleo de la aplicación, donde residen las reglas de negocio y la lógica principal. La capa más externa representan los puertos de entrada y salida, a través de los cuales el sistema se comunica con el mundo exterior, como interfaces de usuario, servicios externos, bases de datos, etc. En la arquitectura hexagonal podemos encontrar 3 capas:

1. Capa de Dominio: Representa la lógica central de la aplicación, independiente de cualquier detalle de implementación. Aquí se encuentran los modelos de dominio, reglas de negocio y servicios que definen el comportamiento fundamental de la aplicación. Es la capa más interna y se enfoca en la funcionalidad central y los conceptos clave del negocio.
2. Capa de Aplicación: En esta capa se encuentran los casos de uso o servicios de aplicación. Se encarga de coordinar la interacción entre la capa de dominio y los adaptadores externos, gestionando las solicitudes y respuestas. Aquí se define la lógica para seleccionar las comprobaciones y visualizaciones necesarias basadas en las acciones del usuario. También pueden existir funciones auxiliares para el funcionamiento de la lógica de la aplicación.
3. Capa de Infraestructura: En esta capa se definen adaptadores que actúan como conectores entre el núcleo de la aplicación y los puertos externos. Estos se encargan de traducir las solicitudes y respuestas entre el formato utilizado por el núcleo y el formato requerido por los puertos externos, permitiendo que el núcleo de la aplicación sea independiente de los detalles

de implementación de los puertos externos, lo que facilita la modularidad y la reutilización del código.

La arquitectura hexagonal, también es conocida como arquitectura de puertos y adaptadores, esto se debe a que el nombre de arquitectura hexagonal ha generado diversas interpretaciones y discusiones a lo largo del tiempo.

En este sentido, muchos defienden la idea de llamarla arquitectura de puertos y adaptadores para enfatizar que en la capa de presentación se encuentran los puertos, mientras que en la capa de infraestructura se desarrollan los adaptadores encargados de obtener los datos de los puertos y adaptarlos a la lógica de negocio de la aplicación. Esta perspectiva busca clarificar la separación de responsabilidades y el flujo de datos en el diseño del sistema.

4.3.1. Ventajas

Una de las principales ventajas de la arquitectura hexagonal es su capacidad para facilitar las pruebas automatizadas. Al aislar el núcleo de la aplicación de los detalles de implementación externos, se puede probar la lógica de negocio de manera aislada sin depender de componentes externos, lo que facilita la creación de pruebas unitarias y la adopción de prácticas de desarrollo impulsadas por pruebas (TDD).

Además, la arquitectura hexagonal promueve la flexibilidad y la evolución del sistema a lo largo del tiempo. Al separar claramente las responsabilidades y las capas del sistema, es más fácil realizar cambios y actualizaciones en una parte específica sin afectar otras áreas. Esto facilita la adaptación del sistema a nuevos requisitos o tecnologías, sin comprometer la estabilidad y la integridad de la lógica de negocio.

Otra ventaja importante de la arquitectura hexagonal es su capacidad para facilitar la integración con sistemas externos. Al utilizar adaptadores en la capa de infraestructura, se pueden implementar fácilmente mecanismos de comunicación con servicios externos, como bases de datos, APIs o sistemas legados. Esto permite una mayor flexibilidad para interactuar con diferentes tecnologías y sistemas, sin comprometer la coherencia del núcleo de la aplicación.

Además, la arquitectura hexagonal fomenta la reutilización de componentes. Al separar claramente la lógica de negocio en la capa de dominio, esta puede ser utilizada por diferentes interfaces de usuario o canales de entrada/salida, sin necesidad de duplicar la lógica en cada adaptador. Esto conduce a un código más modular y mantenible, así como a una reducción de la duplicación de código y la complejidad innecesaria.

Por último, la arquitectura hexagonal mejora la comprensión del sistema. Al separar las responsabilidades en capas bien definidas, se facilita la comprensión y el razonamiento sobre el sistema en su conjunto. Cada capa tiene una responsabilidad clara y se puede entender de manera aislada, lo que simplifica el proceso de desarrollo, depuración y mantenimiento.

4.3.2. Desventajas

La arquitectura hexagonal, a pesar de sus numerosas ventajas, también presenta algunas desventajas que deben tenerse en cuenta al considerar su implementación.

Una de las desventajas es la complejidad inicial que implica. La separación de responsabilidades en capas y la definición de interfaces pueden resultar complicadas para equipos sin experiencia en este enfoque arquitectónico.

Además, el uso de la arquitectura hexagonal puede aumentar la complejidad general del código. La introducción de capas adicionales y abstracciones puede hacer que el código sea más difícil de entender y mantener, especialmente si no se sigue una estructura clara.

Otra desventaja es la posible sobrecarga en el rendimiento. La necesidad de adaptar y convertir los datos entre diferentes capas y sistemas puede generar una pérdida de rendimiento y un mayor uso de recursos.

Además, la arquitectura hexagonal puede implicar una mayor cantidad de código debido a la creación de capas y componentes adicionales. Esto puede aumentar la complejidad del proyecto y dificultar su mantenimiento.

Finalmente, la adopción de la arquitectura hexagonal puede requerir un tiempo de aprendizaje y adaptación por parte del equipo. Esto puede generar una curva de aprendizaje inicial y retrasar el desarrollo del proyecto.

Es importante evaluar cuidadosamente estas desventajas en relación con las necesidades y requisitos del proyecto antes de decidir implementar la arquitectura hexagonal. Cada proyecto tiene sus propias características y lo que funciona bien en un caso puede no ser adecuado en otro.

4.4. Vertical Slicing

El vertical slicing, también conocido como "corte vertical", es un enfoque de desarrollo de software que se ha vuelto popular en los últimos años. Surge como una alternativa al enfoque tradicional de desarrollo basado en capas horizontales.

En el enfoque de la arquitectura hexagonal, el desarrollo se organiza en torno a capas horizontales, como la capa de infraestructura, la capa de aplicación y la capa de dominio. Cada capa se desarrolla de manera independiente, lo que puede llevar a una fragmentación del código y dificultades en la integración y la entrega de valor al usuario final.

El vertical slicing aborda este problema al enfocarse en el flujo de trabajo funcional de la aplicación. En lugar de desarrollar por capas, se desarrolla una funcionalidad completa desde la capa de infraestructura hasta la capa de dominio, a lo largo de todo el stack tecnológico.

La principal ventaja del vertical slicing es que permite entregar valor al usuario de manera más rápida y continua. Al desarrollar una funcionalidad completa, se pueden obtener retroalimentación y resultados tangibles de manera temprana, lo que facilita la validación del enfoque y la detección temprana de posibles problemas.

Además, el vertical slicing promueve la colaboración y la comunicación efectiva entre los diferentes roles del equipo de desarrollo. Al trabajar en una funcionalidad completa, los desarrolladores, los diseñadores y los especialistas en experiencia de usuario pueden colaborar estrechamente y comprender mejor los requisitos y las necesidades del usuario.

Otra ventaja del vertical slicing es que facilita la evolución y el mantenimiento del software. Al tener una funcionalidad completa, los cambios y las actualizaciones se pueden realizar de manera más enfocada y se reduce la posibilidad de introducir problemas en otras partes del sistema.

Sin embargo, es importante tener en cuenta que el vertical slicing puede requerir un mayor nivel de coordinación y planificación, ya que las funcionalidades se desarrollan de manera integral. Además, el diseño y la arquitectura del software deben ser adecuados para soportar este enfoque y evitar la creación de dependencias no deseadas entre las funcionalidades.

4.5. Clean architecture

El concepto de Clean Architecture, también conocida como Arquitectura Limpia, surge como una propuesta de diseño de software que busca separar las preocupaciones y responsabilidades de las diferentes capas de una aplicación. Fue propuesto por el reconocido desarrollador de software Robert C. Martin (también conocido como Uncle Bob) y se basa en los principios del diseño SOLID y la arquitectura hexagonal.

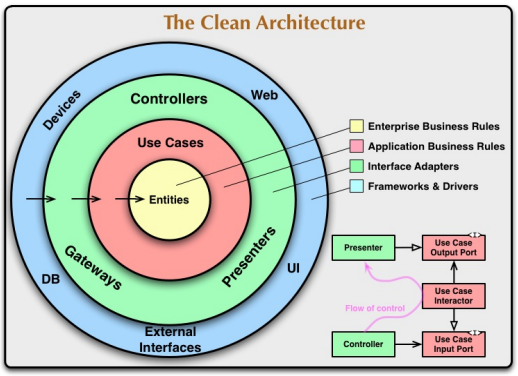


Figura 23: Clean architecture.

Tiene como objetivo principal lograr un código fuente limpio, modular, mantenible y testable. Para ello, establece una estructura clara y jerárquica de capas, donde las capas internas contienen las políticas y reglas de negocio, mientras que las capas externas se encargan de los detalles de implementación y la interacción con el mundo exterior.

Este nuevo enfoque establece que la comunicación entre capas debe ser limitada, esto se logra al proponer una separación estricta de responsabilidades y

una clara dirección de dependencias, utilizando principios como la inversión e inyección de dependencias, permitiendo que las capas internas, que contienen la lógica de negocio, sean independientes de los detalles de implementación externos, lo que facilita su comprensión, prueba y modificación. Los niveles de los que se compone Clean Architecture son los siguientes:

- **UI**: Interfaz de usuario.
- **Presenters**: Clases que se suscriben a los eventos generados por la interfaz y que responden en consecuencia representado la información en la interfaz de forma gráfica.
- **Use Cases**: Reglas de negocio y tomas de decisiones.
- **Entities**: Modelo de datos de la aplicación, comunicación con servicios web, caché de datos.

En resumen, Clean Architecture surge como una propuesta de diseño de software que busca superar las limitaciones de otras arquitecturas más tradicionales, promoviendo la separación de responsabilidades, la independencia de los detalles de implementación y la modularidad del código. Su objetivo es lograr un código fuente limpio, mantenible y testable, facilitando la evolución y el mantenimiento a largo plazo del software.

4.6. Diseño orientado a recursos

El diseño orientado a recursos de Google es una metodología arquitectónica que se utiliza en el desarrollo de aplicaciones en Google. Este enfoque se basa en los principios de la arquitectura REST y está diseñado para crear sistemas escalables, flexibles y de alto rendimiento.

Los recursos son considerados como entidades fundamentales de la aplicación y se acceden a través de identificadores únicos (URLs). Estos recursos pueden ser cualquier entidad o concepto significativo en el dominio del problema, como usuarios, documentos, imágenes, etc.

Aprovechando este enfoque, con la separación de capas de la arquitectura hexagonal y el vertical slicing, podemos crear una arquitectura modular y fácilmente escalable.

Capítulo 6: Resultado Final.

El resultado final se puede dividir en dos partes: en primer lugar, se encuentra el servidor encargado de procesar los algoritmos y proporcionar los resultados correspondientes. En este servidor, está toda la lógica de los algoritmos, podemos destacar dos algoritmos que se han implementado:

1. **Algoritmo de conductores:** Busca optimizar la asignación de conductores a una serie de viajes, minimizando la cantidad de conductores requeridos. Este proceso toma en consideración la disponibilidad y la jornada de trabajo de cada conductor, así como el tiempo adicional de quince minutos que les lleva buscar y dejar la guagua en la estación correspondiente al inicio y fin de su jornada laboral.
    a. **Lógica:** Consiste en asignar a un conductor para cada viaje, siempre y cuando dicho conductor esté disponible y pueda realizar el viaje. En caso contrario, se asignará un nuevo conductor. Cada conductor comienza su jornada en la estación donde inicia su trabajo y, a medida que se le asignan viajes, su información se actualiza, incluyendo la estación en la que se encuentra y el tiempo de viaje.
2. **Algoritmo de zonas:** Busca optimizar recorridos entre zonas teniendo en cuenta datos de telefonía para determinar las rutas que mejor se adapten a la demanda de los usuarios.
    a. Lógica: Cada zona está representada por la parada con más afluencia de personas, cada recorrido tiene un tiempo máximo y has que no se cumpla se añaden pares de paradas para llegar desde una zona A hasta una zona B.

Por otro lado, en el frontend, podemos encontrar la visualización de estos algoritmos, además de establecer visualizaciones por defecto, actualmente podemos destacar 3 visualizaciones:

1. **Algoritmo de conductores:** Una vez que el algoritmo se ejecuta correctamente se retornan dos dataframes:
    a. Dataframe de viajes asignados: Con este dataframe se crea un cronograma donde se muestran los viajes asignados a cada conductor.

b. **Dataframe de descanso de conductores:** Con este dataframe se crea un cronograma con los descansos de cada conductor.

Figura 24: Visualizaciones del algoritmo de conductores.

2. Algoritmos de zonas: Al ejecutarse correctamente, se retorna un dataframe con el que se crea un mapa interactivo a través del servicio de mapbox.

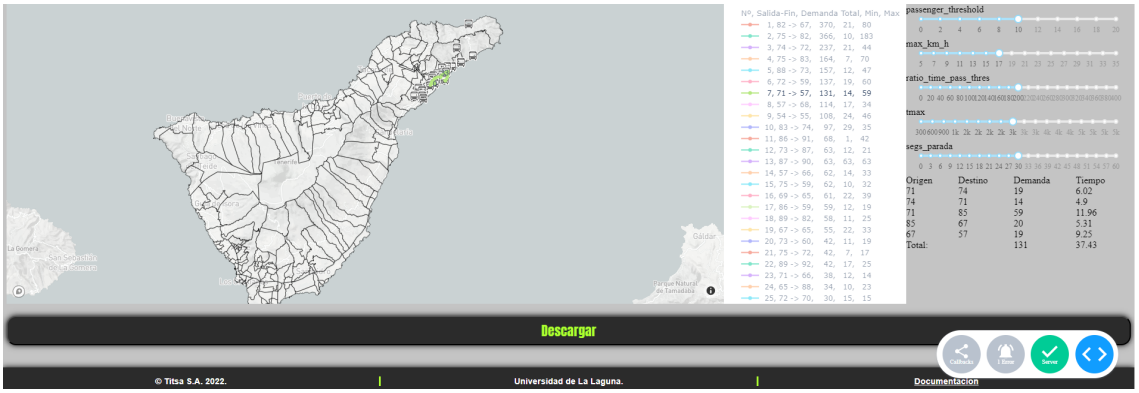


Figura 25 Visualizaciones del algoritmo de zonas.

3. **Visualizaciones por defecto:** En caso de que no exista una visualización para el resultado de un algoritmo se muestra el resultado en formato de tabla.

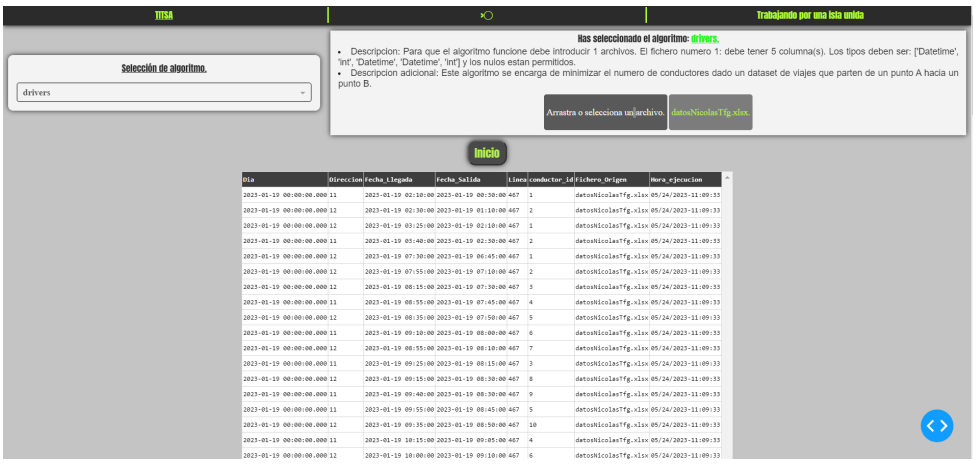


Figura 26 Visualizaciones por defecto.

Por último, se añadió una página de documentación donde se detalla el proceso para añadir nuevos algoritmos y se muestran vídeos explicativos de cómo se añade el algoritmo de zonas tanto en el backend como en el frontend, destacando además que mediante el uso de unidades relativas toda la aplicación es responsiva.

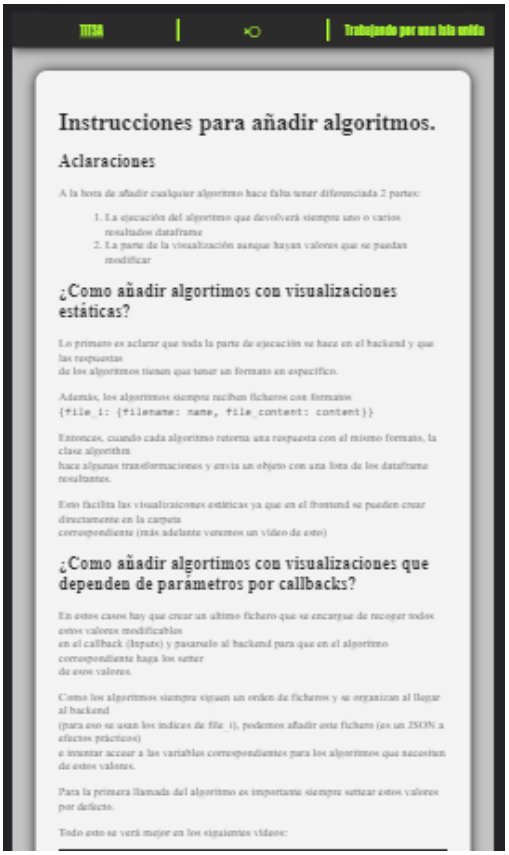


Figura 27: Diseño móvil de la página de documentación.

Capítulo 6: Conclusiones y líneas futuras.

En conclusión, se ha logrado desarrollar de manera exitosa una herramienta web escalable para la visualización de algoritmos y la centralización de su ejecución a través de un servidor. El objetivo principal de este proyecto fue crear una plataforma amigable y accesible que permita a los usuarios ejecutar y visualizar algoritmos de manera eficiente.

Durante el desarrollo de la herramienta, se han abordado diferentes aspectos clave para garantizar su escalabilidad y funcionalidad. Se ha implementado una arquitectura de tres capas, que separa claramente la lógica de negocio, la interfaz de usuario y la infraestructura del servidor. Esto ha permitido una mayor modularidad y mantenibilidad del código, facilitando la incorporación de nuevos algoritmos en el futuro.

Además, se han aplicado buenas prácticas de desarrollo, como el uso de pruebas unitarias y el desarrollo dirigido por pruebas, que han asegurado la calidad y la robustez de la herramienta. Asimismo, se ha hecho uso de tecnologías y frameworks modernos, como Python Flask y Docker, para garantizar un despliegue eficiente y una gestión de dependencias efectiva.

Como línea de mejora futura, se propone la implementación de una arquitectura basada en el enfoque de "vertical slicing" y un diseño centrado en los recursos. Actualmente, el proyecto ha sido desarrollado utilizando una arquitectura hexagonal que ha demostrado ser efectiva en la centralización de la ejecución de algoritmos y la visualización de resultados. Sin embargo, la adopción del enfoque de "vertical slicing" permitiría una mayor modularidad y escalabilidad del sistema.

Además, se plantea la funcionalidad de que la herramienta pueda comparar el resultado de la ejecución de dos o más algoritmos para la misma tarea; o dos parametrizaciones de un mismo algoritmo, mostrando alguna medida de la calidad de la solución propuesta.

Capítulo 7: Summary and conclusions

In conclusion, a scalable web tool has been successfully developed for algorithm visualization and centralized execution through a server. The main objective of this project was to create a user-friendly and accessible platform that enables users to efficiently execute and visualize algorithms.

During the development of the tool, various key aspects were addressed to ensure its scalability and functionality. A three-tier architecture was implemented, clearly separating the business logic, user interface, and server infrastructure. This has allowed for greater code modularity and maintainability, facilitating the incorporation of new algorithms in the future.

Furthermore, good development practices have been applied, such as the use of unit testing and test-driven development, which have ensured the quality and robustness of the tool. Modern technologies and frameworks, such as Python Flask and Docker, have also been utilized to guarantee efficient deployment and effective dependency management.

As a future improvement, the implementation of an architecture based on the "vertical slicing" approach and resource-centric design is proposed. Currently, the project has been developed using a hexagonal architecture, which has proven effective in centralizing algorithm execution and result visualization. However, adopting the "vertical slicing" approach would allow for greater modularity and scalability of the system.

Furthermore, the functionality is proposed for the tool to be able to compare the outcome of executing two or more algorithms for the same task, or two parameterizations of the same algorithm, displaying some measure of the quality of the proposed solution.

Capítulo 8: Presupuesto

Se ha llevado a cabo un presupuesto sobre el esfuerzo realizado durante el Trabajo de Fin de Grado. La duración de este se encuentra alrededor de unas 300 horas, según lo establecido por el plan de estudios de la Universidad de La Laguna.

Tareas	**Duración**	**Coste**
Estudio previo de los visualizadores actuales y las tecnologías a utilizar.	30h	10€/h
Estudio previo de la arquitectura del prototipo para facilitar la incorporación de distintos algoritmos y sus visualizaciones.	30h	25€/h
Desarrollo del Backend.	80h	20€/h
Desarrollo del Frontend.	100h	20€/h
Documentación.	60h	15€/h
Total	**300h**	**5.550€**

Tabla 7.1: Presupuesto.

Finalmente, este proyecto se ha llevado a cabo en **300** horas de trabajo, con un coste de **5.550** euros.

Bibliografía

[1] Ramalhinho Lourenço, H., Pinto Paixão J. y Portugal R. (1998). *Metaheuristics for The Bus-Driver Scheduling Problem*. Recuperado de http://www.econ.upf.edu/~ramalhin/OSHwebpage/Files/WP304.pdf.

[2] Kletzander L., Mannelli Mazzoli T. y Musliu N. *Metaheuristic algorithms for the bus driver scheduling problem with complex break constraints*. In Proceedings of the 2022 ACM Conference on Human-Computer Interaction (pp. 232-240), Recuperado de https://dl.acm.org/doi/abs/10.1145/3512290.3528876.

[3] Álvarez Medina, D. (2022). *Tratamiento y visualización de datos abiertos sobre la demografía en España*. Universidad de La Laguna. Recuperado de https://riull.ull.es/xmlui/handle/915/30308.

[4] Armas Alonso, D. (2022). *Visualiza: procesamiento y visualización de datos abiertos sobre el empleo en Canarias*. Universidad de La Laguna. Recuperado de https://riull.ull.es/xmlui/handle/915/30294.

[5] Castillo De La Rosa, D. (2022). *Prodef-Algorithm: Interfaz para el modelado de meta-heurísticas*. Universidad de La Laguna. Recuperado de https://riull.ull.es/xmlui/handle/915/28703.

[6] Tornero Hernández, A. (2022). *Prodef-saas: despliegue y puesta en marcha de un servicio para la resolución de problemas de optimización*. Universidad de La Laguna. Recuperado de https://riull.ull.es/xmlui/handle/915/29397.

[7] *Dash Python User Guide*. (s.f.). Recuperado el 17 de mayo de 2023 de https://dash.plotly.com/.

[8] *Simple Python version managemen*t. (s.f.). Recuperado el 17 de mayo de 2023 de https://github.com/pyenv/pyenv.

[9] *Crear un entorno virtual en Python utilizando Pyenv*. (2022). Recuperado el 17 de mayo de 2023 de https://roylans.dev/entorno-virtual-en-python-pyenv.

[10] *pyenv*. (2023). pyenv/pyenv. GitHub. https://github.com/pyenv/pyenv

[11] Docker. (s.f.). Recuperado el 17 de mayo de 2023 de https://docs.docker.com/.

[12] *A perfect way to Dockerize your Pipenv Python application*. (2020). Recuperado el 17 de mayo de 2023 de https://sourcery.ai/blog/python-docker/.

[13] *Docker Compose overview*. (s.f.). Recuperado el 17 de mayo de 2023 de https://docs.docker.com/compose/.

[14] Suárez Vera, C. & López Herrera, Y. (2021). *Mesa redonda, arquitectura hexagonal y organización de directorios.*Recuperado el 17 de mayo de 2023 de https://leanmind.es/es/blog/mesa-redonda-arquitectura-y-organizacion-de-directorios

[15] Salguero E. (2018). *Arquitectura hexagonal*. Recuperado el 17 de mayo de 2023 de https://medium.com/@edusalguero/arquitectura-hexagonal-59834bb44b7f.

[16] Novoseltseva E. (2020). *¿Qué es la Arquitectura Hexagonal? Definición y Ejemplos*. Recuperado el 17 de mayo de 2023 de https://apiumhub.com/es/tech-blog-barcelona/arquitectura-hexagonal/.

[17] Ferrera, A. (2020). *Arquitectura Hexagonal en el FrontEnd*. Recuperado el 17 de mayo de 2023 de https://softwarecrafters.io/react/arquitectura-hexagonal-frontend.

[18] *Guía de diseño de API*. (2021). Recuperado el 17 de mayo de 2023 de https://cloud.google.com/apis/design?hl=es-419.

[19] Martin Cecil, R. (2012). *The Clean Architecture*. Recuperado el 17 de mayo de 2023 de Uncle Bob's blog - The Clean Architecture.

[20] Barea, A. (s.f.). *Clean architecture*. Recuperado el 17 de mayo de 2023 de https://www2.deloitte.com/es/es/pages/technology/articles/clean-architecture.html