



UNIVERSIDAD DE MÁLAGA



Grado en Ingeniería del Software

Desarrollo de un modelo del tráfico de una zona urbana de Málaga
a través de técnicas de modelado basada en agentes

Development of a Traffic Model for an Urban Area in Málaga
Using Agent-Based Modeling Techniques

Realizado por
Jorge Camacho García

Tutorizado por
Eduardo Guzmán de los Riscos
Juan Palma Borda

Departamento
Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, septiembre de 2024



UNIVERSIDAD
DE MÁLAGA



UNIVERSIDAD
DE MÁLAGA



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA DEL SOFTWARE

**Desarrollo de un modelo del tráfico de una zona urbana
de Málaga a través de técnicas de modelado basada en
agentes**

**Development of a Traffic Model for an Urban Area in
Málaga Using Agent-Based Modeling Techniques**

Realizado por
Jorge Camacho García

Tutorizado por
Eduardo Guzmán de los Riscos
Juan Palma Borda

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMBRE DE 2024

Fecha defensa: septiembre de 2024

Abstract

The rapid growth of urban populations has led to an increased demand for mobility, significantly intensifying traffic flow within cities. This phenomenon not only deteriorates the quality of life for citizens by exacerbating congestion and stress but also exerts considerable negative impacts on the environment and the economy due to time lost and high resource consumption. In response to these challenges, efficient traffic management has become a crucial priority for authorities and urban planners, who seek innovative solutions to address both current and future demands.

This project focuses on addressing these issues through the development of advanced technological tools for traffic visualization and simulation. The first tool is designed to efficiently analyze large volumes of traffic data, helping to identify critical patterns and support informed decision-making to improve traffic flow. By optimizing data extraction and post-processing, this tool ensures that the information used is accurate and relevant, which is essential for tackling complex urban scenarios.

The second tool centers on traffic simulation, providing a predictive perspective that models vehicular behavior under various scenarios. This capability is crucial for anticipating how transportation systems might respond to changes in infrastructure or traffic management policies. By integrating both tools, the project not only enhances traffic management in densely populated urban areas but also offers a flexible and replicable model applicable to other regions. This approach provides a versatile and scalable solution to mobility challenges in an expanding world.

Keywords: Agent-based model, Traffic simulation, Python, Traffic data visualization, Data refinement

Resumen

El crecimiento acelerado de la población en las zonas urbanas ha generado una creciente demanda de movilidad, lo que ha incrementado significativamente el flujo de tráfico en las ciudades. Este fenómeno no solo deteriora la calidad de vida de los ciudadanos al aumentar los atascos y el estrés, si no que también supone un considerable impacto negativo en el medio ambiente y en la economía, debido al tiempo perdido y al consumo elevado de recursos. En respuesta a estos retos, la gestión eficiente del tráfico se ha convertido en una prioridad crucial para autoridades y planificadores urbanos, que buscan soluciones innovadoras capaces de hacer frente a estos desafíos presentes y futuros.

Este proyecto se enfoca en abordar esta problemática mediante el desarrollo de herramientas tecnológicas avanzadas para la visualización y simulación del tráfico. La primera herramienta permite analizar grandes volúmenes de datos de tráfico de manera eficiente, facilitando la identificación de patrones críticos y la toma de decisiones informadas para mejorar la fluidez del tráfico. Mediante un proceso optimizado de extracción y post-procesamiento de datos, esta aplicación asegura que la información utilizada sea precisa y relevante, elemento fundamental para abordar situaciones complejas en entornos urbanos.

La segunda herramienta se centra en la simulación del tráfico, ofreciendo una perspectiva predictiva que permite modelar el comportamiento vehicular bajo distintos escenarios. Esta capacidad predictiva es crucial para anticipar cómo podrían responder los sistemas de transporte a cambios en la infraestructura o en las políticas de gestión del tráfico. Al integrar ambas herramientas, el proyecto no solo contribuye a la optimización del tráfico en áreas urbanas densamente pobladas, si no que también presenta un modelo adaptable y replicable para otras regiones. Esto proporciona una solución versátil y escalable a los desafíos de movilidad en un mundo en constante expansión.

Palabras clave: **Modelo basado en agentes, Simulación de tráfico, Python, Visualización de datos de tráfico, Refinamiento de datos**

Índice

1. Introducción	9
1.1. Motivación	9
1.2. Objetivos	10
1.3. Estado del arte	11
1.4. Estructura de la memoria	14
2. Metodología de trabajo y Arquitectura del proyecto	17
2.1. Estructura del Proyecto: Etapas e Iteraciones	18
2.2. Arquitectura general del proyecto	19
3. Tecnologías utilizadas	23
3.1. Python	23
3.2. Librerías de Python	24
3.2.1. gdal	24
3.2.2. requests	24
3.2.3. osmnx	24
3.2.4. geojson	24
3.2.5. networkx	24
3.2.6. shapely	24
3.2.7. geopandas	25
3.2.8. pymongo	25
3.3. Dash	25
3.4. Dash Sylvereye	25
3.5. MongoDB	26
3.6. MESA	26
3.7. Notion	27
3.8. Git	27
3.9. GitHub	28

3.10. Diagrams.net	28
4. Especificación y análisis de requisitos	29
4.1. Requisitos	29
4.1.1. Requisitos funcionales	29
4.1.2. Requisitos no funcionales	30
4.2. Casos de uso	31
5. Extracción de la información	49
5.1. Datos necesarios	49
5.2. Evaluación de proveedores	54
5.3. Extracción de datos	55
5.4. Post-procesado de la información	55
5.4.1. Objetivos del post-procesado	55
5.4.2. Traducción a GeoJSON	57
5.4.3. Inclusión de información	65
5.5. Grafo con información	70
5.5.1. Traspaso de datos a la estructura	70
5.5.2. Interpolación de la información	71
5.5.3. Resultado	75
5.6. Almacenamiento	75
6. Panel de datos	81
6.1. Objetivos del tablero	81
6.2. Información disponible	82
6.3. Información procesable	83
6.4. Interfaz de usuario	87
6.4.1. Plataforma del tablero	87
6.4.2. Mapa	88
6.4.3. Extracción de Datos	89
6.4.4. Información del Nodo y de la Arista	90
6.4.5. Opciones de Visualización de Nodos y Aristas	92

6.4.6. Gráficas	92
6.5. Síntesis del Diseño	95
7. Modelo basado en agentes	97
7.1. Introducción	97
7.2. <i>Framework</i> del simulador	98
7.3. Desarrollo del modelo	99
7.3.1. Entorno	99
7.3.2. Agente	101
7.3.3. Comunicación	103
7.3.4. Salida	105
7.4. Interfaz de usuario	107
7.4.1. Parámetros de la simulación	107
7.4.2. Resultados	109
7.5. Conclusiones	110
8. Pruebas de carga	113
8.1. Introducción	113
8.2. <i>Dashboard</i> de los datos de tráfico	114
8.3. Simulador de flujo de tráfico	119
9. Conclusiones y Líneas Futuras	127
9.1. Conclusiones	127
9.2. Dificultades encontradas	128
9.2.1. Datos de tráfico	128
9.2.2. Representación del tráfico en el mapa	129
9.3. Líneas Futuras	129
Apéndice A. Manual de Instalación	135
A.1. Prerrequisitos	135
A.2. Aplicación para obtener datos	136
A.3. Aplicación para procesar los datos	136
A.4. Inicializar la base de datos	138

A.5. Aplicación para el análisis de datos	138
A.6. Aplicación para realizar simulaciones	138
Apéndice B. Manual de Usuario	143
B.1. Extractor y procesador de datos	143
B.2. Aplicación para el análisis de datos	143
B.2.1. Editar estilo del mapa	143
B.2.2. Ver información adicional	145
B.2.3. Filtrar las vías del grafo	146
B.2.4. Realizar análisis de los datos	146
B.2.5. Cargar tráfico de una fecha en el mapa	148
B.2.6. Gráficas	148
B.2.7. Interfaz completa	150
B.3. Aplicación para realizar simulaciones	151
B.3.1. Filtrar las vías del grafo	151
B.3.2. Gráficas	151
B.3.3. Realizar simulaciones	151
B.3.4. Cargar tráfico de una fecha en el mapa	154
B.3.5. Interfaz completa	154

1

Introducción

1.1. Motivación

En la actualidad, las ciudades enfrentan desafíos complejos y en constante evolución en la gestión del tráfico y la movilidad urbana. El **crecimiento de la población**, la expansión urbana y el aumento del uso de vehículos motorizados contribuyen a la congestión, los accidentes y las emisiones de gases contaminantes. Estos problemas no solo afectan la **calidad de vida** de los habitantes, si no que también tienen repercusiones económicas y ambientales significativas. Para abordar estos desafíos de manera efectiva, es esencial comprender en profundidad el comportamiento del tráfico en entornos urbanos específicos.

Una herramienta poderosa para lograr este entendimiento es la creación de un **modelo digital del tráfico**, que actúa como una representación virtual precisa y dinámica de un sistema de tráfico real. En este proyecto, el modelo digital se centra en una sección urbana de Málaga, replicando digitalmente el flujo de tráfico, la topología de la ciudad, las interacciones entre vehículos, y otros factores clave que influyen en la movilidad. Un modelo digital no es solo una colección de datos almacenados, si no una herramienta que integra diversos aspectos del tráfico en un entorno virtual que permite simular y analizar su comportamiento bajo diferentes condiciones.

Los modelos digitales del tráfico permiten a los investigadores y a los gestores urbanos explorar cómo los cambios en las infraestructuras, las políticas de tráfico o las condiciones externas podrían afectar el flujo vehicular y la movilidad en la ciudad. Por ejemplo, se pueden simular escenarios de emergencia o la implementación de nuevas normativas de tráfico. Esto facilita la planificación y optimización de estrategias de gestión del tráfico, lo que resulta en una mayor eficiencia y seguridad vial.

Además, un modelo digital ofrece una plataforma avanzada para la **visualización y análisis**

lisis de datos de tráfico a lo largo de un periodo determinado. A través de herramientas interactivas, como filtros y gráficos dinámicos, los usuarios pueden desglosar y examinar los datos con un alto grado de detalle, lo que permite identificar patrones, predecir tendencias y tomar decisiones basadas en datos sólidos. Esta capacidad de análisis detallado es crucial para optimizar el flujo de tráfico, mejorar la seguridad vial y promover una movilidad más sostenible.

El valor de un modelo digital del tráfico radica en su capacidad para proporcionar información dinámica que puede guiar la toma de decisiones en el ámbito de la planificación urbana y la gestión de la movilidad. Al **combinar datos históricos con simulaciones predictivas**, los modelos digitales permiten a los gestores de tráfico anticiparse a los problemas y aplicar soluciones antes de que se conviertan en crisis. En resumen, la creación y utilización de un modelo digital del tráfico es fundamental para abordar de manera proactiva los desafíos urbanos modernos.

1.2. Objetivos

En este contexto, el proyecto tiene como objetivo diseñar y construir un modelo digital del tráfico en la ciudad de Málaga utilizando datos reales. Este modelo se desarrollará utilizando el lenguaje de programación Python y aplicando diversas técnicas de modelado basadas en Inteligencia Artificial, como el Modelado y Simulación basada en Agentes. Específicamente, modelaremos una sección del tráfico urbano en el área de Teatinos, lo que permitirá una representación detallada y localizada del flujo vehicular en esta zona clave de la ciudad.

Este objetivo principal se descompone en los siguientes subobjetivos de investigación:

- **Extracción y recopilación de datos en tiempo real:** Recopilar y extraer datos de tráfico en tiempo real para modelar el tráfico urbano de manera realista. Este objetivo incluye hacer uso de diferentes APIs para la recopilación de los datos necesarios.
- **Almacenamiento de datos:** Almacenar toda la información recogida en una base de datos que permita un acceso rápido y eficiente a los datos para su posterior análisis.
- **Transformación de datos:** Transformar los datos recogidos de tráfico a una estructura matemática, como un grafo, que permita su tratamiento y estudio de manera adecuada.

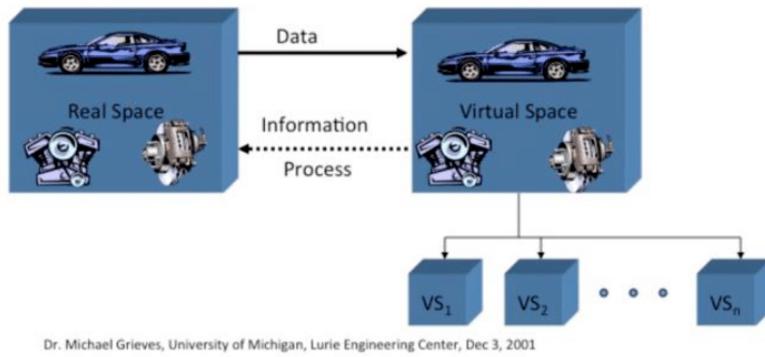


Figura 1: Idea conceptual ideal del ciclo de vida del producto por Grieves [9].

- **Diseño de un interfaz gráfica para el modelo digital:** Diseñar un *dashboard* que reúna toda la información recogida, permitiendo su análisis y estudio a través de diferentes filtros. Este *dashboard* también deberá ofrecer gráficas adicionales con información relativa a diferentes plazos de tiempo, no solo instantes específicos. Se deberá garantizar una buena interfaz de usuario que trabaje en conjunto con un sistema eficiente para manejar grandes volúmenes de datos.
- **Simulación basada en agentes:** Desarrollar un modelo basado en agentes que sea capaz de simular el tránsito de vehículos en diferentes instantes de información del tráfico recogidos, proporcionando información valiosa a partir de dichas simulaciones.

1.3. Estado del arte

Los gemelos digitales, o *digital twins*, han emergido como herramientas fundamentales en diversas industrias desde principios de los años 2000. Este concepto fue inicialmente propuesto por Michael Grieves en 2002, como parte de un modelo para la Gestión del Ciclo de Vida del Producto (PLM, por sus siglas en inglés) (fig. 1). El objetivo principal del gemelo digital en este contexto era **crear una réplica digital precisa de un producto físico**, permitiendo a las empresas gestionar y optimizar todo el ciclo de vida del producto, desde su diseño hasta su retirada del mercado [10].

El concepto original no fue denominado de inmediato como «gemelo digital». En sus primeras etapas, Grieves se refirió a él como el «Modelo de Espacios Reflejados» (*Mirrored Spaces Model*) en 2005, y posteriormente como el «Modelo de Espejo de Información» (*Information*

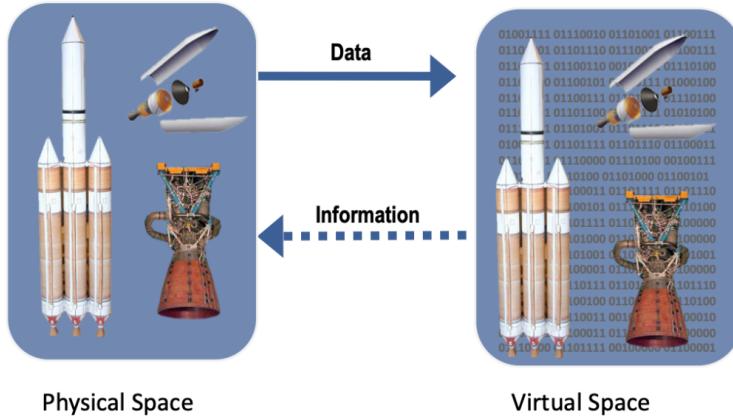


Figura 2: Concepto de gemelo digital de Grieves y Vickers [11].

Mirroring Model) en 2006. No fue hasta 2011 que el término «*Digital Twin*» comenzó a consolidarse, con el apoyo de John Vickers de la NASA (fig. 2), quien contribuyó a popularizar esta denominación [10].

El modelo de gemelo digital propuesto por Grieves incluye tres componentes principales:

- los productos físicos en el «Espacio Real».
- los productos virtuales en el «Espacio Virtual».
- las conexiones de datos e información que vinculan los productos virtuales y reales.

Esta integración permite que cada producto físico tenga un homólogo virtual detallado, lo que facilita el seguimiento y la optimización de la fabricación, el mantenimiento y el desarrollo de nuevas generaciones de productos [10].

Inicialmente, los gemelos digitales encontraron aplicación en sectores altamente técnicos, como la industria aeroespacial y de defensa, donde la precisión y la optimización eran cruciales. Sin embargo, a medida que la tecnología ha avanzado, su uso se ha expandido a otros sectores, incluyendo manufactura, salud, infraestructura urbana, y más. Hoy en día, las empresas líderes incorporan gemelos digitales en diversas fases de sus procesos. Junto a los gemelos digitales, han surgido conceptos relacionados como el «**Digital Shadow**» y los «**Modelos Digitales**», que representan diferentes niveles de integración y retroalimentación entre el mundo físico y el virtual. Estas variaciones se han desarrollado para satisfacer necesidades específicas en diferentes industrias, adaptándose a la complejidad y los requisitos de cada entorno.

El estudio y la aplicación de modelos digitales en el ámbito del tráfico urbano comenzaron a ganar tracción a partir de la última década. Impulsado por avances tecnológicos significativos como sensores mejorados y análisis avanzado de datos, este desarrollo se alineó con el creciente interés en ciudades inteligentes y la necesidad de mejorar la gestión urbana, debido al crecimiento significativo de población de muchas ciudades en los últimos años [3].

Desde las primeras investigaciones en simulaciones y modelos predictivos para optimizar el flujo de tráfico, se ha avanzado notablemente en la **integración de estos modelos en proyectos más grandes**. En muchas ocasiones, los gemelos digitales del tráfico forman parte de iniciativas más amplias, como el gemelo digital de una ciudad completa. Un ejemplo destacado es Helsinki, donde se ha implementado un modelo de gemelo digital para toda la ciudad, permitiendo una gestión integrada y optimizada del tráfico [13].

Las investigaciones y los proyectos relacionados con modelos del flujo de tráfico se han ido centrando en torno a **tres tipos de representaciones**:

- **Microscópico:** este tipo de modelos busca representar con exactitud el comportamiento y las interacciones individuales, destacando por lo costoso que puede llegar a ser este modelo en comparación a los demás. Las representaciones de este tipo destacan también por la forma en la que se modela el comportamiento de los vehículos (matemático, autómata celular, basado en actividades ...).
- **Macroscópico:** este modelo es el más abstracto siendo, por ello, incapaz de representar ciertos detalles del tráfico, como el cambio de carril o gestión individual de destinos, ya que representa el flujo de tráfico como nubes de gas (*Gas-kinetic Models*). Además este tipo de modelos quedan limitados a zonas urbanas.
- **Mesoscópico:** este es el modelo más extendido y los podríamos ubicar entre el microscópico y el mesoscópico. Aunque tenga un comportamiento más generalizado que el microscópico, tendremos resultados más precisos que con el modelo macroscópico. Los vehículos pueden representarse de una manera individualizada (como en el microscópico) o también como paquetes de vehículos (denominados en la literatura como convoyes).
- **Híbrido:** finalmente, tenemos la posibilidad de mezclar dos o incluso los tres tipos de modelos. Aunque lo más común es mezclar el modelo mesoscópico con otro, ya que al

ser este el más genérico, permite incluir modificaciones que lo aproximan a otro tipo de modelos.

La zona que se quiera modelar, junto con los factores de tráfico que se quieran estudiar, son algunos de los principales factores que dictaminarán qué tipo de modelo para cada problema.

Además, la técnica de **Modelado Basado en Agentes (ABM)** ha adquirido una importancia significativa en estos proyectos. En Sarajevo, por ejemplo, el uso de ABM ha sido clave para simular políticas de tráfico y planificar infraestructuras urbanas de manera precisa [23]. Esta técnica de inteligencia artificial permite modelar interacciones complejas entre agentes (como vehículos y peatones), proporcionando una visión detallada del comportamiento del tráfico y mejorando la capacidad para prever y gestionar diversos escenarios urbanos.

Dentro de este contexto, son muy utilizados para poder ver como afectarían cambios en las políticas de transporte o en las carreteras, y en general, para el estudio previo a la toma de decisiones reales. Las simulaciones basadas en agentes son las más empleadas para este tipo de problemas, ya que gracias a la **flexibilidad y escalabilidad** que nos ofrecen, podremos alcanzar **simulaciones muy realistas** y fieles a la realidad del entorno que tenemos.

La **interfaz de usuario** en los paneles de datos interactivos para el análisis de tráfico es un aspecto crucial que a menudo se pasa por alto en muchos proyectos. En la mayoría de los casos, no se presta suficiente atención a la usabilidad y presentación visual de estos paneles, lo que puede limitar su efectividad. Sin embargo, una interfaz bien diseñada es fundamental para maximizar la eficiencia en la toma de decisiones. Un ejemplo inspirador es el panel desarrollado por **NVIDIA**, que visualiza el tiempo de transporte dentro de un radio determinado desde un punto seleccionado, acompañado de datos demográficos [14]. Esta herramienta destaca por su enfoque en la usabilidad y accesibilidad, y aunque sea una demostración de una herramienta para el procesamiento de datos, su interfaz ha servido como referencia en el desarrollo del panel de datos de este TFG, que busca ser igualmente útil e intuitivo para los usuarios .

1.4. Estructura de la memoria

El documento se organizará siguiendo el orden de las fases que se han seguido en el desarrollo del proyecto.

- **Capítulo 1: Introducción:** En este primer capítulo se ha presentado una visión general

del proyecto, explicando la motivación que subyace a su desarrollo y los objetivos específicos que se pretenden alcanzar. Se ha discutido la relevancia del estudio del tráfico urbano y cómo un modelo digital puede contribuir a mejorar la movilidad y la gestión del tráfico en Málaga. Además, se ha realizado un estudio del estado del arte, en el que se han revisado las principales investigaciones y enfoques en el modelado del tráfico utilizando técnicas de Inteligencia Artificial.

- **Capítulo 2: Metodología de trabajo y arquitectura del proyecto:** Este capítulo describirá en detalle la metodología de trabajo adoptada para el desarrollo del proyecto, fundamentando la elección del modelo iterativo. Se explicará cómo cada etapa del proyecto se basa en la completitud y consistencia de la anterior, y la importancia de seguir un enfoque estructurado y secuencial. Se detallarán las partes y fases del proyecto.
- **Capítulo 3: Tecnologías utilizadas:** Este capítulo recogerá el lenguaje de programación en el que se ha realizado el proyecto, incluyendo las principales librerías y frameworks utilizados, junto a la base de datos.
- **Capítulo 4: Especificación y análisis de requisitos:** Este capítulo se centrará en la especificación y análisis de los requisitos funcionales y no funcionales de la aplicación. Además, se desarrollarán casos de uso detallados, incluyendo sus descripciones, pre-condiciones, post-condiciones, escenarios principales y alternativos, así como diagramas de secuencia, para asegurar una clara comprensión de las interacciones del sistema.
- **Capítulo 5: Extracción de la información:** Este capítulo se centrará en el proceso de selección y evaluación de los diferentes servicios y proveedores de información de tráfico. Se detallará cómo se investigaron y compararon las APIs disponibles para determinar la más adecuada para el proyecto. Se describirá el desarrollo de la plataforma de recopilación de datos. Y finalmente se detallará el proceso de procesado y refinado de la información obtenida.
- **Capítulo 6: Panel de datos:** En este capítulo se abordará la implementación del panel de visualización de datos. Se describirá la arquitectura del sistema, incluyendo las conexiones necesarias con la base de datos MongoDB y las tecnologías empleadas para el *front-end*. Se explicará cómo se desarrollaron las métricas relativas al tráfico y las

diversas funcionalidades del panel, como filtros y gráficos interactivos.

- **Capítulo 7: Modelo basado en agentes:** En este capítulo se detallará el modelo desarrollado para simular el tráfico de vehículos. Se explicarán las diferentes opciones implementadas para que los agentes presenten comportamientos variados y la interfaz utilizada para configurar el simulador y visualizar el grafo y los resultados de cada simulación realizada.
- **Capítulo 8: Pruebas de carga:** En este capítulo se detallarán las pruebas de carga llevadas a cabo tanto en el panel de datos como en el modelo basado en agentes desarrollado. Se registrarán los tiempos de procesamiento asociados a distintas operaciones y escenarios para cada aplicación. Posteriormente, se realizará un análisis comparativo de estos tiempos para evaluar el rendimiento de las aplicaciones bajo diferentes condiciones. Además, se discutirá cómo estos resultados afectan la eficiencia general y la capacidad de respuesta de las aplicaciones.
- **Capítulo 9: Conclusiones y Líneas Futuras:** En este capítulo se presentarán las conclusiones derivadas del proyecto, recapitulando los principales hallazgos y logros obtenidos en los capítulos anteriores. También se discutirá la precisión de las simulaciones y la utilidad de las herramientas y tecnologías empleadas. Finalmente, se explorarán posibles direcciones para futuras investigaciones y la ampliación del proyecto, incluyendo la integración de nuevas fuentes de datos, la mejora de los algoritmos de simulación y la implementación de nuevas funcionalidades en la interfaz gráfica.
- **Apéndice A. Manual de Instalación**
- **Apéndice B. Manual de Usuario**

2

Metodología de trabajo y Arquitectura del proyecto

Para la realización de este proyecto, se ha seguido una **metodología iterativa de investigación y desarrollo**. Este enfoque es ideal para proyectos en los que los requisitos pueden evolucionar con el tiempo y donde es necesario realizar ajustes continuos para asegurar que el producto final cumpla con los objetivos definidos. A diferencia de metodologías como las ágiles, que se centran en ciclos de desarrollo muy cortos y la entrega continua de valor, la metodología iterativa proporciona un marco estructurado que permite un equilibrio entre flexibilidad y planificación detallada.

En un proyecto como este, que involucra un alto grado de investigación técnica y científica, es crucial tener **fases claramente definidas** que permitan una validación y verificación continuas antes de avanzar a la siguiente etapa. La metodología iterativa permite estos ajustes y validaciones de manera controlada, asegurando que cada componente se integre correctamente y funcione de manera óptima. Esto es especialmente importante dado que el desarrollo de un modelo digital complejo requiere precisión y una integración adecuada de todos sus componentes.

Además, la naturaleza del proyecto, que abarca desde la investigación tecnológica hasta la implementación de técnicas avanzadas de simulación y análisis, demanda un enfoque que permita **revisiones y mejoras continuas**, algo que se logra eficientemente a través de iteraciones sucesivas. Por estas razones, la metodología iterativa se considera la opción más

adecuada para asegurar la calidad y el éxito del proyecto.

2.1. Estructura del Proyecto: Etapas e Iteraciones

Cada una de las iteraciones llevadas a cabo en el proyecto, se programaron para tener una duración aproximada de entre una y tres semanas, en función de la complejidad de la tarea a realizar, de forma que tras cada iteración se tuviera un producto o resultado funcional sobre el que seguir iterando hasta cambiar de etapa. En la fig. 3 podemos ver representadas las etapas, en color azul, y cada una de las iteraciones, de color verde, a modo de esquema del trabajo realizado.

- **Etapa 1: Estudio del arte:** En cualquier trabajo de investigación, es fundamental estudiar tanto las referencias más actualizadas de la literatura científica como los enfoques históricos o iniciales en el modelado del tráfico, con el fin de comprender las propuestas realizadas por diversos investigadores en este campo. Durante este período, se llevaron a cabo 2 iteraciones, en las que se revisaron y refinaron continuamente las fuentes y enfoques relevantes de diferentes tipos de investigaciones y artículos científicos.
- **Etapa 2: Investigación y Evaluación de Tecnologías:** En esta fase inicial, se realizó una investigación exhaustiva de las tecnologías disponibles para el proyecto, incluyendo *OSMnx*, *NetworkX*, *OpenStreetMap*, *GeoPandas*, *MongoDB*, etc. Se evaluaron sus ventajas y desventajas para determinar las más adecuadas para la creación del modelo digital y el análisis del tráfico. Esta etapa se dividió en 2 iteraciones, donde se evaluó cada tecnología y se estudió si integrarla en el proyecto o descartarla según sus ventajas y desventajas.
- **Etapa 3: Viabilidad de la extracción de datos sobre el tráfico:** Se investigó la viabilidad de utilizar *APIs* de diferentes proveedores para recopilar datos de tráfico en tiempo real y se exploró su posible integración en el modelo digital. Esta etapa constó de 2 iteraciones, en las que se exploró y seleccionó la mejor *API* y se decidió la zona de Málaga para el modelado. Los resultados de cada iteración ayudaron a refinar la selección y la integración de los datos.
- **Etapa 4: Recopilación, Análisis y Modelado de los datos:** Una vez decidida la *API* a utilizar, se desarrolló una aplicación de recogida de datos que estuvo en funcionamiento

constante hasta la finalización del proyecto. Y otra para el post-procesado y refinado de los datos. Esta etapa fue una de las más largas debido al grueso de trabajo, con 4 iteraciones. En cada iteración, se trabajó en una correcta transformación de los datos para asegurar la calidad y precisión de la información para las siguientes etapas.

- **Etapa 5: Desarrollo de la Aplicación para la Gestión de la Información Recolec-tada:** Durante esta etapa, se desarrolló una aplicación destinada a la gestión y análisis de la información recolectada. Permite a los usuarios explorar los datos, realizar estudios automáticos mediante diferentes algoritmos y generar informes detallados. La fase se completó en 2 iteraciones, donde se integraron los datos recopilados y se implementaron herramientas de análisis avanzadas.
- **Etapa 6: Creación del Simulador Basado en Agentes e Interfaz de Usuario:** En esta última etapa, se desarrolló un simulador basado en agentes que permite la simu-lación del tráfico en un entorno digital con múltiples agentes interactuando. Además, se creó una interfaz de usuario que permite configurar las simulaciones, gestionar los resultados y ajustar los parámetros de los agentes. Esta etapa también fue muy extensa, ya que involucró 3 iteraciones, centradas en el desarrollo del entorno de simulación, la programación de los agentes y la integración de una interfaz intuitiva.

2.2. Arquitectura general del proyecto

El proyecto se divide en **tres partes principales**, como se muestra en la figura 4: recolección y procesado de datos, visualización de los datos procesados, y simulación del tráfico urbano utilizando un modelo basado en agentes.

La **primera parte** se enfoca en la recolección, procesamiento y almacenamiento de los datos. Aquí se integran diversas fuentes de datos sobre el tráfico y la red de carreteras. Se realiza su transformación y limpieza para garantizar su utilidad y coherencia.

La **segunda parte** se centra en un *dashboard* de datos diseñado para visualizar los datos recolectados. Este módulo facilita la exploración de datos de tráfico mediante análisis automá-ticos del histórico disponible. El usuario puede seleccionar los datos que desea analizar, y una vez realizado el análisis, el sistema genera gráficos y tablas interactivas que permiten explorar diversos aspectos del tráfico urbano de forma visual y accesible.

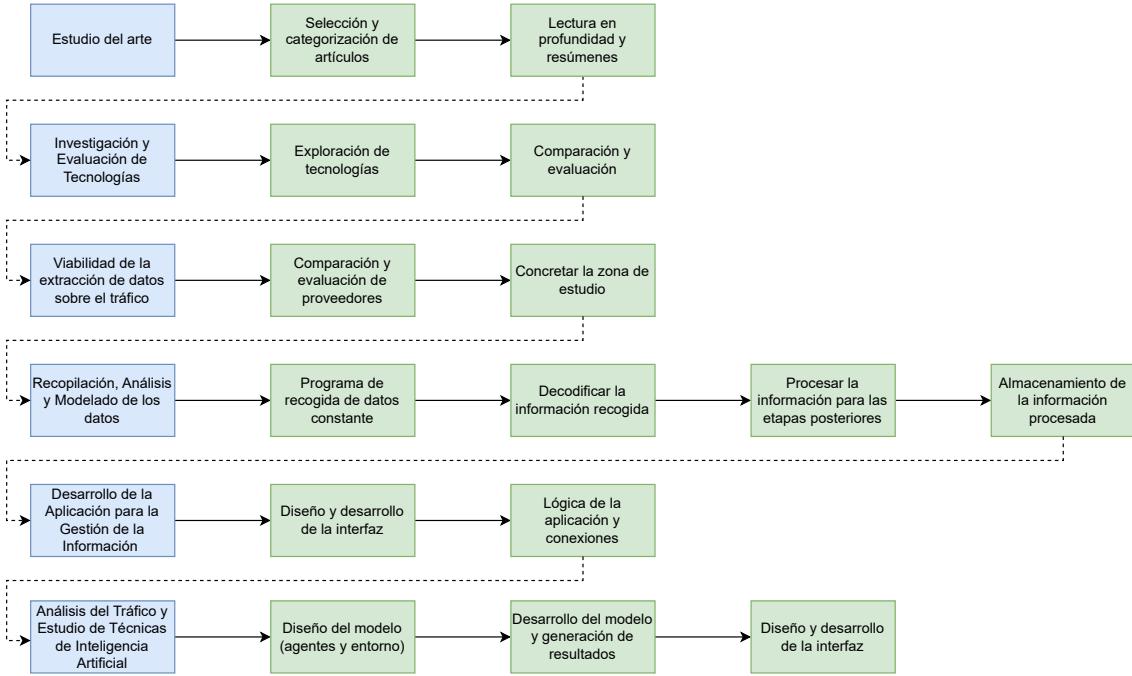


Figura 3: Diagrama con las fases e iteraciones del desarrollo.

La **tercera parte** se dedica al modelo basado en agentes y a el *dashboard* que lo incorpora. Este módulo permite simular el comportamiento del tráfico en la red vial urbana, utilizando los datos previamente procesados. El simulador utilizará las políticas seleccionadas por el usuario. Gracias a la información recolectada del modelo, podrán ofrecerse gráficas explicativas.

Como se observa en la fig. 4, las secciones azul, morada y verde de la parte superior corresponden a la **primera parte** del proyecto, relacionada con la recolección y procesamiento de datos. Debajo de esta, en la sección roja, se encuentran el modelo y el analizador de datos con sus respectivas interfaces, que forman parte de la **segunda y tercera parte** del proyecto. La figura ilustra esquemáticamente la interacción y el flujo de datos entre las tres partes del proyecto, destacando los puntos clave de entrada y salida de datos entre los diferentes módulos.

Los capítulos 5, 6 y 7 del presente documento corresponden a estas tres partes del proyecto, donde se explicarán en detalle cada uno de los componentes y su implementación.

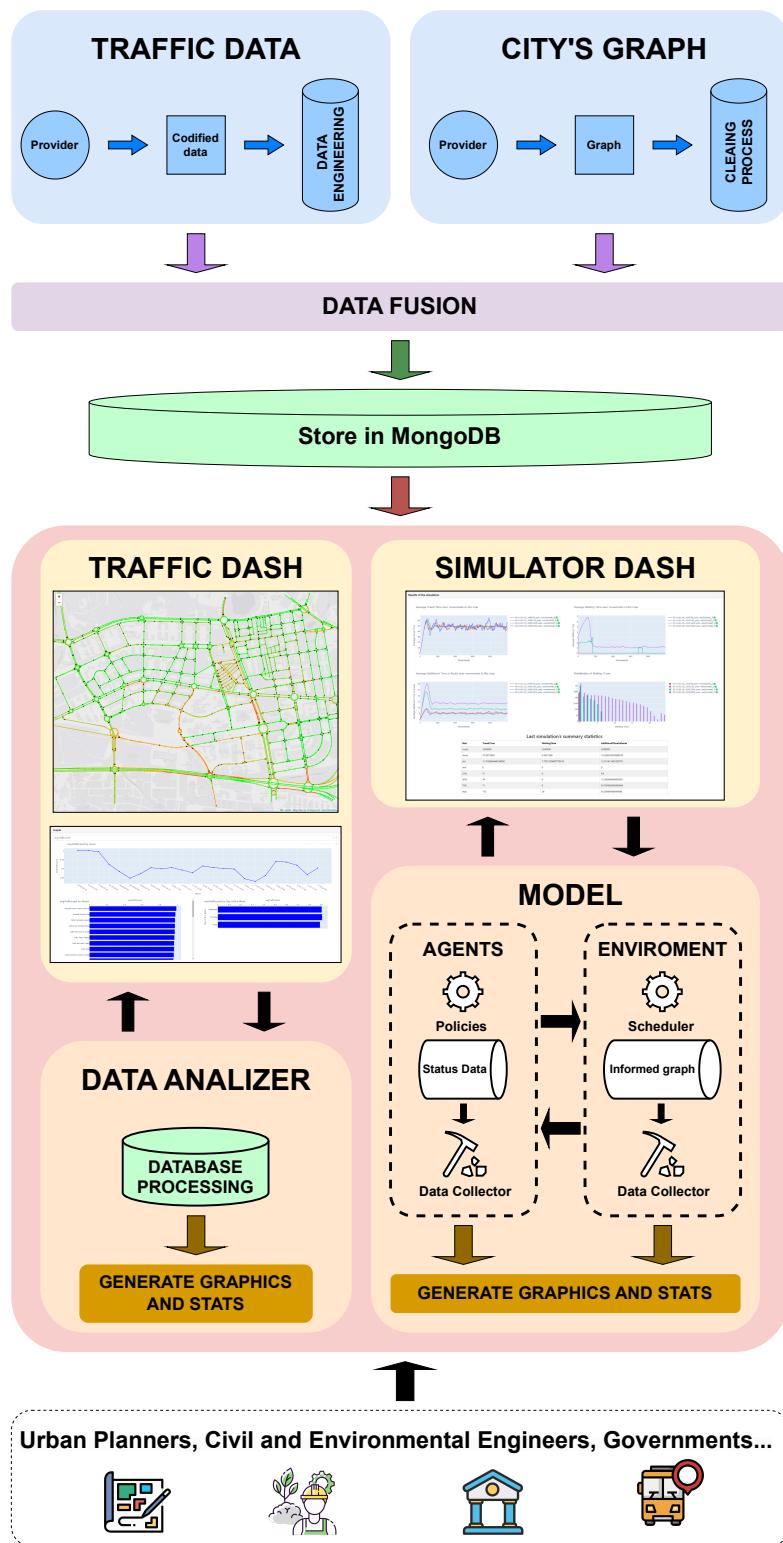


Figura 4: Diagrama general del proyecto.

3

Tecnologías utilizadas

3.1. Python

Python es un lenguaje de programación de **alto nivel, interpretado y de propósito general** [26]. Es conocido por su sintaxis clara y legible, lo que permite reducir significativamente la cantidad de líneas de código en comparación con otros lenguajes. Python ha liderado los rankings de popularidad global durante varios años, principalmente por su versatilidad y la gran cantidad de librerías desarrolladas por la comunidad para diversos propósitos [28].

Python es particularmente destacado en áreas como el **manejo y la representación de datos**, así como en el desarrollo de **herramientas de inteligencia artificial**. Su ecosistema de librerías permite a los desarrolladores construir aplicaciones complejas con rapidez y eficiencia. Además, la vasta comunidad de usuarios y desarrolladores garantiza un soporte constante y la evolución continua del lenguaje.

Una de las ventajas clave de Python es su **compatibilidad** con una amplia gama de plataformas y sistemas operativos, lo que lo convierte en una opción ideal para proyectos que requieren portabilidad y flexibilidad. Asimismo, Python facilita la integración con otros lenguajes y tecnologías, lo que permite la creación de soluciones híbridas adaptadas a necesidades específicas.

Todo el código desarrollado para este proyecto ha sido escrito en Python, utilizando diferentes librerías y frameworks que se describen a continuación.

3.2. Librerías de Python

3.2.1. gdal

GDAL es una biblioteca geoespacial utilizada para el procesamiento y análisis de datos geográficos [5]. Es crucial para manejar grandes volúmenes de datos espaciales, permitiendo su transformación y análisis de manera eficiente.

3.2.2. requests

Requests es una librería que facilita la realización de solicitudes *HTTP* de manera simple y eficiente [27]. Ha sido utilizada para interactuar con *APIs* y obtener datos en tiempo real.

3.2.3. osmnx

OSMnx es una herramienta para descargar y modelar datos de *OpenStreetMap* en formato de red de grafos [2]. Es fundamental para la extracción y manipulación de datos geoespaciales en este proyecto.

3.2.4. geojson

GeoJSON es una librería para trabajar con el propio formato *GeoJSON*, utilizado para representar datos geoespaciales [25]. Es esencial para la integración y manipulación de datos espaciales en Python.

3.2.5. networkx

NetworkX es una librería para la creación, manipulación y estudio de la estructura, dinámica y funciones de grafos complejos [12]. Ha sido utilizada para analizar redes de tráfico y modelar rutas.

3.2.6. shapely

Shapely es una librería para la manipulación y análisis de objetos geométricos planos [6]. Es crítica para realizar operaciones geométricas en el análisis de rutas y modelos de tráfico.

3.2.7. geopandas

GeoPandas es una extensión de pandas que facilita el trabajo con datos geoespaciales en Python [19]. Su integración ha mejorado el manejo y análisis de datos espaciales, aportando robustez al proyecto.

3.2.8. pymongo

PyMongo es la librería oficial para interactuar con bases de datos *MongoDB* en Python [22]. Facilita la conexión y las operaciones *CRUD*, siendo esencial para manejar la base de datos en este proyecto.

3.3. Dash

Dash es un framework de Python desarrollado por *Plotly* para la construcción de **aplicaciones web interactivas** y visualizaciones de datos en tiempo real [24]. Está diseñado para permitir a los desarrolladores crear aplicaciones web complejas con una interfaz gráfica rica utilizando únicamente Python. *Dash* combina la simplicidad y la potencia de los gráficos de *Plotly* con la flexibilidad de las aplicaciones web interactivas, permitiendo construir *dashboards* analíticos que pueden integrarse fácilmente en sistemas más amplios.

En este proyecto, *Dash* ha sido utilizado para desarrollar una interfaz de usuario que permite **visualizar los datos** recolectados, realizar análisis interactivos y generar reportes en tiempo real. La capacidad de *Dash* para integrar múltiples componentes visuales, como gráficos, tablas y controles de interfaz de usuario, ha sido fundamental para crear una herramienta que no solo muestra datos, si no que también permite su exploración en profundidad.

Además, *Dash* facilita la creación de aplicaciones que pueden ser desplegadas tanto en entornos locales como en la nube, ofreciendo una gran versatilidad en la implementación. Su capacidad para manejar grandes volúmenes de datos y actualizar visualizaciones en tiempo real ha sido crucial para el proyecto.

3.4. Dash Sylvereye

Dash Sylvereye es una librería de código abierto creada por la comunidad de *Dash* que proporciona un **componente tipo mapa** con más funcionalidades que otros componentes

similares [4]. Gracias a la integración de los grafos en el componente, y su capacidad dinámica, ha facilitado la creación de una interfaz óptima para el proyecto.

3.5. MongoDB

MongoDB es una **base de datos no relacional basada en documentos** que almacena datos en un formato flexible similar a *JSON* [21]. Esto permite una representación más natural y dinámica de los datos, adaptándose perfectamente a la naturaleza cambiante de los proyectos modernos. *MongoDB* es conocida por su capacidad de escalar horizontalmente y por su flexibilidad, lo que la hace ideal para aplicaciones que requieren manejar grandes volúmenes de datos y alta disponibilidad.

Una de las principales ventajas de *MongoDB* es su **modelo de datos flexible**, que no requiere un esquema fijo. Esto permite agregar nuevos campos a los documentos sin necesidad de redefinir toda la estructura de la base de datos, lo que acelera el desarrollo y la iteración. Además, *MongoDB* proporciona potentes capacidades de consulta y agregación en tiempo real, lo que facilita el análisis de datos directamente desde la base de datos.

Para este proyecto, *MongoDB* ha sido esencial debido a su **capacidad para manejar grandes volúmenes de datos** en tiempo real y su facilidad de integración con Python mediante la librería *PyMongo*. Dado que la prioridad de nuestra aplicación es la velocidad y la flexibilidad, no se ha utilizado un mapeo relacional a objetos (*ORM*), lo que podría haber ralentizado el procesamiento. *PyMongo* ha sido la opción ideal, recomendada oficialmente por *MongoDB*, permitiendo un acceso rápido y eficiente a los datos.

3.6. MESA

MESA es un *framework* de Python diseñado para la **construcción y simulación de modelos basados en agentes** [20]. Los modelos basados en agentes son una poderosa herramienta para simular comportamientos complejos en sistemas dinámicos, donde múltiples agentes autónomos interactúan entre sí y con su entorno. *MESA* proporciona una plataforma flexible y extensible que permite a los investigadores y desarrolladores crear, experimentar y analizar modelos de simulación de manera eficiente.

En este proyecto, *MESA* ha sido utilizado para **simular el tráfico urbano** mediante un

modelo basado en agentes, donde cada vehículo es representado como un agente independiente con comportamientos definidos. Esta simulación ha permitido estudiar cómo diferentes políticas de tráfico podrían afectar el flujo de tráfico en la ciudad.

Una de las fortalezas de *MESA* es su capacidad para integrarse con otras librerías y *frameworks* de Python, como *NetworkX* para el análisis de grafos o *Plotly* para la visualización de resultados. Esta integración ha sido crucial para desarrollar un modelo de tráfico robusto y altamente configurable.

3.7. Notion

Notion es una **plataforma de productividad y organización** que combina funcionalidades de toma de notas, gestión de proyectos y colaboración en un solo espacio de trabajo [16]. Permite a los usuarios crear, organizar y gestionar tareas, documentos, bases de datos y calendarios de manera flexible y personalizada. Su capacidad para integrar diferentes tipos de contenido en un solo lugar la hace ideal para la gestión de proyectos complejos.

En el proyecto, *Notion* ha sido utilizado como organizador de tareas y documentos. Se ha empleado para llevar un registro de los documentos científicos leídos, los resúmenes realizados y otros aspectos clave del proyecto. La plataforma ha facilitado la organización del trabajo al permitir la centralización de información y el seguimiento del progreso en un solo lugar.

3.8. Git

Git es un **sistema de control de versiones distribuido** que permite gestionar el historial de cambios en el código fuente de manera eficiente [29]. Ofrece la capacidad de mantener un registro detallado de cada modificación realizada, facilitando la colaboración y la gestión de versiones en proyectos de software. Git permite a los desarrolladores realizar seguimiento de cambios, revertir a versiones anteriores y gestionar ramas del proyecto de manera organizada.

En este proyecto, *Git* ha sido utilizado como el sistema de control de versiones para gestionar el código fuente. Ha permitido llevar un registro detallado de las modificaciones realizadas a lo largo del desarrollo del proyecto, facilitando el manejo de diferentes versiones y la implementación de nuevas funcionalidades sin perder el historial de cambios.

3.9. GitHub

GitHub es una plataforma basada en la web que utiliza *Git* para el control de versiones y el **almacenamiento de código fuente** [7]. Ofrece un entorno centralizado para alojar repositorios *Git*, permitiendo el acceso remoto al código y facilitando la colaboración a través de herramientas de revisión de código y gestión de incidencias. *GitHub* también proporciona funcionalidades adicionales como la integración continua y el despliegue.

En este proyecto, *GitHub* ha sido utilizado para almacenar el código fuente en un repositorio centralizado. Ha permitido mantener el código organizado y accesible desde cualquier ubicación, así como gestionar el historial de versiones del proyecto. Aunque el trabajo se ha realizado de forma individual, *GitHub* ha sido crucial para mantener un registro claro y ordenado del desarrollo del proyecto.

3.10. Diagrams.net

diagrams.net (antes conocido como *draw.io*) es una herramienta en línea para la **creación de diagramas y gráficos** [18]. Ofrece una amplia gama de plantillas y herramientas para diseñar diagramas de flujo, diagramas de redes, organigramas y otros tipos de representaciones visuales. Su interfaz intuitiva permite a los usuarios crear y personalizar diagramas de manera eficiente.

En este proyecto, *diagrams.net* ha sido utilizado para crear diversos diagramas necesarios para el desarrollo y la documentación del proyecto. La herramienta ha facilitado la visualización de conceptos, la planificación de arquitecturas y la representación de procesos. Los diagramas generados con *diagrams.net* han sido una parte integral de la documentación.

4

Especificación y análisis de requisitos

En esta sección se detallan los **requisitos** que guiarán el desarrollo del sistema, organizados en dos categorías principales: funcionales y no funcionales. Los requisitos **funcionales** (tabla 1) describen las capacidades y comportamientos específicos que debe tener el sistema para cumplir con sus objetivos, mientras que los **no funcionales** (tabla 2) abarcan las características de calidad, como rendimiento, usabilidad y seguridad, que condicionan cómo se deben implementar las funciones.

Además, se presentan los **casos de uso** que describen interacciones específicas entre los usuarios y el sistema, proporcionando una visión clara de cómo se comportará la aplicación en diferentes escenarios. Cada caso de uso incluye una descripción detallada con: su descripción, las condiciones previas, los posibles escenarios alternativos, etc. Esto no solo garantiza una comprensión integral de los requisitos, si no que también facilita la planificación y validación del sistema durante su desarrollo.

4.1. Requisitos

4.1.1. Requisitos funcionales

Cuadro 1: Requisitos funcionales

Identificador	Nombre	Descripción
RF01	Mostrar un histórico de tráfico	El usuario podrá seleccionar el tráfico de una fecha del pasado para mostrarlo en el mapa

Identificador	Nombre	Descripción
RF02	Información de nodos y aristas	El usuario podrá visualizar la información del último nodo o arista sobre el que haya pulsado
RF03	Cambio de estilo del grafo	El usuario podrá cambiar algunas opciones de visualización tanto de los nodos como de las aristas del grafo
RF04	Filtrado de carreteras	El usuario podrá filtrar las carreteras mostradas en el mapa con una serie de filtros habilitados
RF05	Procesado de información	El usuario podrá procesar la información con los filtros establecidos y generar gráficas con los resultados
RF06	Realizar simulaciones	El usuario podrá realizar simulaciones del flujo de tráfico con agentes gracias a una serie de configuraciones disponibles
RF07	Alternar tráfico	El usuario podrá elegir que se muestre en el mapa el tráfico resultante la simulación o el tráfico de la fecha seleccionada
RF08	Añadir resultados en simulaciones	Conforme el usuario ejecute diferentes simulaciones, se incluirán los resultados en las gráficas
RF09	Descargar resultados	El usuario podrá descargarse los resultados de la última información procesada o de la última simulación realizada
RF10	Descargar gráficas	El usuario podrá descargarse las gráficas generadas
RF11	Notificaciones	Cuando se procesen los datos o acabe la simulación, el usuario verá un mensaje de notificación

4.1.2. Requisitos no funcionales

Cuadro 2: Requisitos no funcionales

Identificador	Nombre	Descripción	Tipo
RNF01	Formato de los datos	Los datos de tráfico extraídos del proveedor deberán ser procesados a un formato matemático operable (estructura de grafo)	Interoperabilidad de los datos
RNF02	Procesamiento de datos en el <i>dashboard</i>	El usuario debe ser capaz de procesar información relativa a cuatro año en menos de 30 minutos	Rendimiento
RNF03	Desarrollo	El proyecto deberá desarrollarse en Python	Producto
RNF04	Formato de descarga	Los resultados descargados por el usuario estarán en formato CSV	Interoperabilidad de los datos
RNF05	Grafos completos	Todas las aristas de los grafos almacenados en la base de datos deberán tener información de tráfico	Integridad de los datos

4.2. Casos de uso

Cuadro 3: Caso de Uso RF01: Mostrar tráfico del pasado

Título	Mostrar un histórico del tráfico
Descripción	El usuario podrá seleccionar el tráfico de una fecha del pasado para mostrarlo en el mapa.
Pre-condición	El usuario ha accedido a la interfaz principal y se han subido los datos a la base de datos.

Post-condición	El tráfico de la fecha seleccionada se muestra en el mapa.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario selecciona un rango de fecha con un calendario. 2. El sistema carga las instancias en las que hay información de tráfico en el rango de fecha seleccionado por el usuario dentro de un desplegable. 3. El usuario elige la fecha y la hora de tráfico que quiere cargar del desplegable. 4. El sistema carga y muestra el tráfico correspondiente a esa fecha en el mapa.
Escenarios alternativos	<ul style="list-style-type: none"> ■ 2.a. Si no hay datos disponibles para el rango seleccionado el sistema no carga ninguna instancia dentro del desplegable.
Diagrama de secuencia	<pre> sequenceDiagram participant User participant App participant MongoDB User->>App: open web activate App User->>App: select filters activate App App->>MongoDB: get available dates with traffic info in calendar range activate MongoDB MongoDB-->>App: deactivate MongoDB App-->>User: update dates dropdown deactivate App User-->>App: update screen deactivate User App-->>User: select date deactivate App User-->>App: update map deactivate User </pre> <p>The sequence diagram illustrates the interaction between three components: User, App, and MongoDB. It begins with the User opening a web application. The User then selects filters, which triggers an action in the App component. The App sends a request to the MongoDB database to retrieve available dates with traffic information within the selected calendar range. The MongoDB database returns the data, which the App uses to update a dropdown menu. Subsequently, the User performs an action that triggers an update screen in the App. The App then sends a 'select date' message back to the User, and finally, the User triggers an 'update map' action, which is processed by the App.</p>

Cuadro 4: Caso de Uso RF02: Información de nodos y aristas

Título	Información de nodos y aristas
Descripción	El usuario podrá visualizar la información del último nodo o arista sobre el que haya pulsado.
Pre-condición	El usuario está en la interfaz principal y el mapa ya ha cargado.
Post-condición	Se muestra la información detallada del nodo o arista seleccionado en una tarjeta de la interfaz.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario hace clic en un nodo o arista en el mapa. 2. El sistema carga la información del elemento pulsado en la tabla correspondiente y genera una o dos aristas representando dicha carretera.
Escenarios alternativos	<ul style="list-style-type: none"> ■ 2.a. El usuario hace clic en un área sin nodos ni aristas. ■ 3.a. El sistema no carga información en ninguna tabla.

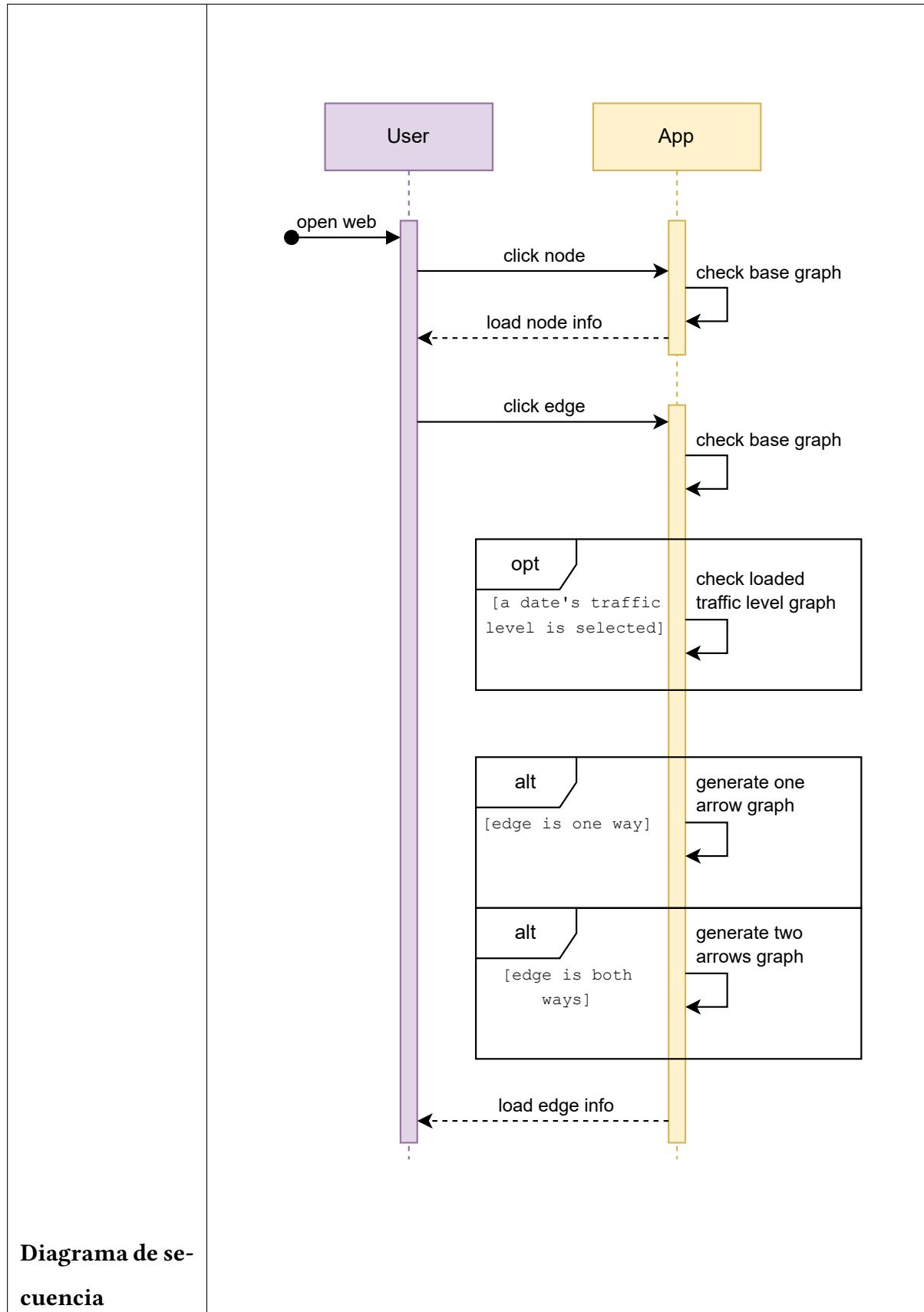


Diagrama de se-
cuencia

Cuadro 5: Caso de Uso RF03: Cambio de estilo del grafo

Título	Cambio de estilo del grafo
Descripción	El usuario podrá cambiar algunas opciones de visualización tanto de los nodos como de las aristas del grafo.
Pre-condición	El usuario está en la interfaz principal y el mapa ya ha cargado.
Post-condición	El estilo del grafo se actualiza según las opciones seleccionadas.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario accede a las opciones de visualización del grafo. 2. El usuario selecciona las opciones deseadas para nodos y aristas (coloreado, grosor, transparencia, ocultar, etc.). 3. El sistema aplica los cambios y actualiza la visualización del grafo.

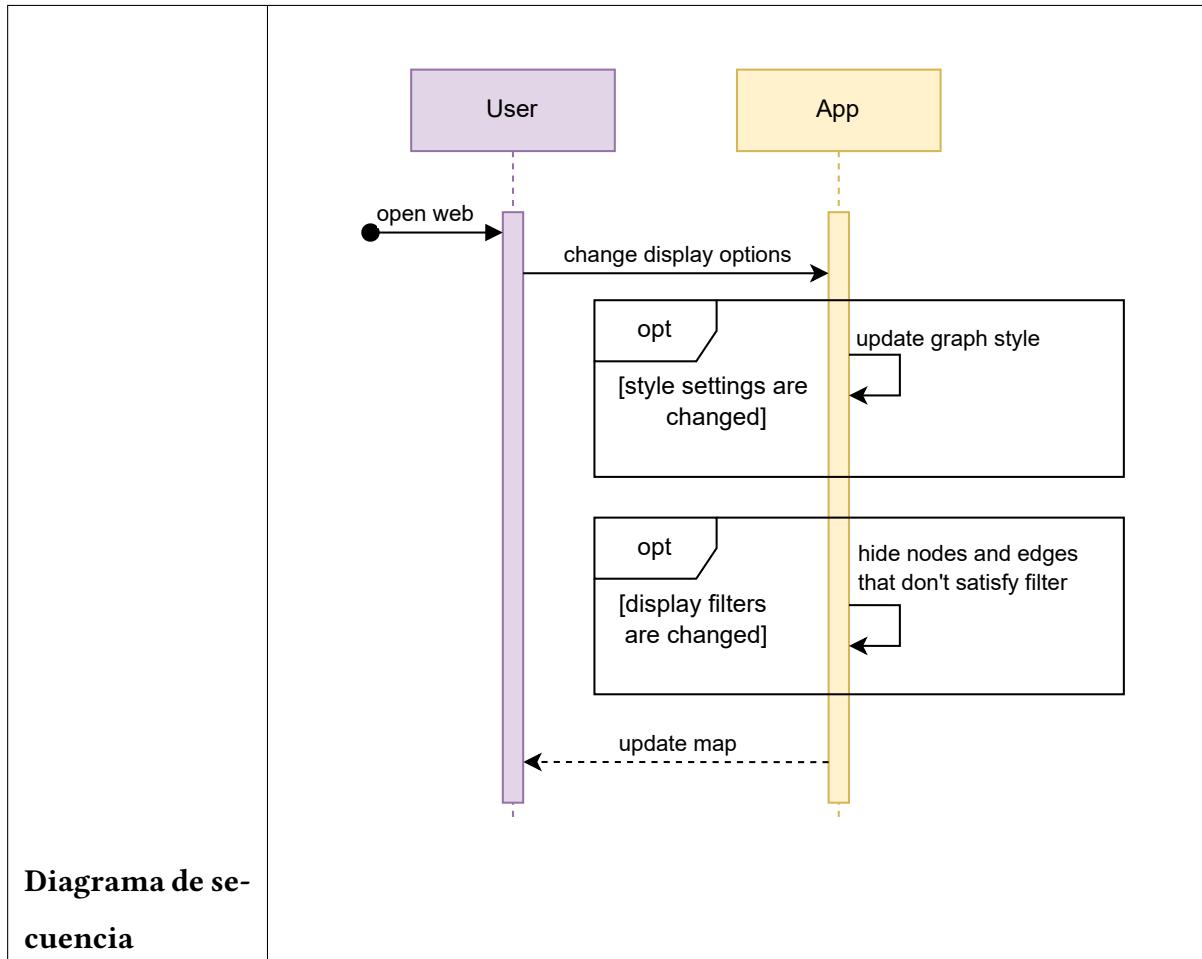


Diagrama de secuencia

Cuadro 6: Caso de Uso RF04: Filtrado de carreteras

Título	Filtrado de carreteras
Descripción	El usuario podrá filtrar las carreteras mostradas en el mapa con una serie de filtros habilitados.
Pre-condición	El usuario está en la interfaz principal y el mapa ya ha cargado.
Post-condición	El sistema muestra solo las carreteras que cumplen con los filtros seleccionados.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario accede a la opción de filtros. 2. El usuario selecciona uno o más filtros para las carreteras. 3. El sistema aplica los filtros y actualiza el mapa mostrando solo las carreteras correspondientes.
Escenarios alternativos	<ul style="list-style-type: none"> ■ 2.a. El usuario elimina todos los filtros. ■ 3.a. El sistema devuelve el grafo original sin filtrar. ■ 3.b. El proceso de filtrado no ha arrojado ninguna carretera, por lo que el sistema oculta el grafo y muestra el mapa vacío.

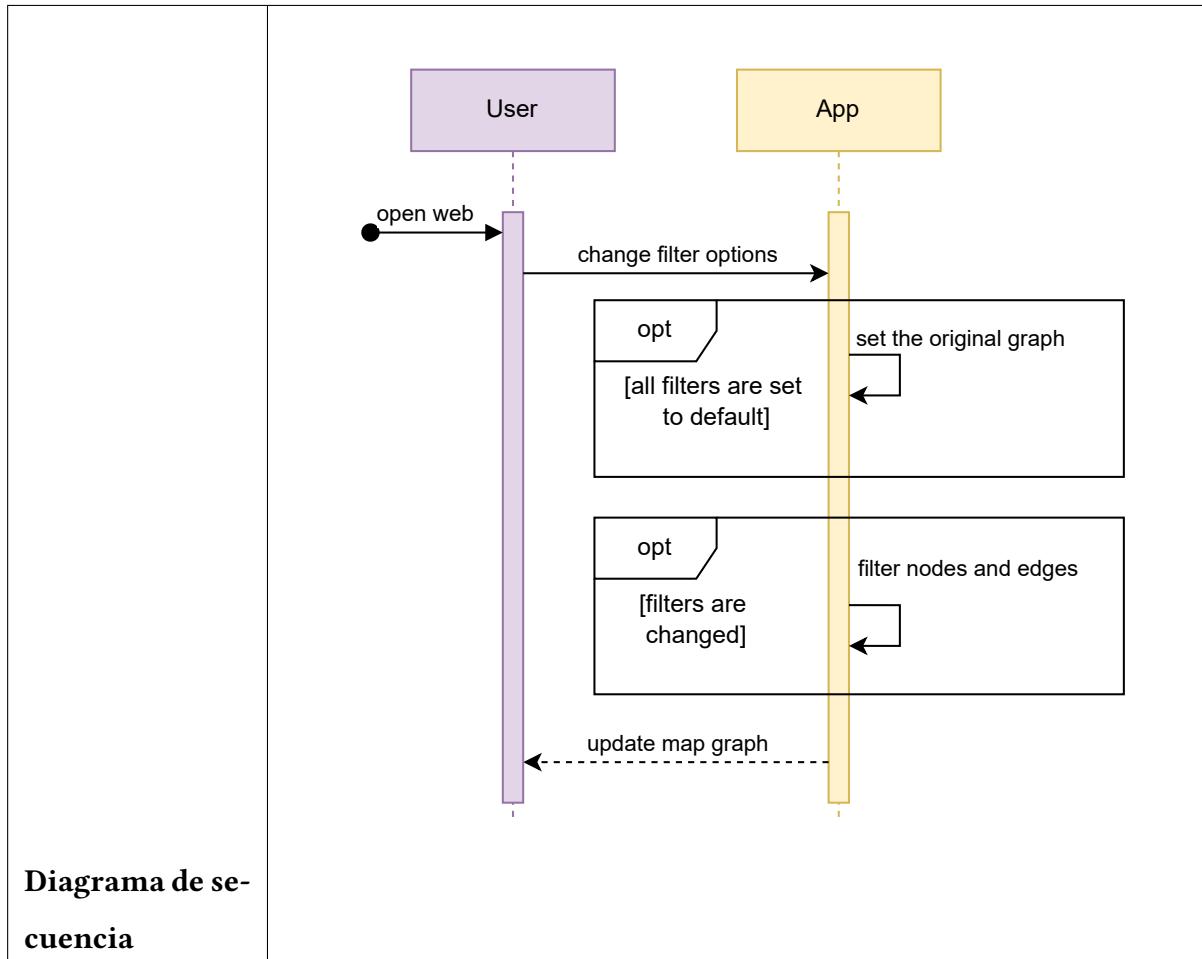
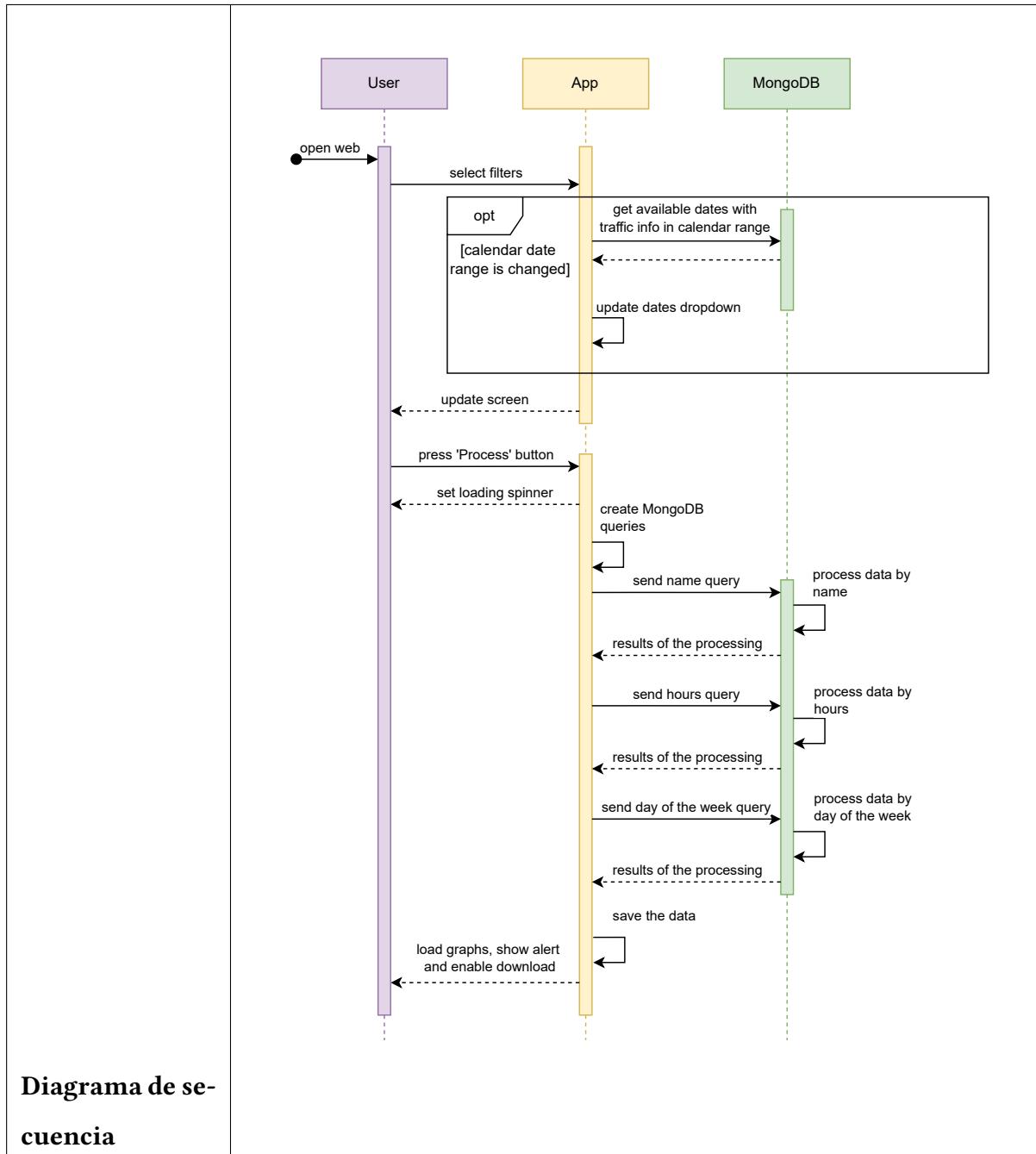


Diagrama de secuencia

Cuadro 7: Casos de Uso RF05 y RF11: Procesado de información

Título	Procesado de información y Notificación
Descripción	El usuario podrá procesar la información con los filtros establecidos y generar gráficas con los resultados y recibirá una notificación cuando el procesamiento esté completado.
Pre-condición	El usuario está en la interfaz principal.
Post-condición	El sistema muestra las gráficas generadas basadas en los filtros aplicados y habilita los botones de descarga de resultados.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario selecciona el rango de fechas deseado. 2. El usuario selecciona el rango de horas que le interesa. 3. Si el usuario lo desea, filtra las calles cuya información quiere procesar. 4. El usuario pulsa sobre la opción «<i>Process</i>». 5. El sistema muestra un <i>spinner</i> de carga. 6. El sistema carga las gráficas con los resultados y habilita los botones de descarga. 7. El sistema muestra una notificación al usuario para avisar.
Escenarios alternativos	<ul style="list-style-type: none"> ■ 4.a. Si no hay datos para procesar, el sistema muestra las gráficas vacías.



Cuadro 8: Casos de Uso RF06 y RF11 : Realizar simulaciones y Notificación

Título	Realizar simulaciones y Notificación
Descripción	El usuario podrá realizar simulaciones del flujo de tráfico con agentes gracias a una serie de configuraciones disponibles.
Pre-condición	El usuario está en la interfaz principal y ha accedido a la opción de simulación.
Post-condición	El sistema guarda los ajustes de la simulación y la ejecuta, cuando termine mostrará una notificación y habilitará los botones de descarga de los resultados y cargará las gráficas generadas .
Escenario principal	<ol style="list-style-type: none"> 1. El usuario accede a la opción de simulación y configura los ajustes. 2. El usuario pulsa sobre el botón «<i>Start simulation</i>». 3. El sistema guarda las configuraciones seleccionadas. 4. El sistema ejecuta la simulación basada en los ajustes aplicados mostrando al usuario una barra de progreso conforme los pasos van ejecutándose. 5. Una vez terminada la simulación, el sistema carga las gráficas con los resultados y habilita los botones de descarga. 6. El sistema muestra una notificación al usuario para avisar.

Escenarios alternativos	<ul style="list-style-type: none"> ■ 7.a. El usuario accede a la opción de simulación y configura los ajustes. ■ 8.a. El usuario pulsa sobre el botón «<i>Start simulation</i>». ■ 9.a. El sistema guarda las configuraciones seleccionadas. ■ 10.a. El sistema ejecuta la simulación basada en los ajustes aplicados mostrando al usuario una barra de progreso conforme los pasos van ejecutándose. ■ 11.a. Una vez terminada la simulación, el sistema carga las gráficas con los resultados y habilita los botones de descarga. ■ 12.a. Vuelta al punto 6. ■ 3.b. La configuración del usuario es incorrecta o incompleta y el sistema muestra un mensaje de error avisando al usuario.
--------------------------------	--

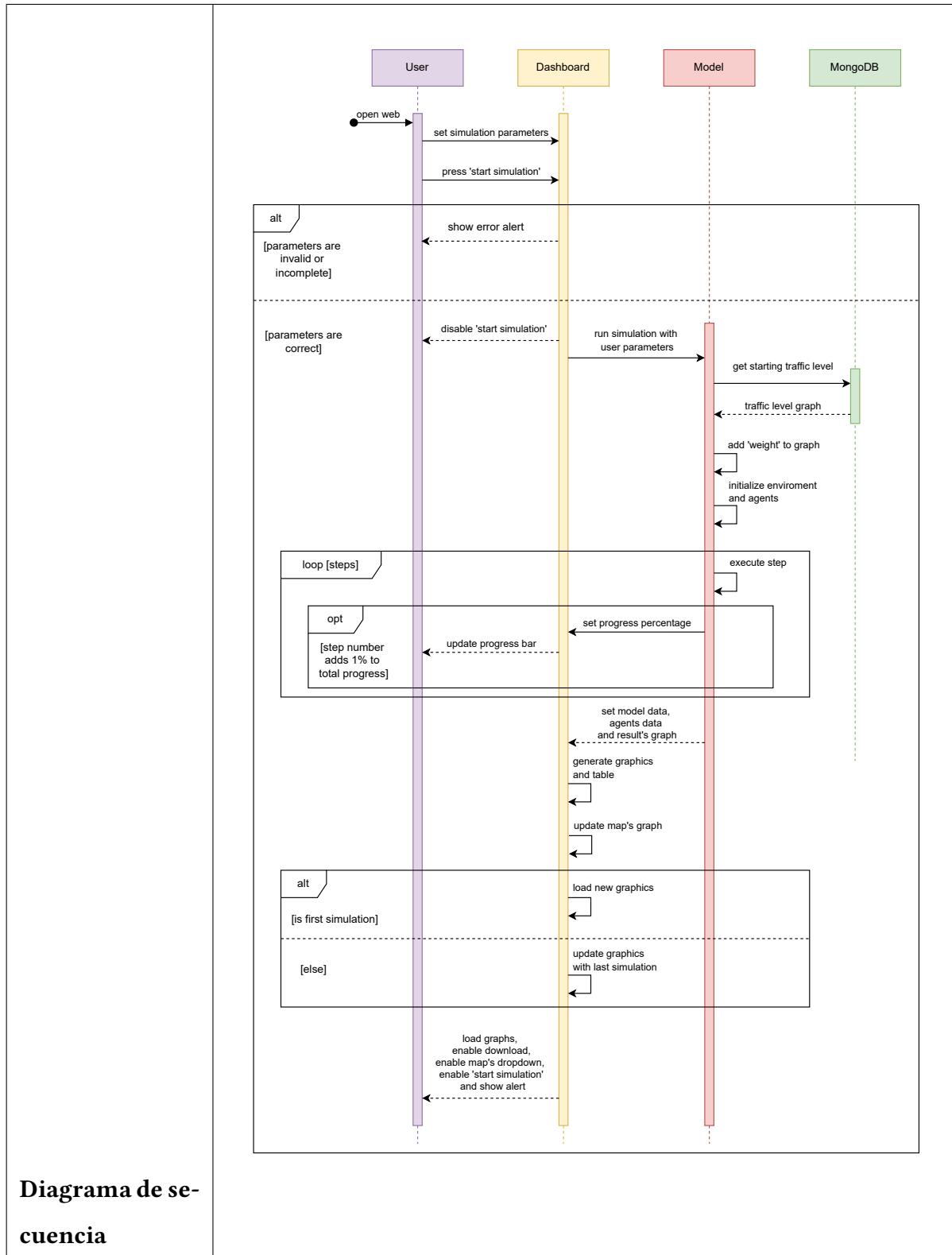
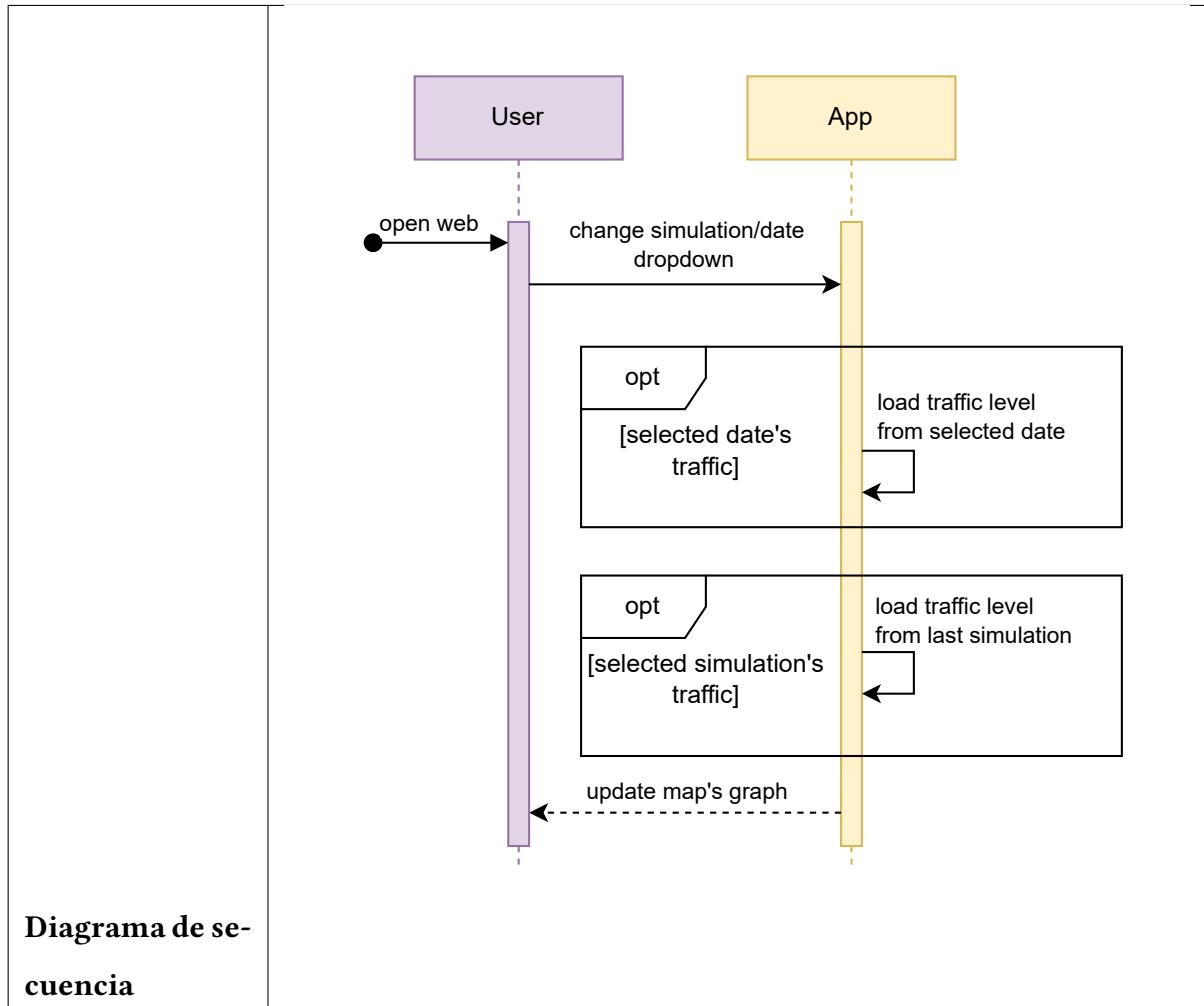


Diagrama de se-
cuencia

Cuadro 9: Caso de Uso RF07: Alternar tráfico

Título	Alternar tráfico
Descripción	El usuario podrá elegir que se muestre en el mapa el tráfico resultante de la simulación o el tráfico de la fecha seleccionada.
Pre-condición	El usuario ha accedido a la visualización del mapa, y ya ha realizado previamente una simulación y esta ha terminado.
Post-condición	El sistema muestra en el mapa el tipo de tráfico seleccionado por el usuario.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario selecciona en un desplegable si quiere visualizar el nivel de tráfico de la simulación o de la fecha seleccionada. 2. El sistema actualiza el grafo conforme a la selección del usuario.
Escenarios alternativos	<ul style="list-style-type: none"> ■ 1.a. Si no hay simulación previa, el desplegable está inhabilitado y cargará automáticamente el tráfico de la fecha seleccionada.



Cuadro 10: Caso de Uso RF09: Descargar resultados

Título	Descargar resultados
Descripción	El usuario podrá descargarse los resultados de la última información procesada o de la última simulación realizada.
Pre-condición	El usuario ha realizado una simulación o procesado información previamente.
Post-condición	El usuario descarga los resultados seleccionados en formato CSV
Escenario principal	<ol style="list-style-type: none"> 1. El usuario selecciona la opción de descarga que desea. 2. El sistema transforma los datos a formato CSV y genera el fichero para el usuario. 3. El usuario comienza la descarga.
Diagrama de secuencia	<pre> sequenceDiagram participant User participant App User->>App: open web activate User User->>App: press any download option deactivate User activate App App->>User: start downloading App->>App: transform data to CSV App->>App: generate file </pre> <p>The sequence diagram illustrates the interaction between the User and App components. It begins with the User opening a web browser. Subsequently, the User presses any download option. This triggers a series of internal events within the App component: it transforms the data into CSV format and generates a file. Finally, the App sends a 'start downloading' signal back to the User.</p>

Cuadro 11: Caso de Uso RF10: Descargar gráficas

Título	Descargar gráficas
Descripción	El usuario podrá descargarse las gráficas generadas.
Pre-condición	El usuario ha realizado una simulación o procesado información previamente y por lo tanto, hay gráficas generadas.
Post-condición	El usuario descarga la gráfica seleccionada en formato <i>PNG</i> .
Escenario principal	<ol style="list-style-type: none"> 1. El usuario, dentro de la gráfica que desea descargar, pulsa el botón habilitado para ello. 2. El sistema transforma genera el archivo <i>PNG</i> para el usuario. 3. El usuario comienza la descarga.
Diagrama de secuencia	<pre> sequenceDiagram participant User participant App User->>User: open web activate User User->>App: press download as image icon in graph activate App App->>User: start downloading deactivate User App-->>App: transform dynamic graph into PNG App-->>App: generate file deactivate App </pre> <p>The sequence diagram illustrates the process. It begins with the User opening a web browser. The User then presses a 'download as image' icon on a graph. This triggers a series of internal events within the App component: it transforms the dynamic graph into a PNG file and generates a file. Finally, the App sends a 'start downloading' signal back to the User.</p>

5

Extracción de la información

5.1. Datos necesarios

Antes de comenzar a buscar proveedores de datos de tráfico o navegación, es necesario realizar un estudio previo de los datos que necesitamos, para poder comparar objetivamente los diferentes proveedores y ayudarnos a hacer una elección o a combinarlos.

Como se detallaba en los objetivos del proyecto, modelaremos una sección del tráfico urbano de Teatinos. Por lo tanto, se necesitan los datos relativos a una zona de aproximadamente 4km^2 , tal y como podemos ver en la fig. 5.

Sin entrar dentro de proveedores individuales, tenemos una serie de servicios comunes ofrecidos generalmente por todos, y que proporcionarán información en tiempo real, los más importantes y que nos servirían para un proyectos de estas características serían:

- **Incidentes en el tráfico:** este servicio nos ofrece información de las diferentes alteraciones o problemas que pueda haber en las carreteras en tiempo real, dandonos localizaciones (accidentes de coche, objetos peligrosos en la carretera ...) o tramos (atascos, carreteras en obras).
- **Información detallada de las intersecciones:** el siguiente servicio nos ofrece información detallada de las intersecciones que pidamos, este servicio es bastante limitado a la zona en que se pida, ya que no habrá información disponible en cualquier intersección. Pero podríamos ver datos como la cantidad de vehículos que giran en una u otra dirección, y datos acerca de las vías que se unen (semáforos, señales de tráfico, longitud)
- **Rutas:** las rutas dinámicas en función del tráfico es un servicio comúnmente ofrecido por este tipo de proveedores, por la utilidad que tienen para diferentes sectores como

Measure an Area

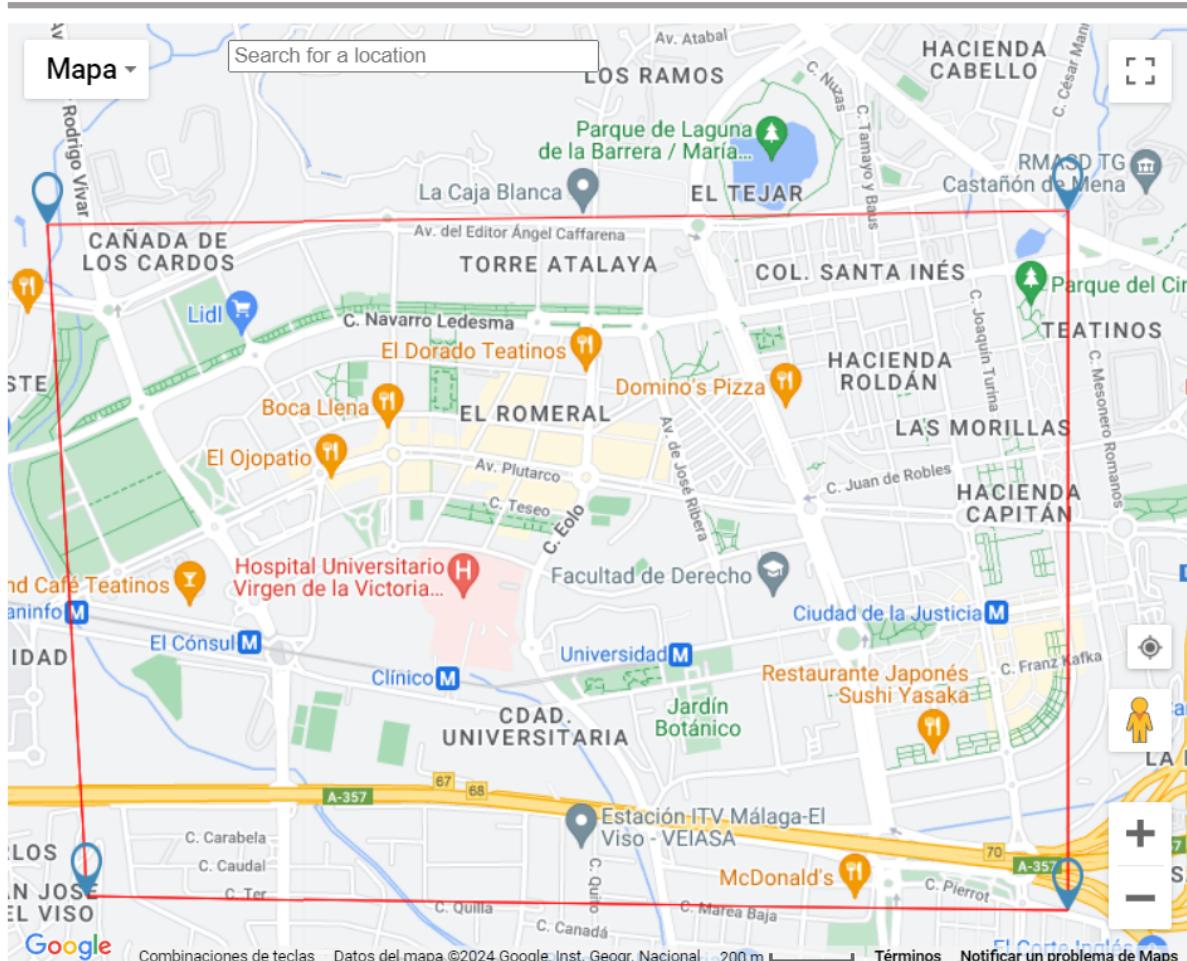


Figura 5: Medida del área utilizando la plataforma *FreeMapTools*.

transportistas, transporte público, taxis, VTCs ...

- **Flujo de tráfico:** finalmente, se nos ofrece información de la congestión que tienen las diferentes carreteras, y cual es la velocidad actual en los diferentes tramos de la carretera en comparación a la velocidad máxima que se puede alcanzar en dichas vías.

La información del flujo de tráfico es la más importante de todas para este proyecto, ya que podremos recogerla y almacenarla con el paso del tiempo, con una frecuencia determinada, y que podremos trasladar posteriormente al modelo, por lo tanto, es la que recogeremos.

Este servicio de flujo de tráfico puede pedirse de dos formas distintas, cada una con sus ventajas y desventajas:

- **Información de una vía:** Podemos pedir información del tráfico de una sola vía, esto es útil si queremos información detallada de una cantidad pequeña de calles, que podría complementarse con la de las intersecciones. Esto sería muy útil para un modelo a escala microscópica, pero no es el caso que nos ocupa.
- **Tiling:** También podemos pedir información de determinados *tiles* (azulejos) del mapa, que dependiendo del nivel de zoom puede abarcar entre 10 y 50 calles, esto nos ahorraría reducir considerablemente la cantidad de peticiones, con lo que además de reducir la carga de trabajo, nos permitiría consumir más peticiones para obtener la información con más frecuencia o en una zona más grande gastando lo mismo que de la otra forma. El problema de este método es el formato de salida:

- **Raster Flow:** Tener una imagen en formato **PNG** del tráfico en un determinado instante tal y como podemos ver en la fig. 6, puede ser útil a la hora de ofrecer ciertos servicios, pero para nuestro proyecto, donde necesitamos trabajar esta información y tener cierta precisión en los datos recibidos y la ubicación de los mismos, necesitaría un post-procesado demasiado complejo y aun así no aportaría buenos resultados.
- **Vector Flow:** El formato de salida de este tipo de peticiones es **PBF (Protobuf Binary Protocol)**, que es una alternativa al formato XML, diseñado para ser más eficiente en términos de tamaño y velocidad. Este formato, comparado con un archivo comprimido con gzip, ocupa aproximadamente la mitad y la escritura es

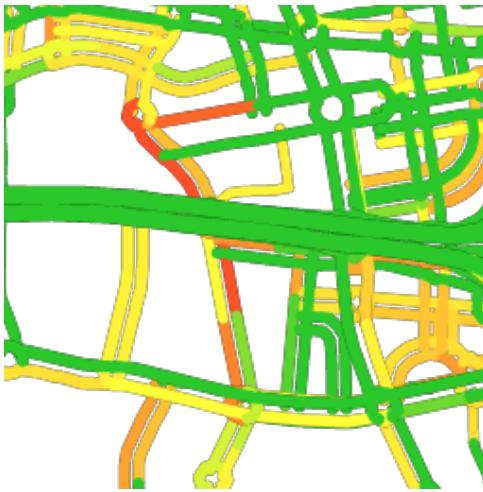


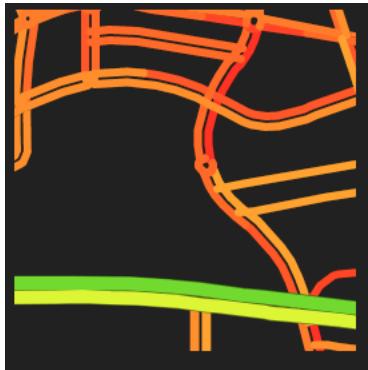
Figura 6: Flujo de tráfico en una zona de Teatinos en formato PNG.

aproximadamente 5 veces más rápida y la lectura 6 más. Los Protocol Buffers (producto de Google) minimizan el tamaño del archivo utilizando una codificación de bits variable para enteros. Y una vez decodificado tendremos un archivo **geoJSON**, mucho más manejable y entendible que una imagen.

Por lo tanto, la información que solicitaremos al proveedor será del tipo ***Vector Flow Tiling***, y tras un post-procesado seremos capaces de manejar grandes volúmenes de datos con los que trabajar en las próximas fases del proyecto. Aunque el formato tipo ***Raster Flow Tiling*** nos será muy útil para asegurar que cubrimos todas las zonas deseadas del mapa.

Las peticiones con este formato tienen una característica que se puede modificar en función del proyecto que se esté realizando. Podemos pedir la información del tráfico además en dos formatos posibles. En formato absoluto, donde se nos enviará la velocidad máxima a la que se puede circular por la vía. Y en formato relativo, donde se indica con un valor en el rango $[(0,0), (1,0)]$, siendo 0,0 el peor caso posible, donde el tráfico está completamente colapsado y la **velocidad máxima** es de 0 km/h y 1,0 es la situación ideal, donde los vehículos pueden transitar a la **velocidad máxima** que permita la vía. Gracias al servicio de *Raster Flow Tiling* podemos visualizar la diferencia (fig. 7).

A continuación debemos estudiar cuál es la mejor opción a la hora de seleccionar los *tiles* a los que nos suscribiremos, de forma que la información que obtengamos abarque toda la zona elegida de Teatinos. Gracias a una herramienta en línea podemos visualizar con precisión que zonas elegiremos (fig. 8)



(a) Tráfico absoluto



(b) Tráfico relativo

Figura 7: Comparación de los formatos del nivel de tráfico de *Raster Flow Tiling* en la zona del Hospital Clínico

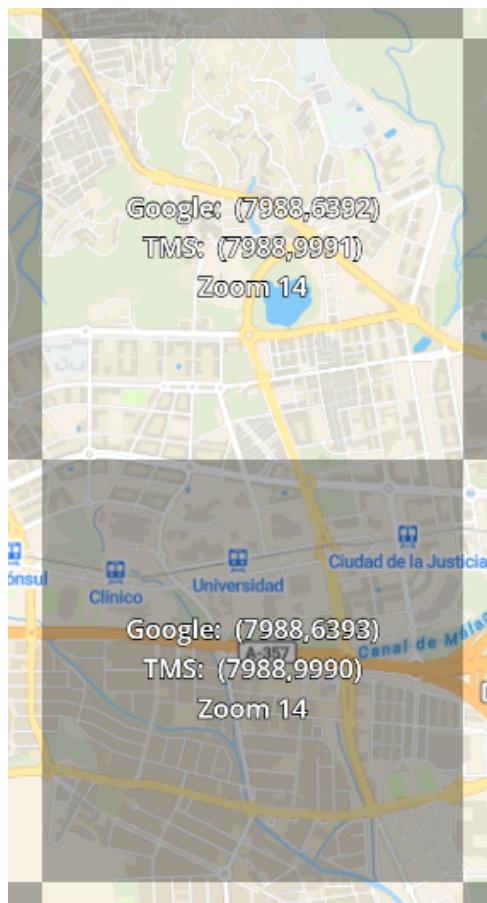


Figura 8: *Tiles* necesarios en mapa de *MapTiler*.

5.2. Evaluación de proveedores

Una vez que se tenga claro cuál es el/los tipo de servicio que requerimos para el proyecto, podemos hacer una **comparación objetiva** de los proveedores de dicho servicio. Los tres más utilizados comúnmente son: Azure Maps de Microsoft, TomTom, y HERE Technologies.

En primer lugar, podemos descartar el servicio de HERE, ya que el formato que devuelve cuando hacemos peticiones por *tiling*, es PNG u OVM (Optimized Map for Visualization), y este formato no es convertible a un formato tipo JSON donde podamos utilizar la información. Por lo tanto, HERE queda directamente descartada.

Solo nos quedaría estudiar TomTom y Azure Maps, haciendo una comparación de cada una de las partes:

- **Comodidad:** TomTom es más sencillo, ya que cuando se crea una cuenta, se aporta una API KEY con la que podemos empezar a hacer peticiones. Mientras que con Azure, una vez creada la cuenta, tenemos que iniciar un nuevo grupo de recursos y posteriormente iniciar una cuenta de Azure Maps, que nos dará la clave con la que hacer peticiones.
- **Precio:** El precio es un gran factor condicionante para este proyecto, para una gran escala de peticiones (cuando empiece a superar el medio millón) es más barato usar Azure, pero para un proyecto como este, es mucho más interesante usar TomTom, ya que nos ofrece 50.000 peticiones diarias gratuitas. Mientras que con Azure, se nos cobraría desde el primer momento, aunque por ser estudiante se podría conseguir un crédito de hasta 200€.
- **Calidad de la información:** La información proporcionada por ambos servicios es la misma, ambos devuelven la misma información cuando se pide en el mismo instante, de hecho, las imágenes formato PNG aportadas por ambos servicios siguen el mismo estilo y son indistinguibles.

Finalmente, la decisión tomada ha sido la de implementar el sistema de extracción de datos con el proveedor de servicios TomTom, que nos brindará la posibilidad de hacer una cantidad de peticiones diarias gratuitas que no superaremos en ninguno de los casos.

5.3. Extracción de datos

Una vez que tenemos la zona elegida (y concretamente los tiles), el servicio y el proveedor, podemos empezar a **planificar la extracción** de los datos que realizaremos para tener una fuente de información consistente para el modelo. Ya que, es imposible obtener el histórico del flujo de tráfico a partir de ningún proveedor, tendremos que almacenar nosotros la información de la serie temporal.

Teniendo en cuenta la cantidad de información que podemos llegar a almacenar y a gestionar, tenemos que **estudiar cada cuanto reclamaremos la información** al proveedor y durante cuanto tiempo. Recopilaremos información del tráfico diariamente durante aproximadamente 6 meses, por lo tanto, teniendo en cuenta que solicitamos a dos *tiles*, tendremos una cantidad cercana a $ficheros = (30 * 6) * (\frac{24*60}{i} * 2) = 180 * \frac{2880}{i}$, siendo i los minutos que transcurran entre peticiones al proveedor.

Para este proyecto, se ha considerado adecuado dejar una franja de tiempo de **30 minutos**, ya que, tras revisar diferentes intervalos, es el tiempo perfecto para poder visualizar cómo se desplazan los focos con mayor flujo de tráfico a lo largo de la zona de Teatinos sin tener que almacenar información extra que no nos sea de utilidad. Por lo tanto, al final del proceso de recolección de información tendremos aproximadamente 17.280 ficheros de información del tráfico de Teatinos.

Finalmente, queda desarrollar un módulo que extraiga la información. Necesitará estar ejecutándose las 24 horas del día en una máquina, y cada 30 minutos hará las dos peticiones correspondientes a TomTom para recopilar la información necesaria, tal y como podemos ver en el diagrama de secuencia de la fig. 9. En la fig. 10, podemos ver una gráfica con esta parte del proceso en relación con las siguientes.

5.4. Post-procesado de la información

5.4.1. Objetivos del post-procesado

Una vez que nuestra aplicación recolectora de información recoja los suficientes datos, y tengamos todos los archivos resultantes organizados y guardados, podemos comenzar con el post-procesamiento de los datos recolectados a lo largo del tiempo.

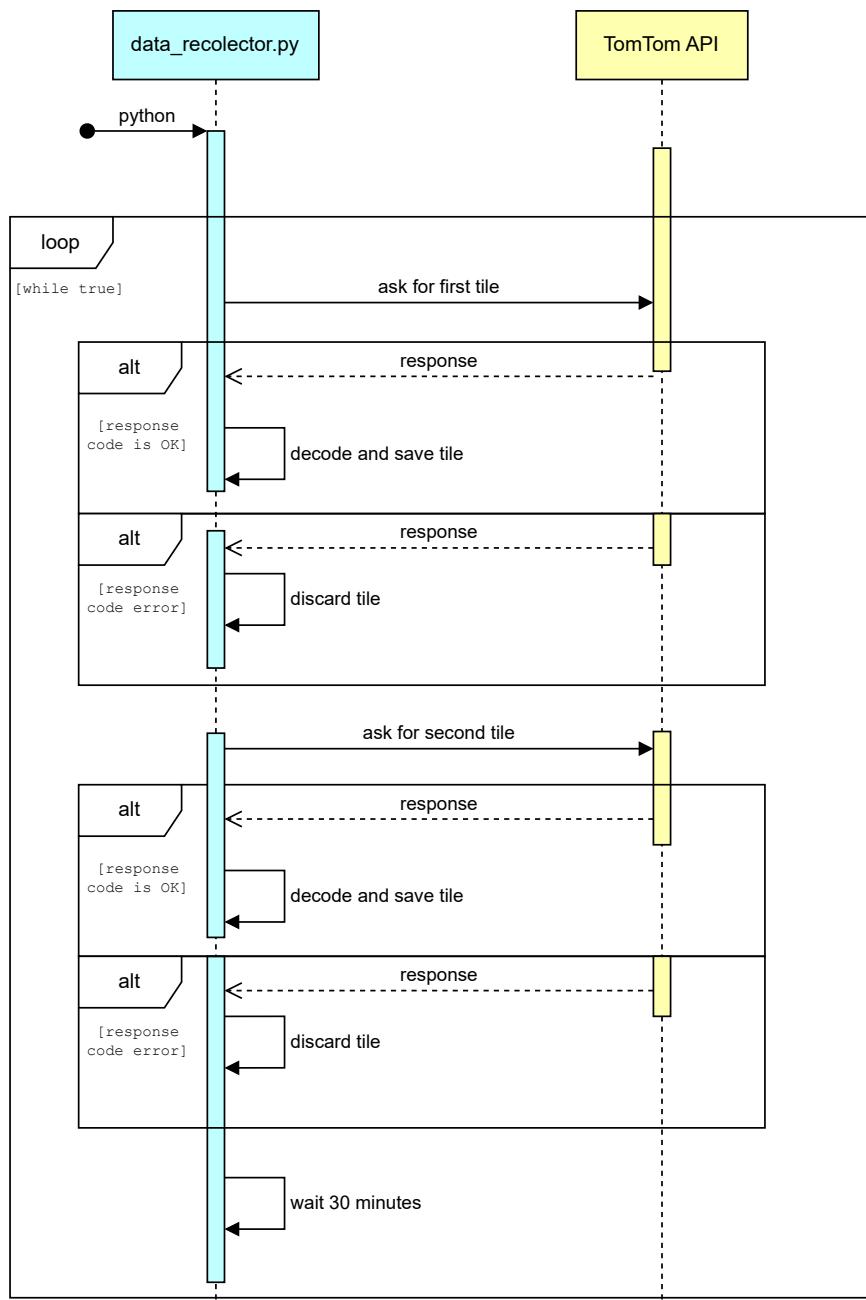


Figura 9: Diagrama de secuencia para la extracción de información.

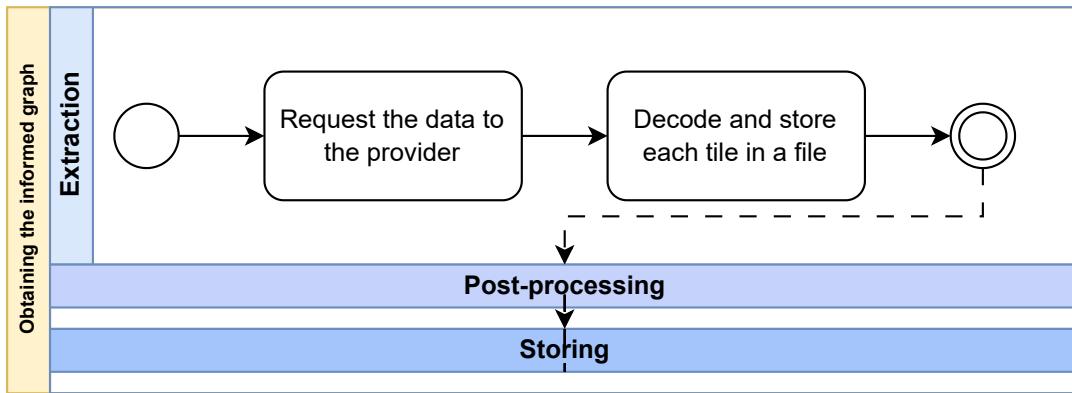


Figura 10: Gráfica con el proceso de extracción de la información del proveedor.

Antes de comenzar, es crucial analizar y definir claramente los **objetivos** posteriores para determinar la dirección que debemos seguir. Por un lado, necesitamos almacenar los datos de manera eficiente para asegurar un funcionamiento óptimo del tablero de datos, lo cual requiere utilizar una **estructura de datos adecuada**. Por otro lado, debemos ser capaces de representar un modelo de la realidad que permita la simulación con agentes.

Estas dos necesidades pueden ser satisfechas utilizando una estructura de tipo **grafo**. Un grafo está compuesto por un conjunto de nodos (vértices o puntos) y aristas (arcos que unen los nodos). Por lo tanto, es esencial tener en mente que todas las transformaciones aplicadas a la información extraída del proveedor deben estar orientadas a su posterior representación en un grafo.

5.4.2. Traducción a GeoJSON

A los resultados proporcionados por el proveedor es necesario aplicar una **decodificación de bytes a texto plano**, lo cual conseguimos con la ayuda de la librería **GDAL**. Esta decodificación produce un archivo GeoJSON que guardaremos localmente para su posterior procesamiento.

Los archivos GeoJSON tienen una estructura bien definida, implementada correctamente en la información enviada por *TomTom* una vez decodificada (fig. 11). Estos archivos están compuestos por una serie de elementos denominados «*features*», que incluyen un campo de propiedades donde se puede añadir información variada, aquí se encontrarían los datos relacionados con el tráfico enviados por TomTom, y que se encuentran recogidos en el diccionario

```
{
  "type": "FeatureCollection",
  "name": "Traffic flow",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "road_type": "Connecting road",
        "traffic_level": 0.2169999272823334,
        "traffic_road_coverage": "full"
      },
      "geometry": {
        "type": "MultiLineString",
        "coordinates": [
          [
            [
              [
                [2252.0, 2194.0],
                [2272.0, 2198.0]
              ]
            ]
          }
        ]
      }
    },
    {
      "type": "Feature",
      "properties": {
        "road_type": "Local road",
        "traffic_level": 0.2969999094009399,
        "traffic_road_coverage": "one_side"
      },
      "geometry": {
        "type": "MultiLineString",
        "coordinates": [
          [
            [
              [
                [1632.0, 2572.0],
                [1678.0, 2476.0]
              ]
            ]
          }
        ]
      }
    }
  ...
  ]
}
```

Figura 11: Ejemplo con dos elementos de un archivo geoJSON de *TomTom*.

de datos de la tabla 12. Cada elemento especifica su tipo en el campo «*geometry*», junto con las coordenadas de su ubicación en formato «RFC7946».

TomTom nos enviará por cada *tile* un archivo GeoJSON con elementos que pueden ser de tipo multilínea, línea y punto. Los elementos multilínea no son más que varias líneas que al estar agrupadas dentro del mismo elemento comparten las propiedades indicadas. En segundo lugar, las coordenadas que indican la ubicación de los elementos no vendrán en un formato estandarizado, ya que utilizará las coordenadas del propio *tile*, ([0, 4095], [0, 4095]), siendo las coordenadas (0, 0) las de la esquina superior izquierda. Por lo tanto, ambos *tiles* tendrán el mismo formato de coordenadas (fig. 12).

Antes de empezar la traducción de coordenadas, necesitamos **estandarizar la información**

Etiqueta	Descripción
<i>road_type</i>	<p>Describe el tipo de carretera que tenemos:</p> <ul style="list-style-type: none"> ▪ Motorway ▪ International road ▪ Major road ▪ Secondary road ▪ Connecting road ▪ Major local road ▪ Local road ▪ Minor local road ▪ Non public road ▪ Parking road
<i>traffic_level</i>	<p>Como la hemos pedido en formato relativo, será un número flotante en el rango $[(0,0), (1,0)]$ como se ha explicado previamente.</p>
<i>traffic_road_coverage</i>	<p>Describe si el tráfico se asigna a toda la carretera full (vía de un solo sentido) o solo a un lado one_side (vía de dos sentidos).</p>
<i>left_hand_traffic</i>	<p>Esta etiqueta no aparece en ninguna de las 400 peticiones, la etiqueta indica si la carretera tiene circulación por la izquierda. Si la etiqueta no está presente, la carretera tiene circulación por la derecha.</p>
<i>road_closure</i>	<p>Esta etiqueta indicará si la carretera se encuentra cerrada (en formato <i>PNG</i> podemos visualizar las vías de color gris).</p>

Cuadro 12: Diccionario de datos de las etiquetas utilizadas en la información de tráfico.

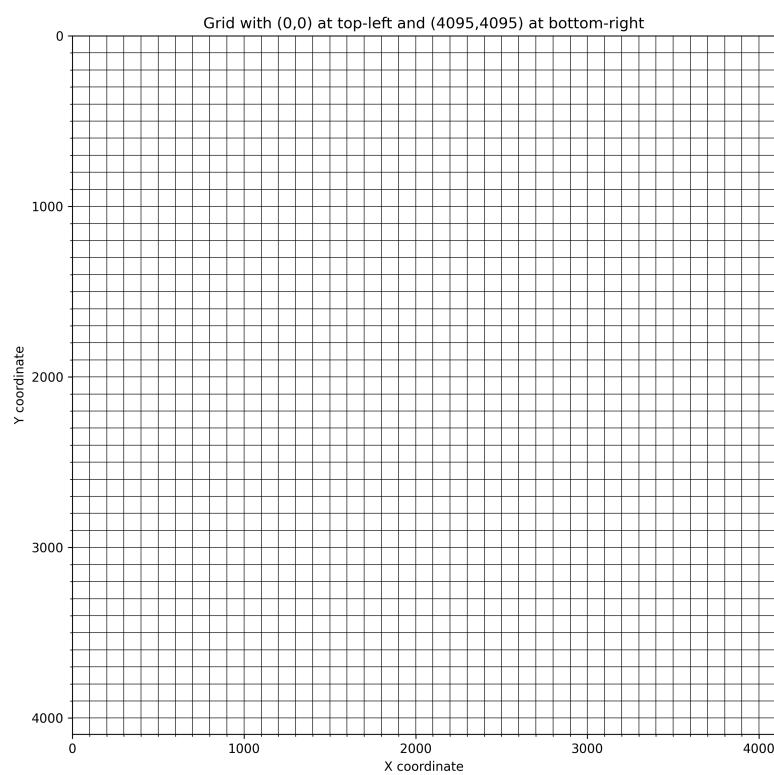


Figura 12: Cuadrícula con las coordenadas de cada *tiles* solicitada.

ción con la que trabajaremos para poder procesarla eficientemente. Para ello convertiremos todos los elementos a líneas compuestas únicamente por dos puntos. Dividiendo las multilíneas en líneas individuales, y todas las líneas a su vez en otras compuestas únicamente por dos puntos, y finalmente, descartar los elementos de tipo punto, ya que, aunque aparezcan con muy poca frecuencia, la información que aportan es irrelevante, y podría incluso llegar a presentar irregularidades.

Con los elementos de los ficheros estandarizados, podemos traducir las coordenadas a valores reales con una interpolación lineal de las coordenadas de cada *tile*, para posteriormente fusionar los dos ficheros en uno. Para ello se utilizará una función de interpolación lineal que transformará un valor x de un rango de entrada $[in_min, in_max]$ a un rango de salida $[out_min, out_max]$. La fórmula utilizada para esta transformación es:

$$y = (x - in_min) \cdot \frac{(out_max - out_min)}{(in_max - in_min)} + out_min \quad (1)$$

donde:

- x es el valor a interpolar, podrá ser longitud o latitud.
- in_min es el valor mínimo del rango de entrada, que será 0.
- in_max es el valor máximo del rango de entrada, que será 4095.
- out_min es el valor mínimo del rango de salida, que será la longitud o latitud real mínima de una de las cuatro esquinas del *tile* que se esté interpolando .
- out_max es el valor máximo del rango de salida, que será la longitud o latitud real máxima de una de las cuatro esquinas del *tile* que se esté interpolando.
- y es el valor interpolado resultante.

Por lo tanto, solamente tendríamos que calcular cuáles son las coordenadas de dichas esquinas de cada *tile*. Para ello, se ha implementado una función que tiene como objetivo obtener las coordenadas de las cuatro esquinas de una *tile* específica dentro de un mapa en el formato «RFC7946», propio de los ficheros GeoJSON.

- **Argumentos:**

- x_tile : La coordenada x del *tile*.
 - y_tile : La coordenada y del *tile*.
 - $zoom$: El nivel de zoom del *tile*.
 - $format$: El formato de las coordenadas. Puede ser *latlng* o *lnglat*.
- **Devuelve:** Una lista con las coordenadas de las esquinas del *tile* en el formato GeoJSON.

Para calcular las coordenadas de las esquinas, la función realiza los siguientes pasos:

1. Encuentra la longitud del lado izquierdo del *tile*:

$$\text{lng_left} = \frac{x_tile \cdot 360}{2^{\text{zoom}}} - 180$$

Esta fórmula convierte la coordenada x del *tile* a una longitud en grados, ajustando según el nivel de zoom. Se multiplica x_tile por 360 grados (la circunferencia completa de la Tierra) y luego se divide por 2^{zoom} , lo que da el tamaño de cada *tile* a ese nivel de zoom. Finalmente, se resta 180 grados para transformar la longitud al rango de -180 a 180 grados, correspondiente al sistema de coordenadas geográficas estándar.

2. Encuentra la longitud del lado derecho del *tile*:

$$\text{lng_right} = \frac{(x_tile + 1) \cdot 360}{2^{\text{zoom}}} - 180$$

Esta fórmula es similar a la anterior, pero usa $x_tile + 1$ para obtener la longitud del borde derecho del *tile*. Esto se debe a que queremos la longitud del siguiente *tile* en la dirección x .

3. Encuentra la latitud del lado superior del *tile* utilizando la función inversa de la proyección de Mercator:

$$\text{lat_top} = \arctan(\sinh(\pi \cdot (1 - \frac{2 \cdot y_tile}{2^{\text{zoom}}}) \cdot \frac{180}{\pi}))$$

Esta fórmula transforma la coordenada y del *tile* en una latitud en grados. La proyección de Mercator mapea coordenadas de latitud y longitud a un plano, y esta fórmula revierte ese mapeo. La función *sinh* (seno hiperbólico) y *arctan* (arcotangente) se utilizan para calcular la latitud correspondiente a la posición del *tile*. La multiplicación por π y división por π ajustan las unidades entre radianes y grados.

4. Encuentra la latitud del lado inferior del *tile* de manera similar:

$$\text{lat_bottom} = \arctan(\sinh(\pi \cdot (1 - \frac{2 \cdot (y_tile + 1)}{2^{\text{zoom}}}) \cdot \frac{180}{\pi}))$$

Esta fórmula sigue el mismo principio que la del lado superior, pero usa $y_tile + 1$ para obtener la latitud del borde inferior del *tile*. De nuevo, esto se ajusta según el nivel de zoom y usa la proyección inversa de Mercator para transformar la coordenada y del *tile* en una latitud geográfica.

Dependiendo del formato especificado (*latlng* o *lnglat*), la función devuelve las coordenadas en el orden adecuado, y si no se especificase un formato válido, la función arrojaría un error:

- Si el formato es *latlng*, las coordenadas se devuelven como:

$$\begin{bmatrix} \text{lat_top}, & \text{lng_left} \\ \text{lat_bottom}, & \text{lng_left} \\ \text{lat_bottom}, & \text{lng_right} \\ \text{lat_top}, & \text{lng_right} \\ \text{lat_top}, & \text{lng_left} \end{bmatrix}$$

- Si el formato es *lnglat*, las coordenadas se devuelven como:

$$\begin{bmatrix} \text{lng_left}, & \text{lat_top} \\ \text{lng_left}, & \text{lat_bottom} \\ \text{lng_right}, & \text{lat_bottom} \\ \text{lng_right}, & \text{lat_top} \\ \text{lng_left}, & \text{lat_top} \end{bmatrix}$$

Podemos visualizar los dos puntos claves, resultantes de cada *tile*, con la máxima y mínima latitud y longitud. Con estos valores seremos capaces finalmente de realizar la traducción correspondientes (fig. 13 y fig. 14).

Una vez que tenemos las coordenadas de cada *tile* interpoladas a las coordenadas reales del mapa, podemos fusionar cada par de ficheros en uno, ya que de esta forma podremos realizar conversiones posteriores de una forma más homogénea y eficiente.

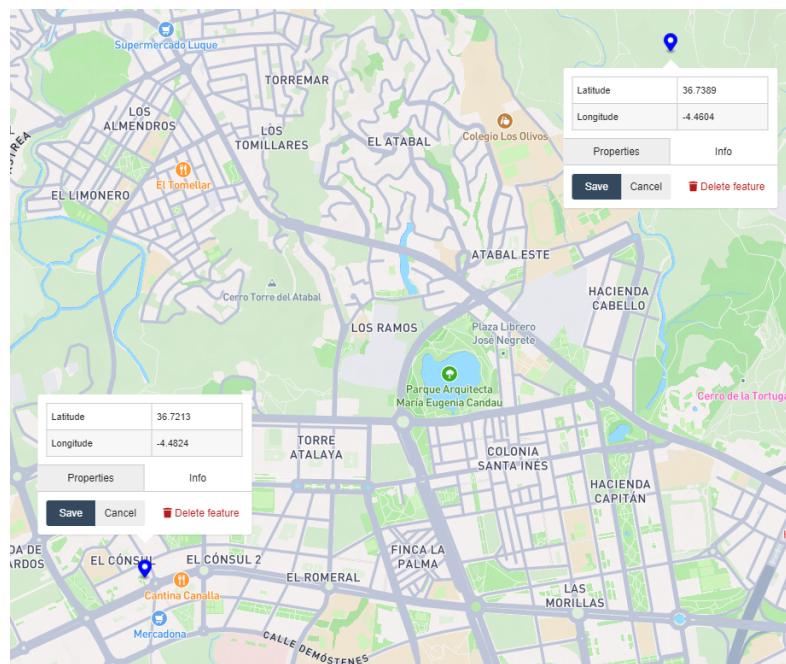


Figura 13: Imagen del mapa del *tile* norte de Teatinos con las longitudes y latitudes mínimas y máximas.

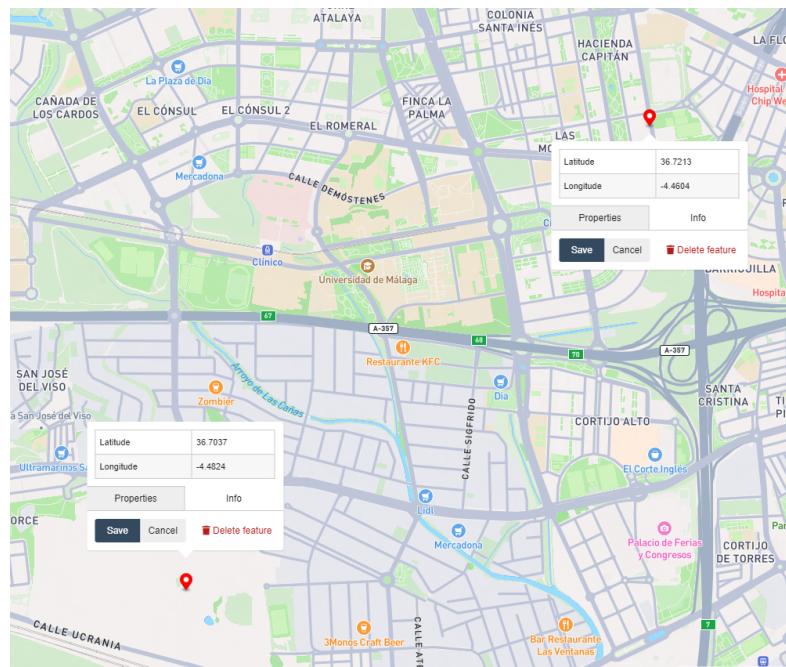


Figura 14: Imagen del mapa del *tile* sur de Teatinos con las longitudes y latitudes mínimas y máximas.



Figura 15: Imagen del grafo generado.

5.4.3. Inclusión de información

Parte de la información que añadiremos la obtendremos del **grafo de la zona**, ya que las coordenadas de los elementos nunca se repiten, y para identificarlos, lo haremos mediante la arista del grafo más cercana, pensando también en el posterior traslado de información a dicha estructura. *Open Street Map* permite obtener el grafo de una zona del mundo de diferentes maneras (nombre, radial a un punto...). Para este proyecto, se ha optado por usar una zona delimitadora (*bounding box*) a partir dos latitudes y dos longitudes que definirán las cuatro esquinas del mapa del que se extraerá el grafo.

Dicha zona será algo mayor que la abarcada por los *tiles*, ya que aunque tengamos aristas que no queden cubiertas directamente por información obtenida del proveedor, es importante tener un grafo con realista, con nodos de entrada y salida claros para pasos posteriores (fig. 15).

El grafo contendrá solo aquellas vías en las que puedan transitar vehículos, ignorando las que sean para peatones y se le añadirá información adicional como la máxima velocidad de las

aristas que podemos consultar, y otra información que podemos extraer de *Open Street Map*, como son la dirección en la que apunta cada arista y las longitudes de las mismas.

Además es muy importante tener en cuenta que obtendremos un grafo, que según la librería *NetworkX* es de tipo *MultiDiGraph*, que no es más que un multigrafo dirigido: un grafo que permite tener múltiples aristas en el mismo par de nodos, lo cual es esencial para las calles con diferentes sentidos.

Esto lo podemos ver de una forma más clara con un ejemplo: tenemos una calle compuesta por dos nodos (A y B) que es de doble sentido. Entonces tendremos dos aristas A→B y B→A, que a la hora de representarlo, se encontrarán superpuestas la una a la otra, pero podremos almacenar información diferente en las aristas.

A continuación añadiremos **información adicional al GeoJSON**, que nos será de gran utilidad para poder encontrar posibles errores y visualizar los resultados de las operaciones realizadas hasta el momento. Esto será muy importante como paso previo al traspaso de información al grafo, para asegurar que se hace correctamente.

- **Longitud del elemento:** gracias a los dos puntos que componen cada línea, seremos capaces de medirla.
- **Arista más cercana:** guardaremos el *OSMID* junto con algunas propiedades más de la arista más cercana del grafo de Teatinos al elemento. Para calcular cuál es la arista más cercana. Lo haremos buscando el punto medio de la línea en el grafo.
- **Vía en otro sentido:** en caso de que la carretera sea en doble sentido, tendremos en cada elemento una referencia a la arista en el sentido contrario al elemento.
- **Orientación:** cada elemento contendrá el valor en grados de la dirección en la que apunta.
- **Opciones de visualización:** aunque para trasladar la información al grafo no sea útil, podremos ver en los mapas los niveles de tráfico con un simple vistazo gracias al gradiante de color de verde a rojo, como se aprecia en la fig. 16

Después de un análisis intensivo de la información añadida a los ficheros y al grafo de la zona, apreciamos la existencia de varios **problemas** que tiene la información proporcionada por TomTom y que hasta ahora no habíamos tratado:

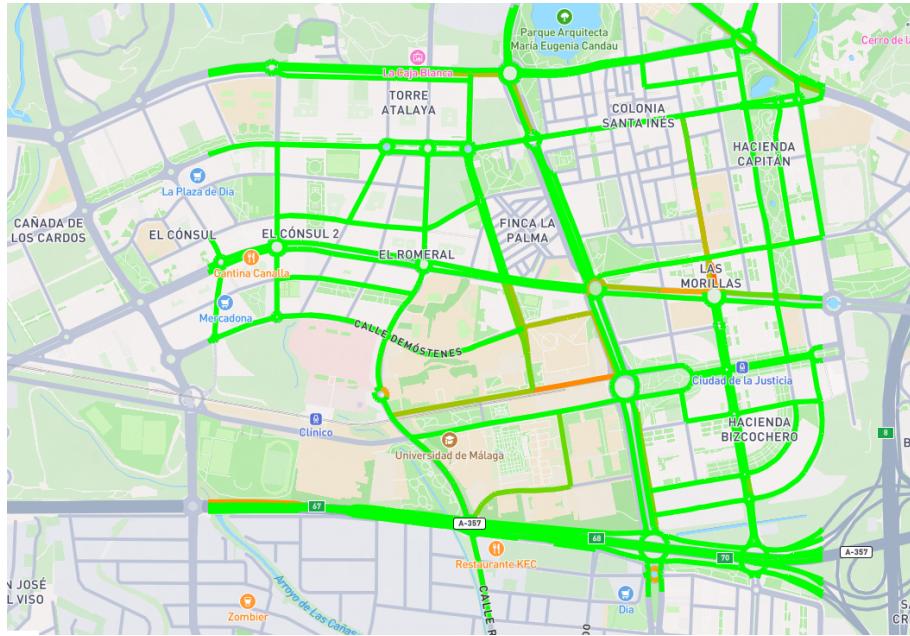


Figura 16: Imagen del mapa con un GeoJSON postprocesado cargado.

- **Longitud de las carreteras:** no hay ningún máximo para la medida de las carreteras. Esto tiene sentido desde el punto de vista del proveedor, ya que busca ahorrar caracteres en el envío, pero para nosotros, que posteriormente deberemos trasladar la información a un grafo con unas aristas de una medida fija y por lo general de pocos metros, no nos interesa tener la información tan poco fragmentada en el *geoJSON*. Por lo tanto, debaremos dividir las líneas para que no haya tramos con una medida superior a una cantidad de metros. Tras hacer algunas pruebas, la mejor medida para conservar la eficiencia del código y evitar la falta de información es de 15 metros.

Como podemos ver en la fig. 17, tenemos aproximadamente un 12 % más de aristas con información del tráfico que el resto de divisiones.

- **Carreteras de doble sentido:** el estándar en los grafos de *Open Street Map* para las carreteras de doble sentido, es utilizar las dos direcciones de una misma arista aunque haya una distancia real de algunos metros entre ambos sentidos. Esto solo supone un problema cuando proveedores como *TomTom* siguen este estándar y los archivos *GeoJSON* proporcionados a los clientes tienen tramos superpuestos y sin identificador. Por lo tanto, no nos serviría con buscar únicamente la arista más cercana al punto central del elemento. Es necesario comprobar que el ángulo con respecto al norte que tiene el ele-



(a) Elementos de 5 metros

(b) Elementos de 10 metros

(c) Elementos de 15 metros

Figura 17: Comparación la información en los grafos en función de la medida de los elementos

mento y la arista del grafo coinciden, para que se asocia el elemento con la arista $A \rightarrow B$ o $B \rightarrow A$. Esto será de una importancia crucial a la hora de trasladar la información al grafo correctamente.

La fig. 18 representa la información que tenemos y que es lo que aspiramos a conseguir: tenemos ambas líneas superpuestas, como refleja la línea azul, y nuestro objetivo es conseguir diferenciar claramente a cuál de los dos sentidos se refiere.

Los dos problemas mencionados se abordarán durante el mismo procesamiento en el que se añade la información. Además de incluir los datos necesarios, se calculará el **número de veces que cada elemento debe dividirse** para que mida 15 metros, asegurando que el punto medio de cada segmento se alinee correctamente con el del grafo. Simultáneamente, se tendrá en cuenta la dirección en la que apuntan los elementos para asociar correctamente la arista del grafo más cercana.

Finalmente, en el último paso del procesamiento de los archivos GeoJSON, nos encargaremos de dividir los elementos las veces calculadas. Podemos visualizar este proceso de traducción y añadido de información en la fig. 19.

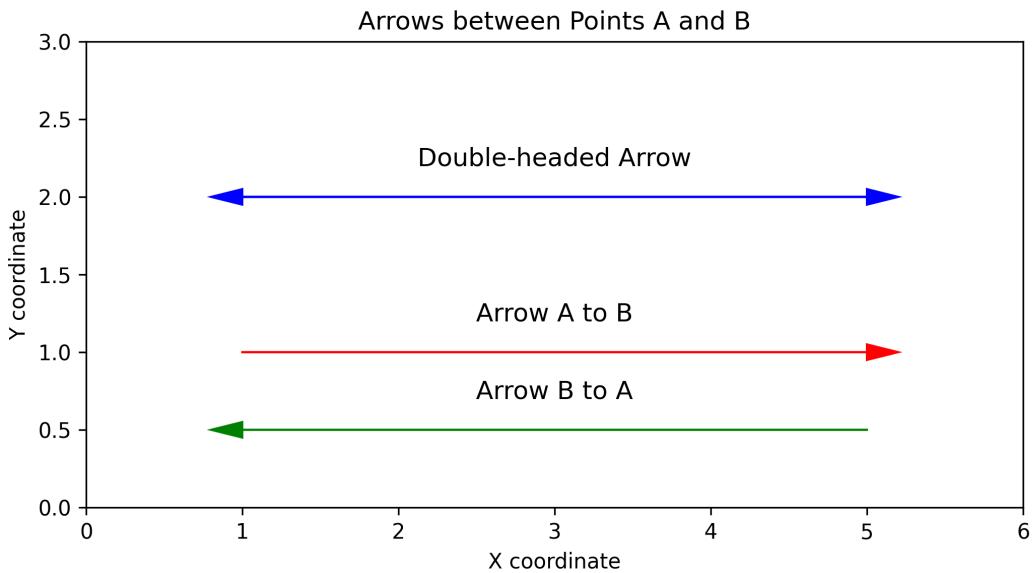


Figura 18: Representación del proceso que se debe realizar en las carreteras de doble sentido.

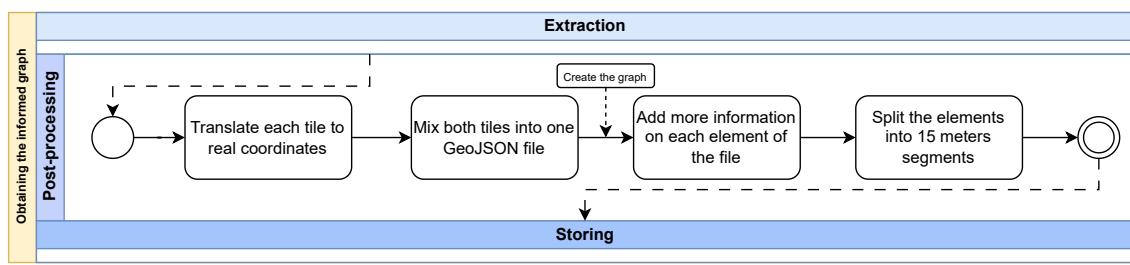


Figura 19: Gráfica con el proceso de traducción y mezcla de archivos.

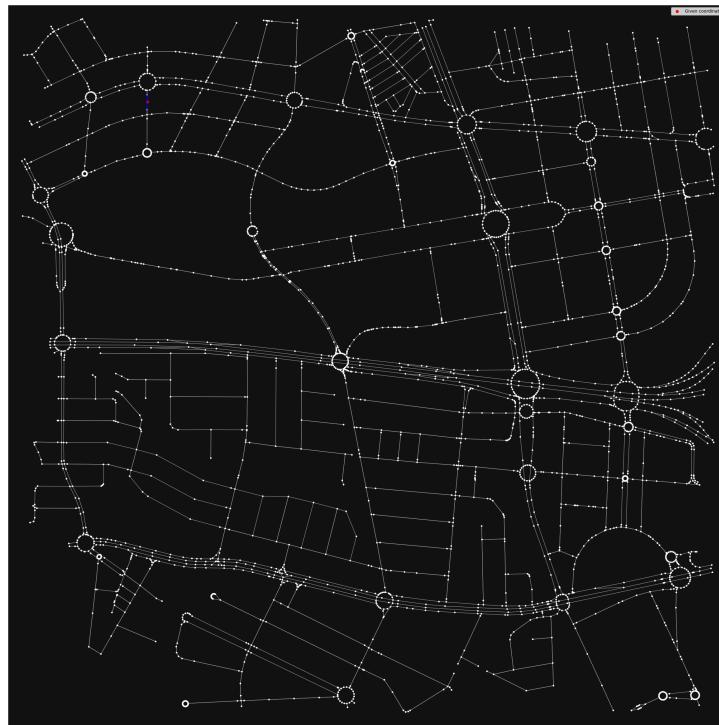


Figura 20: Representación del grafo con la arista más cercana a un elemento del GeoJSON

5.5. Grafo con información

5.5.1. Traspaso de datos a la estructura

Con todos los elementos siendo líneas de dos puntos a una distancia igual o inferior a 15 metros o menos, podemos proceder con el traslado de información a la estructura.

Para ello, como se adelantaba previamente, haremos uso de una función que nos proporciona la librería de *Open Street Map*, que permite localizar la arista más cercana a unas coordenadas dadas que, en este caso, serán las del punto central de la línea, como podemos observar en el ejemplo de la fig. 20. Pero, ahora que tenemos los elementos divididos en secciones de 15 metros, volveremos a calcular cuál es la arista más cercana, para posteriormente traspasar la información a la arista correspondiente del grafo.

Aunque la información de las carreteras de doble sentido recibida de TomTom generalmente coincide con la del grafo (es decir, A→B y B→A), existe un caso aislado donde esto no se cumple. En la **Calle Jiménez Fraud**, las vías en ambos sentidos no aparecen superpuestas en el grafo, aunque sí están superpuestas en la información de TomTom, como se puede observar en la fig. 21. TomTom superpone las vías en la dirección Sur (color verde). Dado que

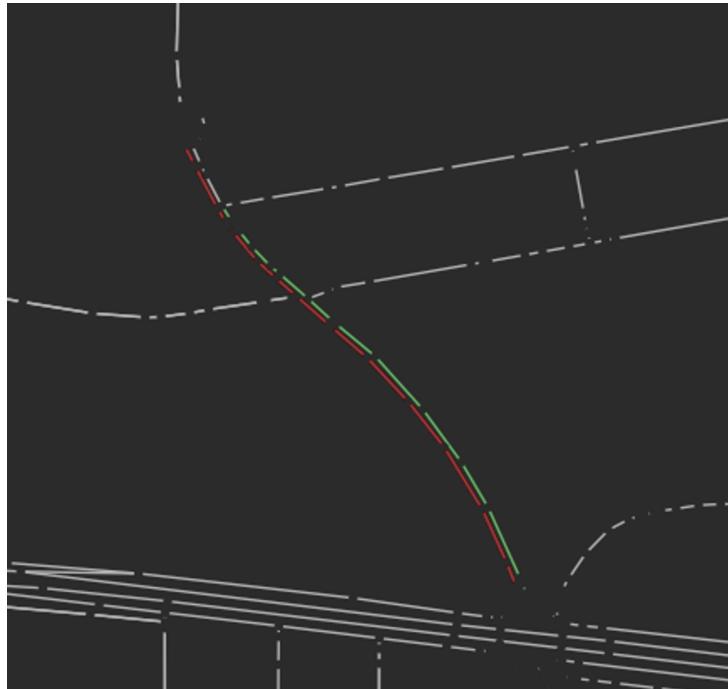


Figura 21: Imagen del grafo ampliado en Calle Jiménez Fraud

estas aristas tienen orígenes y destinos distintos, no podemos asociar el elemento a una arista basándonos en la dirección en la que apuntan. Por lo tanto, para esta calle será necesario implementar una solución específica (*hardcoding*), considerando cada una de las aristas que componen los dos sentidos de la calle.

El procedimiento será el siguiente: durante el procesamiento del archivo GeoJSON con todos los elementos, se verificará si la arista del grafo más cercana al elemento es una de la Calle Jiménez Fraud (con OSMID 199419587) y, si además, la dirección de la arista es opuesta a la de la línea del elemento. En este caso, se realizará una conversión manual a la arista correspondiente, seleccionando la más cercana que apunte en la dirección correcta (Sur).

5.5.2. Interpolación de la información

A pesar de todo el procesamiento, seguiremos teniendo algunas **aristas del grafo sin información**, bien porque a la hora de asociar la información alguna arista ha quedado sin información, que en este caso no suelen ser más de dos aristas seguidas sin información, o bien porque el proveedor no nos da información de esa zona. Esto ocurre mucho en calles residenciales o en partes que salen ligeramente del cuadrante del *tile*, pero la hemos tenido que añadir al grafo para que mantenga la coherencia.

En cualquier caso no supone un problema grave, pero hay que implementar un **algoritmo que complete de información el grafo** con el mayor realismo posible. Se plantearon varias posibles soluciones:

- En un primer momento se barajó la posibilidad de que aquellas aristas que no tuviesen información directamente del proveedor tuvieran el mejor valor de tráfico posible (1.0), pero esto carecía de total sentido. Tal vez podría haberse aplicado solo a las calles residenciales que eran mencionadas anteriormente pero, en muchas ocasiones, las aglomeraciones de tráfico en vías principales suponen alteraciones en el tránsito normal de estas vías. Esta posible solución era la que menor tiempo de ejecución suponía por la ausencia de carga computacional.
- Este algoritmo partía de la premisa de que, si una arista no tenía información, pero al menos dos o más de sus aristas vecinas contaban con niveles de tráfico conocidos, entonces se podía estimar su nivel de tráfico propagando los valores de sus vecinas. El proceso se repetía de manera recursiva, de modo que cada vez que una arista era completada, se utilizaba esa nueva información para ayudar a estimar el tráfico en otras aristas cercanas. Este procedimiento continuaba hasta que el grafo estuviese prácticamente completo. Sin embargo, en aquellas aristas aisladas, sin vecinos con información suficiente, se les asignaba directamente el mejor nivel de tráfico disponible. Esta opción se implementó, pero los resultados eran muy incoherentes, ya que esta forma de propagar el nivel de tráfico daba lugar a situaciones donde dos aristas colindantes tuvieran niveles de tráfico totalmente opuestos.
- Otra alternativa es aplicar un algoritmo que sigue una estrategia iterativa y que tiene en cuenta todas las posibles fuentes de nivel de tráfico para una arista vacía. Para este algoritmo es importante tener un diccionario que contenga los vecinos de cada una de las aristas del grafo. Para ello se ha desarrollado otro algoritmo que lo crea. Durante cada iteración, el algoritmo recorre todas las aristas y, para aquellas que no tienen datos de tráfico, calcula un nuevo nivel de tráfico basado en la media de los niveles de tráfico de sus aristas vecinas que sí tienen datos. Si el nuevo nivel de tráfico calculado difiere del actual (considerando una precisión especificada), se actualiza el nivel de tráfico de la arista y se continúa iterando. Este proceso se repite hasta que no se puedan interpolar

más aristas, asegurando que todas las aristas posibles en el grafo tengan un nivel de tráfico asignado basado en la información disponible de sus vecinas.

Teniendo en cuenta que no existe un algoritmo fiable al 100 % para este problema, los dos primeros fueron descartados para este proyecto por la falta de realismo en los resultados aportados. Se optó por la implementación del último algoritmo de propagación comentado, por la calidad de los resultados aportados en un caso como el nuestro, que no tendremos una gran cantidad de grafos sin información continuada. Una implementación en pseudocódigo está disponible en el algoritmo 1.

El valor de precisión óptimo para este proyecto es de tres decimales, ya que con más cantidad lo único que conseguiremos es que tarde más tiempo del necesario, ya que la información aportada por este cuarto decimal será completamente despreciable a efectos prácticos.

A pesar de los buenos resultados de este algoritmo, es importante tener en cuenta que en términos de complejidad puede no llegar a ser el óptimo (en función del número de aristas que haya que interpolar). El algoritmo presenta una complejidad de $O(U \times E)$, donde E es el número de aristas en el grafo y U es el de aristas que inicialmente no disponen de información de tráfico.

La inicialización del diccionario de vecinos tiene una complejidad de $O(E^2)$, ya que es necesario recorrer todas las aristas por cada una de ellas para encontrar las aristas vecinas. Como es un cálculo que solo será necesario ejecutar una vez por grafo, la función se ha diseñado de forma que podemos pasarlo como argumento y ahorraríamos este cálculo cuando calculemos la interpolación de varias fechas.

Esto implica que, si un gran número de aristas carece de información de tráfico al inicio, el tiempo de ejecución del algoritmo puede incrementarse significativamente. En grafos densos o de tamaño muy grande, el rendimiento del algoritmo podría volverse subóptimo, requiriendo posibles optimizaciones o enfoques alternativos. No obstante, para grafos de tamaño moderado y con un número razonable de aristas sin información, como es nuestro caso, el algoritmo resulta eficaz para la interpolación de niveles de tráfico, proporcionando una solución práctica para completar la falta de información de tráfico en las aristas del grafo.

Algorithm 1 Interpolate Traffic Level

Require: Graph G , Date-info filename f , Neighbours Dictionary d (optional), Precision p

```
1:  $num\_edges\_interpolated \leftarrow 1$ 
2: if  $d$  is None then
3:    $d \leftarrow \{\}$ 
4:   for each edge  $(u, v)$  in  $G$  do
5:      $d[(u, v)] \leftarrow$  Get Neighbours of edge  $(u, v)$  in  $G$ 
6:   end for
7: end if
8: while  $num\_edges\_interpolated > 0$  do
9:    $num\_edges\_interpolated \leftarrow 0$ 
10:  for each edge  $(u, v, data)$  in  $G$  do
11:    if  $data['dates'][f]['api\_data']$  is False then
12:       $neighbours\_edges \leftarrow d[(u, v)]$ 
13:       $neighbour\_traffic\_levels \leftarrow \{$ traffic level of each neighbour edge with traffic
14:        level not None $\}$ 
15:      if size of  $neighbour\_traffic\_levels > 0$  then
16:         $new\_traffic\_level \leftarrow$  average of  $neighbour\_traffic\_levels$ 
17:        if rounded  $new\_traffic\_level$  to  $p$  decimal places  $\neq$  rounded
18:           $data['dates'][f]['traffic\_level']$  to  $p$  decimal places then
19:             $data['dates'][f]['traffic\_level'] \leftarrow new\_traffic\_level$ 
20:           $num\_edges\_interpolated \leftarrow num\_edges\_interpolated + 1$ 
21:        end if
22:      end if
23:    end if
24:  end for
25: end while
```

```
{
  "dates": {
    "YYYY_MM_DD_HH_mm_SS": {
      "traffic_level": "X.Y",
      "api_data": "True/False"
    },
    ...
  },
  ...
}
```

Figura 22: Diccionario que se añadirá a cada una de las aristas del grafo

5.5.3. Resultado

Finalmente, todas las aristas quedarían cubiertas de información de tráfico de las fechas en las que se ha pedido al proveedor. Por lo tanto, tendríamos un grafo, con 4.093 aristas, y cada una de ellas tendrá un diccionario como el que podemos ver en la fig. 22 con las fechas, el nivel de tráfico correspondiente y si esa información ha sido obtenida directamente desde el proveedor o mediante la interpolación de valores.

5.6. Almacenamiento

Almacenar toda la información dentro del grafo resulta muy ineficiente, pero como podría llegar a ser útil para algunos procesos, no se ha descartado por completo esta opción. Pero para los siguientes pasos del proyecto es necesario tener un sistema de almacenamiento fiable, eficiente, rápido y cómodo para asegurar el correcto funcionamiento de las aplicaciones. Por lo tanto se ha optado por utilizar una base de datos no relacional basada en documentos de MongoDB.

Puesto que tenemos que almacenar mucha información y esta deberá ser procesada por MongoDB posteriormente mediante filtros y agregaciones, es importante tener una buena estructura que asegure la eficiencia de las operaciones. Por ello, tras leer documentaciones de MongoDB y hacer diversas pruebas de tiempo, se optado por el siguiente proceso:

```

_id: ObjectId('6659d684a05ac1b28650d988')
  ↳ links : Array (4093)
    filename : "2024_05_08_16_19_06.pbf.json"
    datetime : 2024-05-08T16:19:06.000+00:00
    hour_minute_string : "16:19"
    hour_int : 16
    minute_int : 19
    day_of_week : "Wednesday"

```

Figura 23: Captura de MongoDBCompass con la estructura de cada documento

1. Recorrer todas las aristas, y eliminar toda aquella información que no se altere entre fechas, es decir, no almacenaremos en MongoDB aquella información que podamos obtener de otra forma.
2. Con la librería de *NetworkX* somos capaces de obtener la información de los nodos y las aristas del grafo en un formato adecuado para almacenar la información en MongoDB (serializable en JSON).
3. Descartamos la información del grafo y de los nodos, y nos quedamos únicamente con la de las aristas, de las cuales solo almacenaremos la información necesaria para filtrarla e identificarla, como el nombre, el tipo de vía, el *OSMID* y el origen/destino, junto al nivel de tráfico.
4. Añadimos la información necesaria para el posterior filtrado de los datos, como es la fecha.

Podemos observar en la fig. 23, los resultados de estas operaciones y cómo se almacenará la información en nuestra base de datos de MongoDB.

Habiendo limpiado los datos antes de su almacenamiento, podemos asegurarnos de que las operaciones funcionarán lo más rápido que MongoDB permita, y con la información que hemos incluido, seremos capaces de filtrarla de diferentes maneras que nos ayudarán a entender mejor la información que hemos recogido. En el próximo capítulo nos adentraremos con mayor profundidad en esto.

Todo el proceso para añadir la información al grafo y a su vez exportarlo a la base de datos de MongoDB, ha sido ilustrado en la fig. 24. Y a modo de esquema general de este capítulo, tenemos en la fig. 25 y en la fig. 26 todo el proceso seguido.

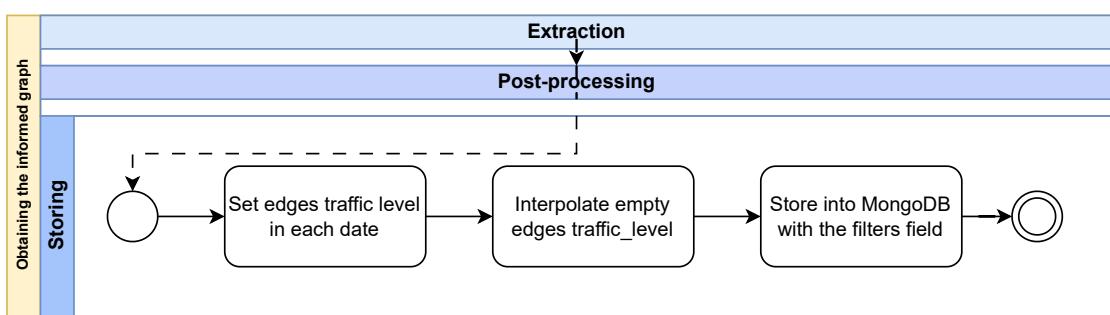


Figura 24: Gráfica con el proceso de exportación de los datos al grafo y MongoDB.

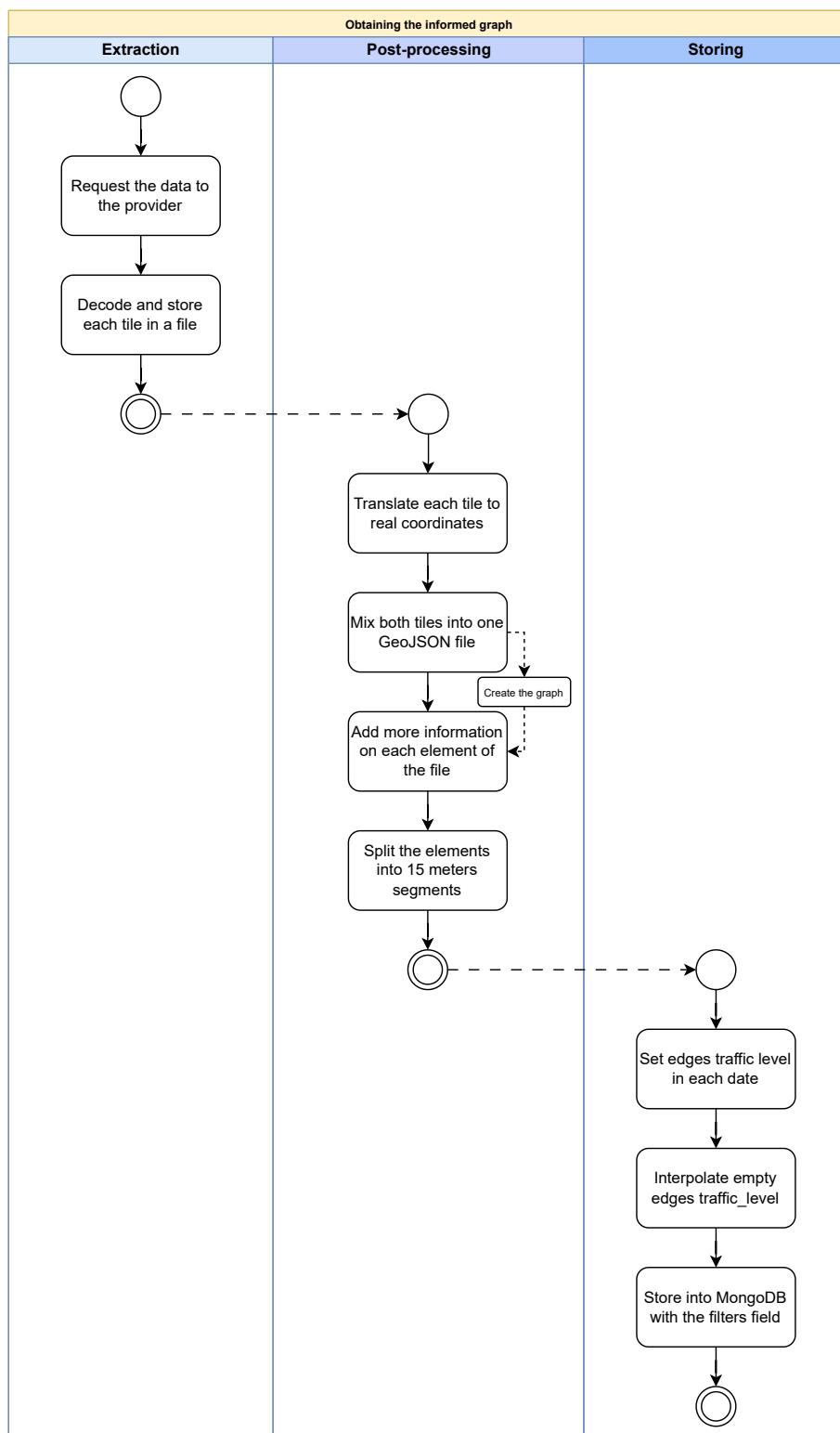


Figura 25: Gráfica con el proceso de extracción de la información.

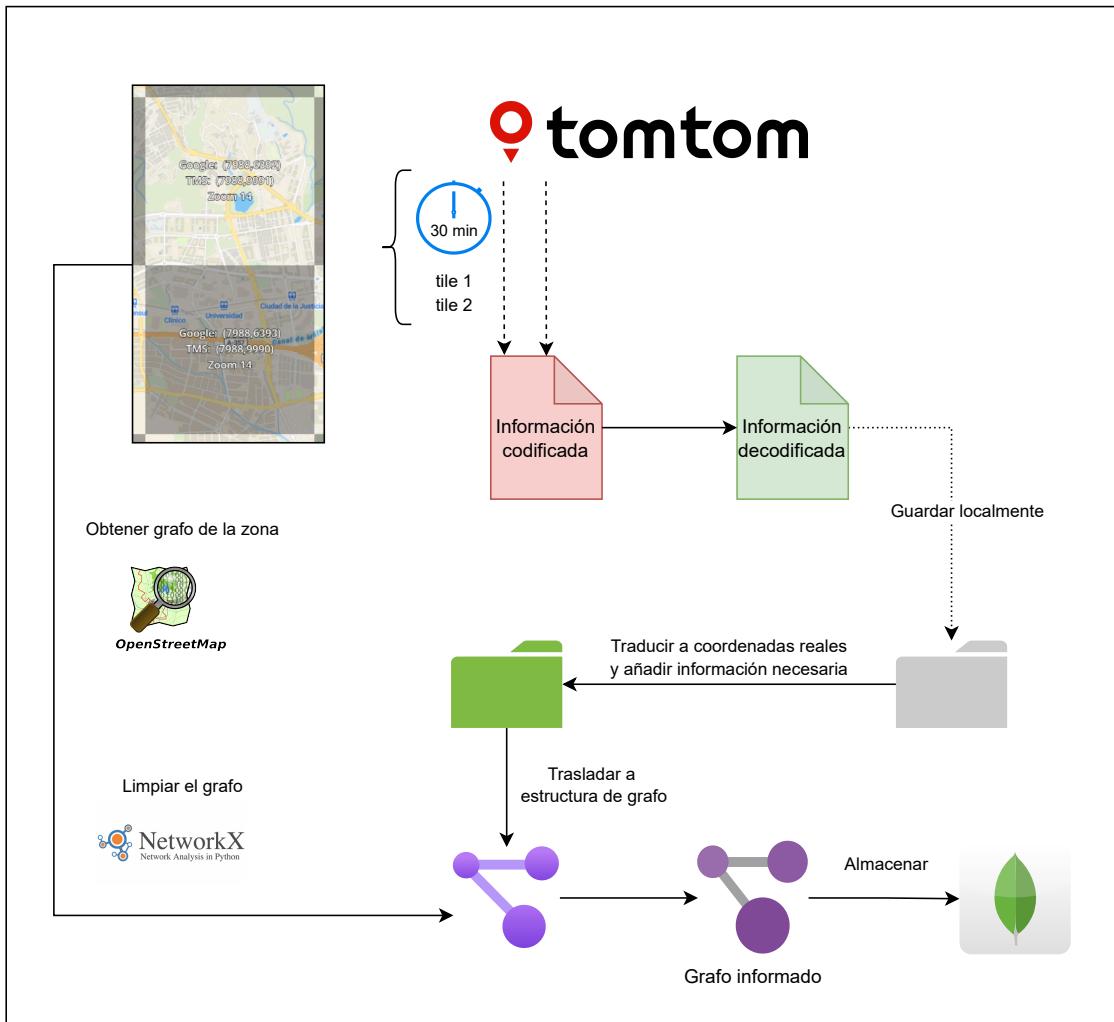


Figura 26: Diagrama general del proceso de extracción de la información.

6

Panel de datos

6.1. Objetivos del tablero

Una vez terminado el proceso de extracción de información y teniéndola almacenada, es necesario desarrollar una plataforma en la que se pueda **representar de forma eficiente y cómoda** los niveles de tráfico, permitiendo alternar entre fechas. Hasta el momento, habíamos usado una funcionalidad incluida dentro de la librería de *Open Street Map* para generar imágenes de nuestro grafo (como la de la fig. 27). Esto está lejos de ser un método eficiente para manejar la información debido al alto coste temporal de generación, la poca manejabilidad de estas imágenes y la falta de diversas funcionalidades especialmente útiles para este proyecto, como hacer zoom, obtener información de zonas determinadas, entre otras.

Por lo tanto, queda descartada la generación de imágenes para la representación de la información. La mejor opción para representar la información en la plataforma sería incluir un **mapa interactivo** donde se pueda visualizar la información disponible de las aristas y los nodos. Concretamente, un mapa que permita cargar un grafo dentro de este, aprovechando al



Figura 27: Imagen del grafo generada con la librería OSMNx.

máximo la eficiencia de esta estructura también en las representaciones. Este mapa debe ser capaz de mostrar al usuario la situación del tráfico con un simple vistazo y, además, permitir la interacción para obtener información más detallada.

Para enriquecer la plataforma, se debería incluir información adicional a la presentada en el mapa. Sería de gran valor aportar al usuario gráficas estadísticas de diversos tipos con información del tráfico. De esta forma, el usuario podría incluir análisis detallados y obtener una comprensión más profunda de los patrones de tráfico, facilitando la toma de decisiones basadas en datos históricos y en tiempo real.

En resumen, la plataforma debe ser una herramienta integral que no solo muestre el estado del tráfico de manera eficiente y dinámica, si no que también proporcione funcionalidades interactivas y datos adicionales que permitan un análisis exhaustivo y práctico de la información de tráfico, facilitando así una mejor gestión y toma de decisiones informadas.

6.2. Información disponible

Es importante rescatar del capítulo anterior la información de la que disponemos y de la que podemos hacer uso. Por cada instante en el que hemos obtenido información del tráfico en los dos cuadrantes de la zona de Teatinos, tenemos un documento con la fecha exacta en la que se obtuvo, incluido el día de la semana que es, una lista con todas las aristas del grafo, cada una de ellas con el nivel de tráfico que tenía, y algunos campos extra que nos ayudarán a la hora de filtrar, como son el nombre de la calle, el tipo de vía y si ha sido un dato interpolado o no.

Adicionalmente a la información que tenemos almacenada en la base de datos, si recordamos, tenemos una estructura que denominamos grafo base, que nos iba a servir de plantilla para el resto de pasos. Este grafo contiene información de cada uno de los nodos: concretamente, podemos saber si en el nodo hay un paso de cebra, un semáforo (esta información se ha añadido manualmente) y el número de calles que conecta. También de cada una de las aristas, como la velocidad máxima a la que se puede circular, el número de carriles que contiene o hay vía de doble sentido (fig. 28).

```
{
  "osmid":359280372,
  "lanes":"3",
  "name":"Plaza Pintor Sandro Botticelli",
  "highway":"secondary",
  "junction":"roundabout",
  "oneway":true,
  "reversed":false,
  "length":13.66,
  "maxspeed":"50",
  "speed_kph":50.0
}
```

Figura 28: Información disponible de una arista en el grafo base.

6.3. Información procesable

Aunque la diversidad de información de la que disponemos es limitada, gracias a que tenemos una gran cantidad, podemos realizar ciertos procesamientos para rescatar información valiosa. Esto nos permitirá generar gráficas adicionales al mapa que ofrecerán al usuario una visión más completa y detallada del tráfico.

Para generar las gráficas, será un requisito indispensable que el **procesamiento de los datos lo haga MongoDB**, ya que proporciona una gran capacidad para manejar y procesar grandes volúmenes de datos en tiempo real, lo que es crucial para las necesidades de la plataforma. Su estructura de documentos flexible permite consultar y procesar datos de tráfico de manera eficiente, facilitando la generación de gráficas a partir de los resultados aportados.

Esto lo conseguiremos gracias a las capacidades de **agregación** de MongoDB, que permiten realizar cálculos complejos y obtener estadísticas de manera rápida y precisa, optimizando el rendimiento de la plataforma. Esto no solo mejora la experiencia del usuario, si no que también asegura que el sistema pueda escalar y manejar grandes cantidades de datos sin comprometer la velocidad ni la precisión de las gráficas presentadas.

Para esta sección será muy importante tener en cuenta dos valores similares pero distintos. Hemos tratado mucho el nivel de tráfico, pero a la hora de visualizar los datos puede ser más interesante ver la velocidad de la vía. Por lo tanto, tendremos que trabajar ambos valores en las gráficas que mostremos al usuario. Teniendo en cuenta que podrán intercambiarse entre los valores mínimos, máximos, medios o medianas de ambas opciones, las gráficas y análisis

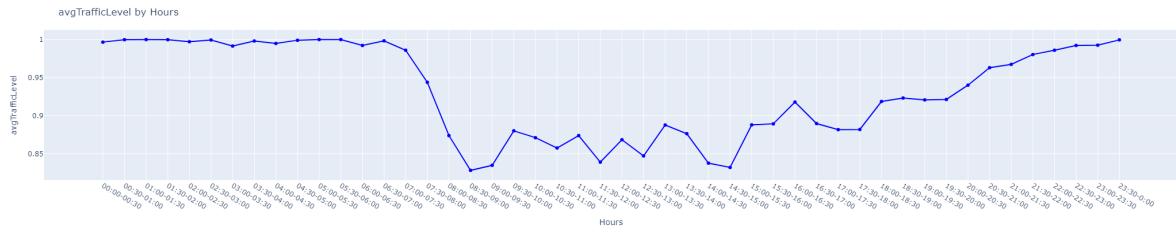


Figura 29: Gráfica con el nivel de tráfico medio en el rango de 24 horas del día.

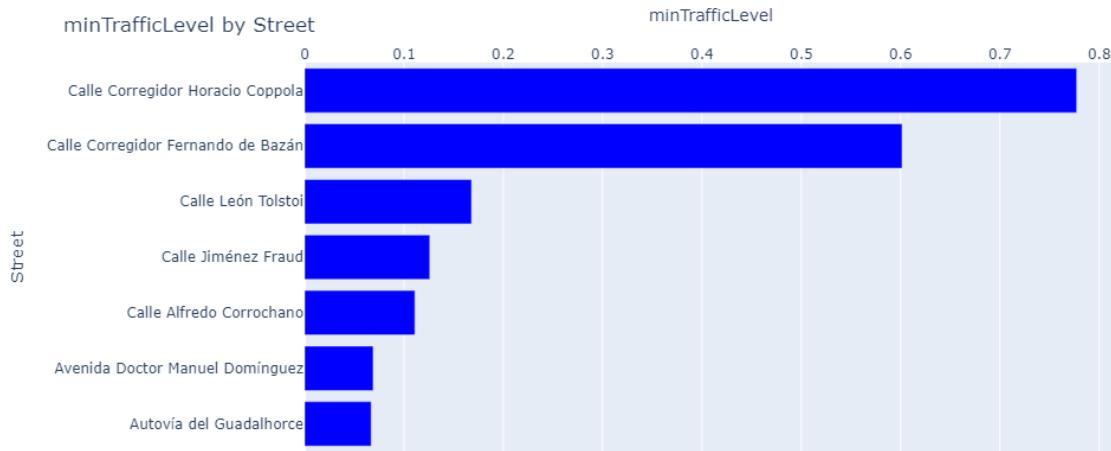


Figura 30: Gráfica con los niveles mínimos de tráfico en las calles.

que se incluirán en la plataforma son:

- **Gráfica por franjas horarias:** Permitirá al usuario visualizar cómo varía el tráfico a lo largo del día. Ayuda a observar patrones específicos de tráfico, lo que es especialmente útil para identificar horas pico y momentos de menor congestión (fig. 29).
- **Gráfica con el nombre de las calles:** Con esta gráfica, el usuario puede comparar claramente los diferentes valores en las calles, siendo capaz de hacer una síntesis general de conjuntos de aristas rápidamente (fig. 30).
- **Gráfica por día de la semana:** Cuando el usuario seleccione franjas de tiempo mayores a un día será muy interesante ver cómo varían las condiciones del tráfico en función del día de la semana y la influencia que tiene esto en el tráfico real (fig. 31).

Para que MongoDB nos proporcione los datos requeridos para la generación de las gráficas, son necesarios tres procesamientos individuales en la base de datos, resultando en cada una



Figura 31: Gráfica con la velocidad media en los días de la semana.

de las gráficas. Conseguiremos delegar el procesamiento de los datos a MongoDB gracias a las **agregaciones**, concretamente usaremos **tuberías de agregaciones**, que nos permite dividir el procesamiento en varias fases, donde la salida de cada fase será la entrada de la siguiente hasta completar la agregación.

Ya que cada documento de la base de datos está compuesto por una lista de aristas, será necesario aplicar una operación para desempaquetarlas (`$unwind`). Esta operación, además de ser muy costosa, aumenta considerablemente la memoria ocupado por los documentos durante el procesamiento. Para reducir al máximo los inconvenientes de esta operación se procederá a hacer un filtrado previo de los documentos, eliminando del procesado aquellos que estén fuera del rango de fecha y hora seleccionados por el usuario. Posteriormente se desagruparán las aristas y se filtrarán, ahora por el nombre y tipo de las vías.

Una vez que en el conjunto de datos solo tenemos aquellos que realmente debemos procesar, podemos continuar con la agregación: concretamente seguiremos con la agrupación (`$group`). Esta fase de la agregación es la que difiere entre las tres del análisis para generar las gráficas, ya que, teniendo como entrada las vías filtradas, es capaz de agruparlas en función de los campos que se indiquen. Será en este campo donde agruparemos las aristas por franja de hora, día de la semana o nombre de la vía. Además dentro de esta misma fase, aprovecharemos el procesado de las aristas para añadir los campos que representaremos posteriormente en las gráficas, como son los mínimos, máximos, valor medio y mediana del nivel de tráfico y velocidad de la vía.

```

{
  "_id": {
    "$concat": [
      { // Si la hora es menor que 10, añade un "0" al principio
        "$cond": [{ "$lt": ["$_id.hour", 10] }, "0", ""]
      },
      { // Convierte la hora a cadena de texto
        "$toString": "$_id.hour"
      },
      ":" ,
      "$_id.halfHour", // Añade el valor de halfHour
      "-",
      { // Si la hora incrementada es menor que 10, añade un "0" al principio
        "$cond": [
          { "$lt": [{"$add": ["$_id.hour", { "$cond": [{ "$eq": ["$_id.halfHour", "30"] }, 1, 0] } ]}, 10] }, "0", ""
        ]
      },
      { // Convierte la hora incrementada (con módulo 24) a cadena de texto
        "$toString": {
          "$mod": [{"$add": ["$_id.hour", { "$cond": [{ "$eq": ["$_id.halfHour", "30"] }, 1, 0] } ]}, 24]
        }
      },
      ":" ,
      { // Si halfHour es "00", devuelve "30", si no, devuelve "00"
        "$cond": [
          { "$eq": ["$_id.halfHour", "00"] }, "30", "00"
        ]
      }
    ]
  }
}

```

Figura 32: Sección de código donde se genera el identificador por la franja de tiempo.

Adicionalmente, para el caso del procesado de los datos por horas, será necesario un paso extra que nos facilitará su representación (fig. 32), ya que con la operación de proyección (`$project`) podemos modificar el identificador de los documentos, pudiendo identificar los documentos directamente por la franja horaria a la que pertenezca.

Todo el proceso puede visualizarse en la fig. 33 a modo de esquema, para entender mejor el filtrado y las transformaciones que sufren los datos.

Finalmente, ya tendríamos todos los datos procesados y podríamos recoger los resultados en la plataforma y generar las gráficas.

En resumen, la inclusión de estas gráficas y análisis adicionales proporcionará a los usuarios una herramienta poderosa para comprender mejor el tráfico urbano. No solo podrán visualizar el estado actual del tráfico, si no también analizar tendencias, comparar datos históricos y anticipar problemas futuros. Esto mejorará significativamente la capacidad de los usuarios para tomar decisiones informadas y optimizar diversos estudios y procedimientos.

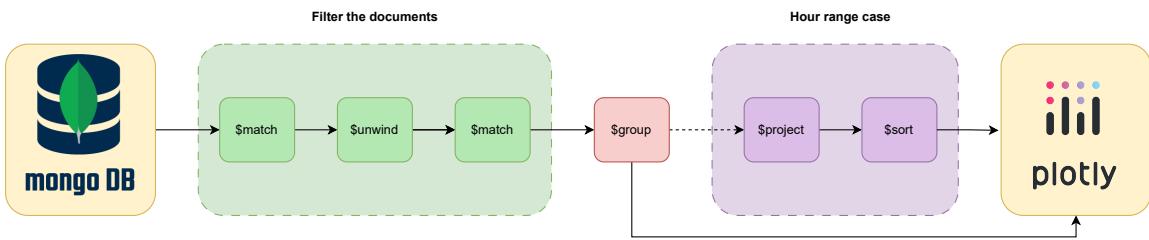


Figura 33: Diagrama con las etapas de las agregaciones en MongoDB.

6.4. Interfaz de usuario

6.4.1. Plataforma del tablero

Una vez que sabemos qué es lo que queremos mostrar, es necesario elegir la plataforma en la que se va a desarrollar. En un primer momento se barajó la posibilidad de utilizar *D3.js* para realizar el tablero, que es una librería de *JavaScript* enfocada en el desarrollo de aplicaciones web en las que se muestren representaciones dinámicas de datos o gráficas. Pero una característica deseable sería poder realizarlo con el lenguaje *Python*, ya que podemos reutilizar código del capítulo anterior, y la librería que tiene *Python* para trabajar con *MongoDB* es de una calidad excelente.

La librería más comúnmente utilizada para este tipo de proyectos en el lenguaje *Python* es ***Plotly Dash***, que también es de código abierto. Permite la creación de una página web dinámica e interactiva, donde cada acción que haga el usuario podrá ser programada y ejecutar cualquier código escrito en *Python*. Otra ventaja importante que tiene esta librería frente a otras es que permite desarrollar la web sin la necesidad de tener ficheros *HTML* o *CSS*. Tenemos la posibilidad de aplicar directamente a la web los estilos predefinidos de ***Bootstrap***, que dotarán a la web de un estilo serio y conocido para los usuarios. Por lo tanto, será la librería que utilicemos para la creación del tablero.

Principalmente se busca que la interfaz sea intuitiva y cómoda de utilizar para cualquier usuario. Es por ello que se seguirá un diseño basado en columnas y tarjetas de *Bootstrap*, con el objetivo de mantener una estructura clara y ordenada. A continuación, se detallan las decisiones tomadas para cada sección y el motivo detrás de ellas.



Figura 34: Gradiente de color para las aristas del mapa.

6.4.2. Mapa

Uno de los componentes más importantes para la interfaz será el mapa. Este se encontraría ubicado de manera central en la interfaz, y que como se comentaba previamente, era importante disponer de uno que permitiera cargar grafos. Esta característica no es nativa en ninguno de los componentes tipo mapa comúnmente usados en los proyectos en *Dash*, por lo que se suele optar por traducir el grafo a un formato *GeoJSON* que sí es un formato admitido por los mapas, pero esto no es lo deseado para nuestro proyecto, ya que queremos la máxima eficiencia y velocidad posible.

Por lo tanto, se ha optado por usar ***Dash Sylvereye*** que es una librería de código abierto que proporciona al desarrollador un componente en *Dash* con un mapa que permite cargar un grafo en tiempo real, con la capacidad de generar eventos cuando se pulsa sobre algún elemento y con diversas opciones de personalización. [4]

Esta librería permite añadir un proveedor de servicio de *tiling* personalizado (permite visualizar el mapa). Como el que venía por defecto daba problemas de rendimiento y el mapa aportado era de muy baja calidad, se barajaron algunos proveedores como *Google*, *HERE*, *OpenStreetMap*, *MapTiler*, pero por la calidad de las imágenes del mapa, posibilidades de personalización y precios, el proveedor seleccionado ha sido ***Mapbox***.

También permite dar color a las aristas en función de un valor dado. En nuestro caso, queremos colorearlo en función de la velocidad, del nivel de tráfico o de la velocidad máxima. Tras hacer diversas pruebas con esta función de la librería, esta no aplica el color de la manera correcta, ya que tiene algunos problemas a la hora de encontrar el valor mínimo. Por lo tanto, es necesario crear una función propia para darle el color a la arista, que tendrá en cuenta el atributo a colorear y sus valores mínimos y máximos en la instancia seleccionada. Esta función generará el gradiente de color de la fig. 34, que coloreará las aristas del grafo en el mapa como en la fig. 35.

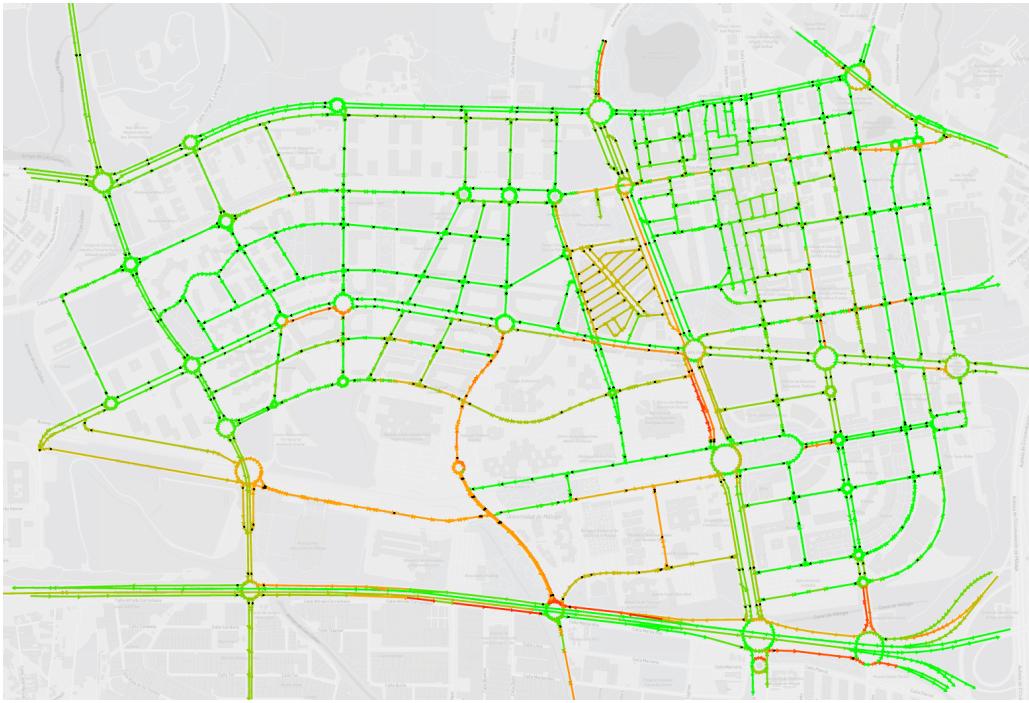


Figura 35: Gradiante de color para las aristas del mapa.

6.4.3. Extracción de Datos

En la parte superior de la interfaz, debajo del título, se encuentra un panel dedicado al **procesamiento de los datos** para poder generar las gráficas comentadas previamente, de acuerdo a una serie de filtros. Este panel permite al usuario seleccionar un rango de fechas, horas, tipos de calles y nombres para filtrar la información que se procesará. Y para que el usuario sea consciente de que filtros esta estableciendo, el mapa se actualizará automáticamente conforme los filtros se vayan seleccionando. La elección de este diseño se debe a la necesidad de proporcionar una forma clara y eficiente de refinar los datos sin sobrecargar la interfaz.

El funcionamiento será el siguiente: cuando el usuario quiera procesar los datos, pulsará el botón central, que tiene un color llamativo para captar la atención del usuario. Una vez que lo pulse, verá un círculo giratorio para que entienda que se está llevando a cabo lo que se ha pedido. Una vez que dicho procesamiento ha terminado, se mostrará una ventana flotante de aviso, y el usuario volverá a tener disponible el botón central, y también tendrá habilitados los botones de descarga, que ofrecerán al usuario los datos recogidos por *MongoDB* en formato CSV. Podemos ver este proceso en la fig. 36.

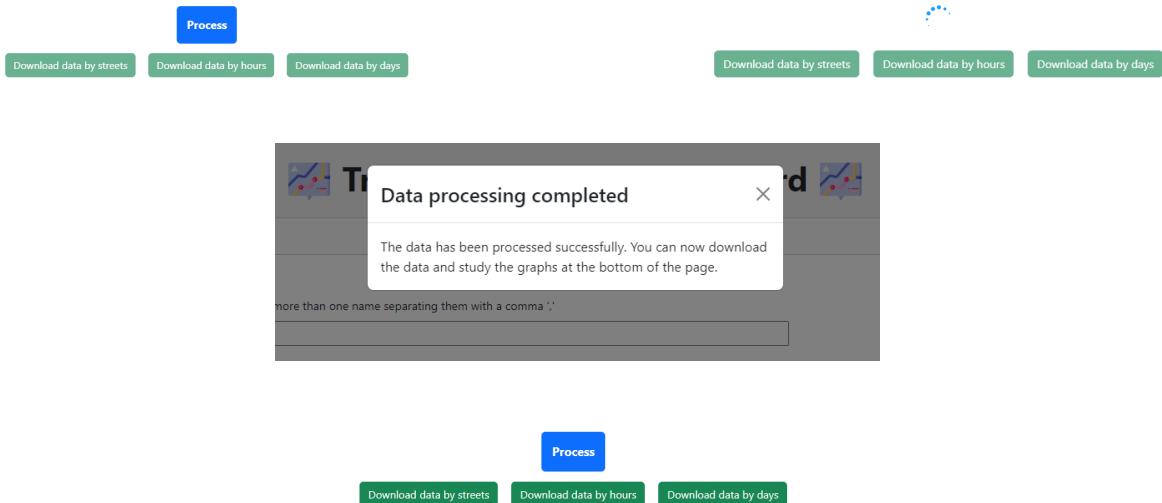


Figura 36: Proceso que sufre la interfaz mientras se procesan los datos

La información estará disponible para **descargar** gracias a que mientras se están creando las gráficas, se está guardando la información localmente en la plataforma, de forma que solamente tendríamos que hacer una traducción al formato *CSV* en el momento de la descarga, gracias a la librería *Pandas*.

6.4.4. Información del Nodo y de la Arista

A la izquierda de la pantalla, tendremos una columna en la que se encuentran dos tarjetas dedicadas a mostrar información detallada sobre los nodos y las aristas sobre las que el usuario pulse. Esta decisión se tomó para no sobrecargar el mapa y que el usuario pueda obtener rápidamente información relevante sin tener que abandonar la vista principal del mapa.

Además de la información, tendremos una o dos flechas que representan los sentidos de la vía de la arista pulsada. Como podemos observar en la fig. 37. Esto, a priori podría, parecer algo de poca utilidad, pero es realmente para visualizar la información del tráfico de aquellas **vías de doble sentido**, ya que como se ha comentado anteriormente, el grafo superponía las aristas de sentidos inversos. Por lo tanto, es la mejor forma de no perder información por las características del grafo. Para generar las flechas, es necesario recoger el sentido al que apunta la arista pulsada y su valor de tráfico, y en caso de ser de doble sentido, recoger también su valor, y generar las dos flechas con los valores correspondientes y apuntando en la misma

Node Information	
Property	Value
Latitude	N/A
Longitude	N/A
Street Count	N/A
Crossing	N/A
Traffic Light	N/A
OSM ID	N/A

Edge Information	
Property	Value
Name	N/A
Maxspeed	N/A
Highway	N/A
Traffic level	N/A
Current Speed	N/A
Lanes	N/A
OSM ID	N/A

Node Information	
Property	Value
Latitude	-4.4762669
Longitude	36.7174842
Street Count	2
Crossing	Yes
Traffic Light	No
OSM ID	5354846229

Edge Information	
Property	Value
Name	Calle Jiménez Fraud
Maxspeed	30
Highway	secondary
Traffic level	0.482
Current Speed	14.46
Lanes	2
OSM ID	284697342

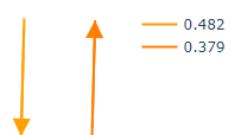


Figura 37: Columna antes y después de pulsar sobre un nodo y una arista

dirección que lo hacen en el grafo.

Adicionalmente a la visualización de las vías de doble sentido, tener toda esta información disponible a una sola acción del usuario es de gran utilidad para estudiar cualquier grafo, ya que no existe una plataforma en línea en la que visualizar grafos de una forma tan detallada y precisa.

6.4.5. Opciones de Visualización de Nodos y Aristas

A la derecha del mapa tendremos otra columna, donde se encuentran los controles para ajustar la visualización de los nodos y aristas. Estos controles permiten al usuario personalizar la apariencia del mapa, ajustando el tamaño, la transparencia y la información que se muestra, permitiendo ocultar aquella información que no le sea necesaria y cambiar el estilo de coloreado de las aristas (en función del nivel de tráfico, velocidad en ese momento o velocidad máxima). Los filtros que se apliquen en esta sección no se tendrán en cuenta a la hora de procesar los datos, por lo que están en otra tarjeta (fig. 38).

La primera tarjeta de la columna servirá para cargar los valores de tráfico de la fecha seleccionada en el mapa. Hasta que no se seleccione una fecha, el grafo no tendrá valores de tráfico y todas las aristas serán de color negro. Este componente desplegable está sincronizado con el calendario de la primera tarjeta. Esto se ha diseñado así para aumentar la eficiencia, porque de lo contrario, tendríamos que tener un desplegable con todos las fechas en las que se han recogido datos. De esta forma también aseguramos consistencia con la información que permitimos seleccionar al usuario.

Para no sobrecargar con información que no vamos a usar a *MongoDB* con las actualizaciones de este componente, se ha creado una nueva colección que solamente contendrá las fechas en las que tenemos información del tráfico. Esto añadirá un paso adicional, pero ligero, al proceso de traducción de la información explicada en el capítulo anterior.

6.4.6. Gráficas

En la parte inferior, se incluyen las gráficas adicionales explicadas. La disposición de las gráficas está pensada para ofrecer al usuario la mejor visualización de datos posible, teniendo una gráfica con las horas ocupando todo el ancho de la pantalla y las dos restantes comparando el ancho debajo de esta (fig. 39 y fig. 40).

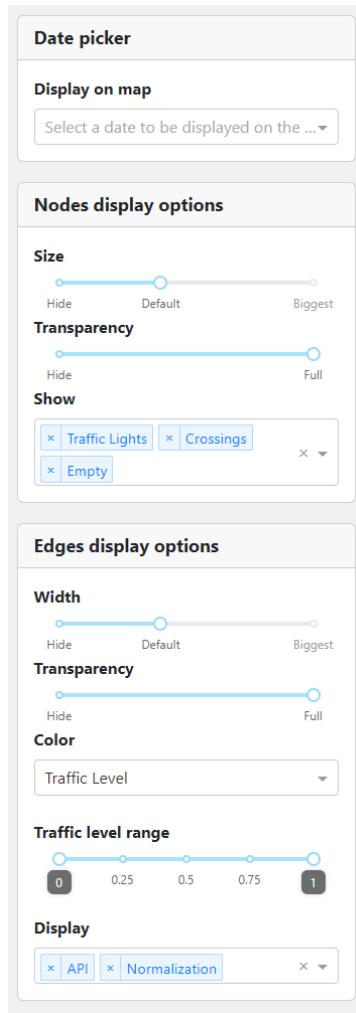


Figura 38: Columna derecha de la interfaz.



Figura 39: Captura de pantalla de la tarjeta con los grafos (vacía)



Figura 40: Captura de pantalla de la tarjeta con los grafos

Estas gráficas creadas mediante *Dash Plotly*, además de ser dinámicas permitiendo hacer zoom y desplazarlo, permiten descargar cada gráfica como una imagen, para poder usarlas en otros estudios o documentos.

Una decisión de diseño importante para esta tarjeta fue la de tener un solo desplegable para todas las gráficas. En vez de tener un desplegable por cada gráfica, que podría llevar a confusiones al usuario, se tiene un único desplegable, y así evitar posibles errores a la hora de estudiar o exportar las gráficas. Para que esto no sea un problema, es importante destacar que el cambio de variable se refleja en las gráficas instantáneamente, gracias a haber guardado la información localmente una vez procesada la información (fig. 41).

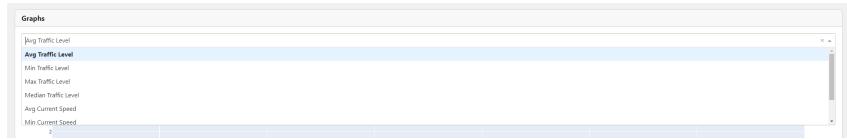


Figura 41: Captura de pantalla del desplegable de la tarjeta con los grafos

6.5. Síntesis del Diseño

Con el objetivo de maximizar la eficiencia para los usuarios que utilicen la aplicación, se ha optado por **compactar la información** y organizar la interfaz en **tres secciones** claramente identificables.

La primera sección se dedica a la configuración y procesamiento de datos; la segunda es un mapa interactivo que ocupa el centro de la interfaz; y la tercera está dedicada a la visualización de resultados.

Además de la funcionalidad, se ha prestado especial atención a la estética de la interfaz para hacerla atractiva y agradable de usar. Un diseño visualmente atractivo no solo mejora la experiencia del usuario, si no que también fomenta un uso más frecuente de la plataforma.

En conjunto, estas tres secciones aseguran una **experiencia de usuario intuitiva y eficiente**, permitiendo una navegación fluida y un acceso rápido a la información necesaria para tomar decisiones informadas. La organización clara, la interacción intuitiva con el mapa y las gráficas, y una interfaz atractiva hacen que la plataforma sea una **herramienta poderosa y agradable** para la gestión y el análisis de datos de tráfico.

Podemos resumir esta sección del proyecto, centrada en la creación del *dashboard* con el diagrama de la fig. 42.

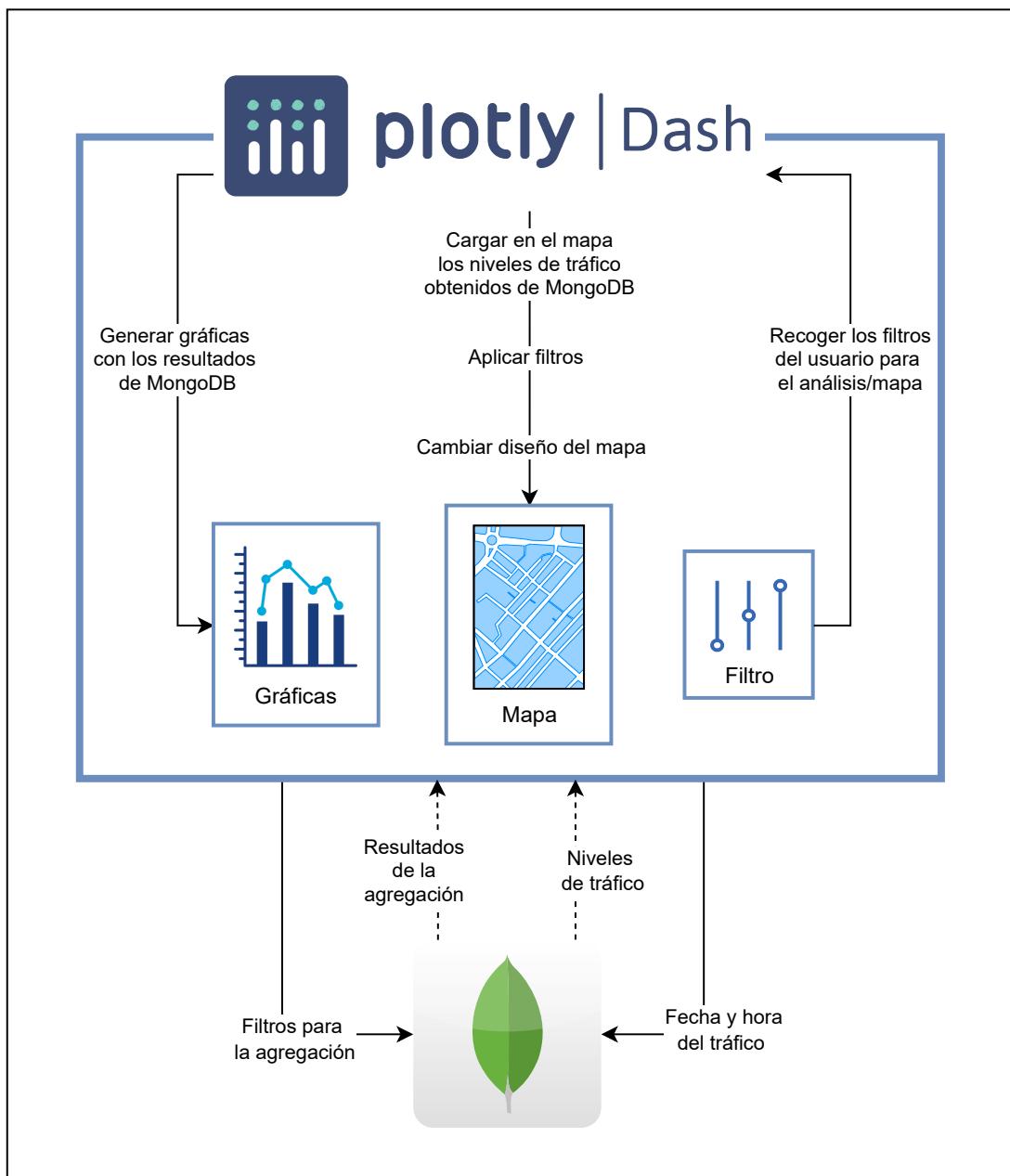


Figura 42: Diagrama general del *dashboard*.

7

Modelo basado en agentes

7.1. Introducción

En la última fase de este proyecto, se ha desarrollado un simulador de flujo de tráfico basado en agentes que permite modelar y analizar el comportamiento del tráfico en una red vial. Este simulador es una herramienta esencial para obtener datos precisos en base a diferentes variables, como la densidad de vehículos y las rutas elegidas por estos, afectan la fluidez del tránsito y contribuyen a la aparición de congestiones.

El objetivo principal de este simulador es proporcionar una plataforma robusta para la realización de experimentos virtuales con diversas opciones de configuración disponibles, que permitan recopilar datos detallados sobre las condiciones de tráfico. Estos datos incluyen tiempos adicionales por congestión, tiempos medios de ruta, y otras métricas clave relacionadas con el flujo vehicular. A partir de estos análisis, se busca prever escenarios de tráfico y apoyar la toma de decisiones informadas para mejorar la gestión del tránsito en tiempo real.

A lo largo de esta sección, se describirán las técnicas empleadas para la construcción del simulador, las características principales del modelo utilizado, la interfaz de usuario para realizar las simulaciones y algunos resultados obtenidos de diversas simulaciones. Estos resultados ofrecen una visión profunda de cómo las condiciones del tráfico influyen en la eficiencia de la red vial y cómo pueden optimizarse las rutas y la gestión del tráfico para minimizar los tiempos de viaje y evitar la congestión.

7.2. Framework del simulador

Actualmente son muchos los *frameworks* existentes para desarrollar modelos de simulaciones basados en agentes. Por lo tanto, es necesario un estudio previo para analizar las diferentes alternativas y tomar una decisión.

Después de una evaluación exhaustiva, se ha decidido utilizar **MESA** para el desarrollo de las simulaciones de flujo de tráfico. A la hora de tomar la decisión se ha comparado exhaustivamente en algunos aspectos con otros frameworks como *Stable Baselines3*, *OpenAI Gym* o *SPADE*:

- **Enfoque en la Modelización Basada en Agentes:** *MESA* está específicamente diseñado para la modelización basada en agentes (*ABM*, por sus siglas en inglés), lo que lo convierte en una herramienta ideal para simulaciones de flujo de tráfico. En este tipo de simulaciones, es crucial representar a cada vehículo o agrupación de vehículos como un agente con comportamientos únicos. *MESA* permite un control detallado sobre las interacciones entre estos agentes, lo cual es esencial para modelar dinámicas de tráfico complejas con precisión.
- **Facilidad de Uso y Flexibilidad:** Una de las principales ventajas de *MESA* es la sencillez de sus funciones, que facilita la definición de agentes, entornos y sus interacciones. Esta flexibilidad permite un desarrollo rápido de prototipos y facilita la integración de diferentes escenarios de tráfico o comportamientos de agentes. Este enfoque ágil es crucial en proyectos donde es necesario experimentar con diversas configuraciones y parámetros, como es el nuestro.
- **Recolección de Datos:** *MESA* ofrece soporte para la recolección de datos de manera integrada. Esto es muy importante a la hora de realizar experimentos, ejecutar múltiples simulaciones con parámetros variables y analizar patrones de tráfico de forma estadística. La facilidad con la que se pueden recolectar y analizar datos de las simulaciones hace que *MESA* sea especialmente adecuado para estudios detallados de tráfico.
- **Comunidad Activa y Extensibilidad:** *MESA* cuenta con una comunidad activa y es altamente extensible, lo que permite personalizar y ampliar el *framework* según las necesidades específicas del proyecto. La comunidad proporciona soporte, ejemplos y bibliotecas.

cas adicionales, que pueden ser de gran ayuda al implementar características avanzadas en las simulaciones de tráfico o al resolver problemas técnicos.

- **Ligereza y Dependencias Mínimas:** *MESA* es un *framework* ligero con dependencias mínimas, lo que facilita su configuración y despliegue, especialmente en entornos donde los recursos son limitados o donde se quiere evitar la complejidad de *frameworks* más grandes. Esta característica es clave dado que se busca desarrollar una aplicación de navegador que no dependa de infraestructuras complejas y pueda otorgar resultados en tiempos aceptables.
- **Adecuado para Sistemas Complejos:** Finalmente, *MESA* es más que adecuado para modelar y simular sistemas complejos y adaptativos, como las redes de tráfico, donde el comportamiento emergente de los agentes interactuando es clave para comprender el sistema en su totalidad. Esto hace que *MESA* sea particularmente efectivo para estudiar fenómenos como la congestión del tráfico, cuellos de botella y el impacto de diferentes estrategias de gestión de tráfico.

En resumen, las características de *MESA*, como su enfoque en la modelización basada en agentes, su facilidad de uso, el soporte para recolección de datos, así como su ligereza y comunidad activa, lo convierten en la opción más adecuada para el desarrollo de este proyecto de simulación de tráfico.

7.3. Desarrollo del modelo

7.3.1. Entorno

Primero definiremos el entorno en el que funcionarán los agentes. Este debe ser lo suficientemente complejo como para permitir que los agentes obtengan información acerca del entorno en cualquier momento, permitiendo el cálculo de rutas en cada momento, y la actualización de los niveles de tráfico conforme el agente se desplaza.

El entorno funcionará con el **grafo base** obtenido previamente, al cual se le añadirá la información de tráfico deseada automáticamente a la hora de iniciar la simulación (será seleccionado desde la interfaz de usuario). Como los niveles de tráfico del grafo se actualizarán

de forma dinámica con el paso de los agentes por las aristas, dentro del entorno tendremos un diccionario que contendrá la información de las aristas actualizadas.

Es muy importante definir claramente cómo pasará el tiempo en la simulación, para poder definir con claridad las funciones de pasos de los agentes. En este simulador de tráfico, se optará por un **modelo de tiempo discreto** debido a la naturaleza del sistema que se está modelando, en el que cada paso de tiempo provocará el movimiento de los agentes de una arista a otra si el tráfico lo permite [1].

Trabajar con un modelo de tiempo discreto es ideal para simulaciones de tráfico urbano, donde los cambios ocurren en **intervalos regulares**. Este enfoque permite actualizar las posiciones, rutas y decisiones de los agentes de manera eficiente en cada paso de tiempo. Además, facilita la implementación de algoritmos que controlan el comportamiento de los agentes, aplicándolos de manera consistente en cada intervalo. Este modelo ofrece claridad y control sobre los eventos, asegurando que todos los aspectos relevantes del tráfico se consideren y actualicen de manera ordenada [1].

También será necesario definir los **recolectores de datos** presentes en las simulaciones para poder extraer información útil de las simulaciones realizadas. Tendremos dos tipos de recolectores de datos: uno para obtener información del modelo *model_reporters* y otro para obtener información individual de los agentes *agent_reporters*. Para obtener información del entorno usaremos funciones *lambda* que sean capaces de calcular la información en cada paso. Sin embargo, para los agentes, no es recomendable hacer esto, ya que al poder tener muchos agentes simultáneamente, reduciría considerablemente la eficiencia de la simulación. Por lo tanto, solo podremos recoger variables contenidas en los agentes para su posterior procesamiento.

Adicionalmente, tenemos que definir un tipo de **planificador** para el modelo, al que deberán añadirse los agentes para que estos funcionen. Según el tipo de planificador elegido, se comportará de manera diferente con el paso del tiempo (los pasos) dentro del simulador; el tipo elegido es *RandomActivation*. Teniendo en cuenta que la interacción entre agentes se dará dentro del entorno, esta es la mejor opción posible para maximizar la eficiencia del simulador. Este planificador conlleva no poder asegurar un orden en las acciones de los agentes. Esto quiere decir que no existirá una estructura tipo cola o similar que determine cuándo un agente es el primero en entrar o salir de una arista del grafo. Esto aunque pueda parecer un

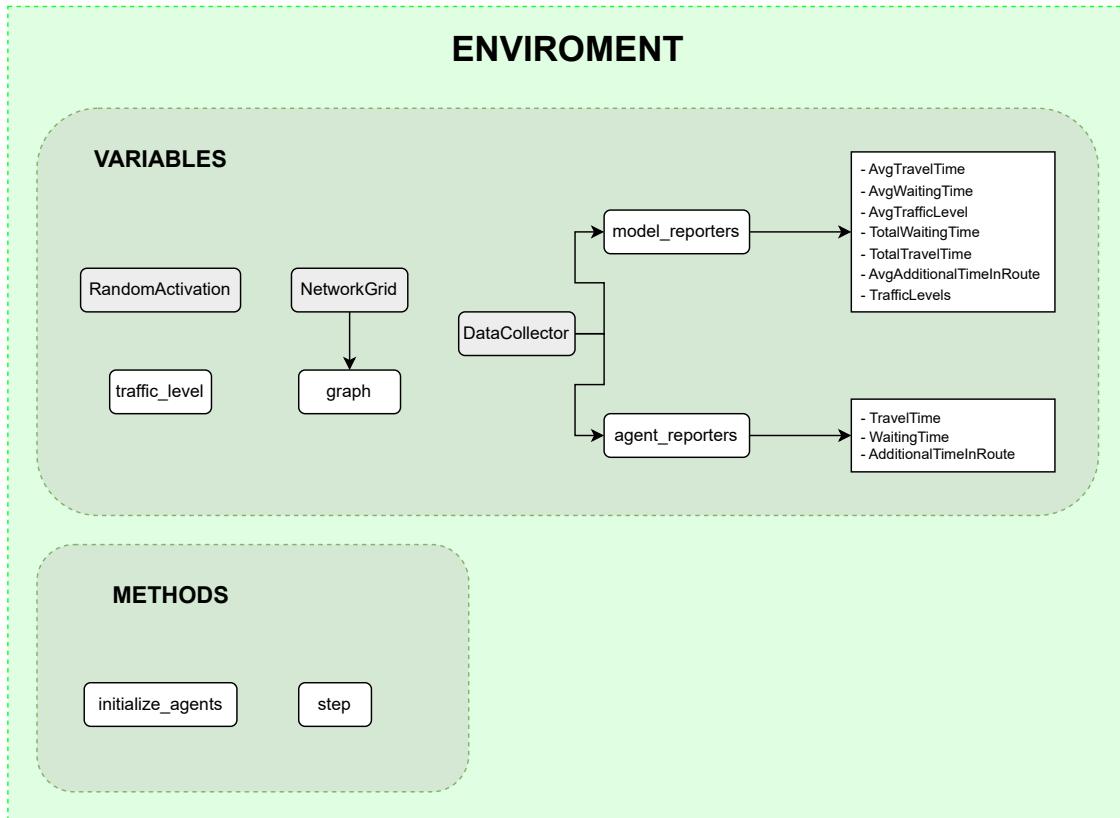


Figura 43: Diagrama del entorno del modelo.

problema grave, es interesante para un modelo como este alejado de la escala microscópica, ya que no nos interesa el efecto individual de los agentes en el tráfico; el objetivo es ser capaces de observar cómo un conjunto de vehículos con diferentes rutas influyen en el entorno en general.

Finalmente, para poder completar la inicialización del entorno y de la simulación, es necesario añadir los agentes. Se añadirán conforme a los ajustes que el usuario proporcione dentro de la interfaz, en la posición correspondiente listos para comenzar la simulación. Esto se abordará en mayor detalle cuando se expliquen los detalles de los agentes.

El diseño del entorno puede ser visualizado de forma esquemática en la fig. 43.

7.3.2. Agente

Una vez definido el entorno de nuestro simulador y el proceso de inicialización del mismo, podemos definir los agentes, los cuales necesitarán acceder a cierta información del entorno

en diversos pasos y dichos accesos deberán ser de forma controlada.

Para definir a cada agente del modelo, es necesario incluir una serie de **atributos básicos** para poder conocer en todo momento la situación en la que se encuentra. Entre estos estarían el nodo en el que se encuentra y el nodo del que proviene, o el punto específico de la ruta predefinida en el que se encuentra (si es que tiene ruta).

Añadiremos también algunas variables que nos servirán como **métricas**, y que serán recogidas por el *DataCollector* definido en el entorno, tales como el número de pasos que está desplazándose, el número de los que está esperando (quieto) o el número de pasos adicionales que va sumando en función del nivel de tráfico que haya (congestiones que se encuentra en el trayecto).

Finalmente, necesitamos definir una serie de atributos donde se recojan **características de la configuración** que el usuario haya escogido para realizar la simulación, que utilizaremos tanto para inicializar a los agentes con sus respectivas rutas como para seguir una lógica acorde a la hora de tomar acciones. Concretamente se configurarán las siguientes características:

- **Política para elegir el nodo inicial y final.** No tiene por qué seguir la misma política la elección del nodo inicial que la del final. Se han desarrollado dos opciones muy interesantes: una donde el nodo se elija a partir de una serie de **puntos de interés** dentro de una lista donde se recogen centros educativos, centros médicos, gimnasios y demás, y otra donde el nodo sea un **punto de entrada o salida** del grafo. Por lo general serán todos aquellos nodos que solo tengan una arista. También se ofrece la posibilidad de ser nodos aleatorios.
- **Método de enrutamiento de los agentes.** El agente podrá elegir el siguiente nodo en función de una ruta predefinida en el momento de su inicialización mediante un camino de *Dijkstra* con o sin pesos, con movimientos aleatorios (sin seguir ruta), o el más costoso de todos, realizar *Dijkstra* con pesos en cada paso para elegir el siguiente movimiento.
- **Habilitar la reaparición de los agentes.** Si esta opción está habilitada, significará que cuando el agente llegue a su destino volverá a aparecer un nuevo agente diferente (se generará una nueva ruta aleatoria). En caso contrario simplemente desaparecerá y la simulación podría terminar antes en caso de no haber más agentes disponibles.

Gracias a haber desarrollado esta sección siguiendo un patrón similar al **Estrategia**, el código tiene una alta flexibilidad y extensibilidad. Por ello, si en algún momento se quisieran añadir nuevos algoritmos o configuraciones para los agentes, esto no sería complejo de realizar. Además de todas las ventajas que conlleva este patrón para el mantenimiento y reutilización del código.

Una vez que tenemos todos los atributos necesarios para los agentes, es crucial definir las acciones que estos podrán realizar en cada paso. La función *step* será llamada por el planificador del entorno que hemos definido previamente, que lanzará las rutinas de cada agente. La lógica que gobierna el movimiento de los agentes y la actualización de los niveles de tráfico es independiente del proceso de selección del próximo nodo. Este nodo se determina en función de la configuración del agente y se pasa como argumento a la función *handle_movement*, la cual gestiona el movimiento:

1. Primero se verifica si el nivel de tráfico en la próxima arista es mayor que cero, lo que significaría que la arista no está completamente congestionada y el agente puede moverse.
2. A continuación tenemos que actualizar el valor de tráfico de la aristas de la que se proviene. Si el tráfico actual es menor o igual a 0.9, se incrementa en 0.1, simulando que el agente ha dejado esa arista y ha reducido la carga en ella.
3. Se actualiza también el valor de tráfico de la la arista hacia la que el agente se mueve. Se disminuye el nivel de tráfico en 0.1, simulando que el agente ahora ocupa esta arista y, por lo tanto, reduce la capacidad de tráfico disponible.
4. Y finalmente se procede a la actualización de todas las variables del agente, como el paso en el que se encuentra, el tiempo de espera si lo hubiese y añadir el tiempo adicional que ha perdido en la ruta por las congestiones del tráfico.

El diseño del agente puede ser visualizado en forma de esquema en la fig. 44.

7.3.3. Comunicación

Como comentábamos, dentro de las simulaciones basadas en sistemas agentes requieren de cierta comunicación entre los diferentes componentes. En nuestro caso, los agentes deberán

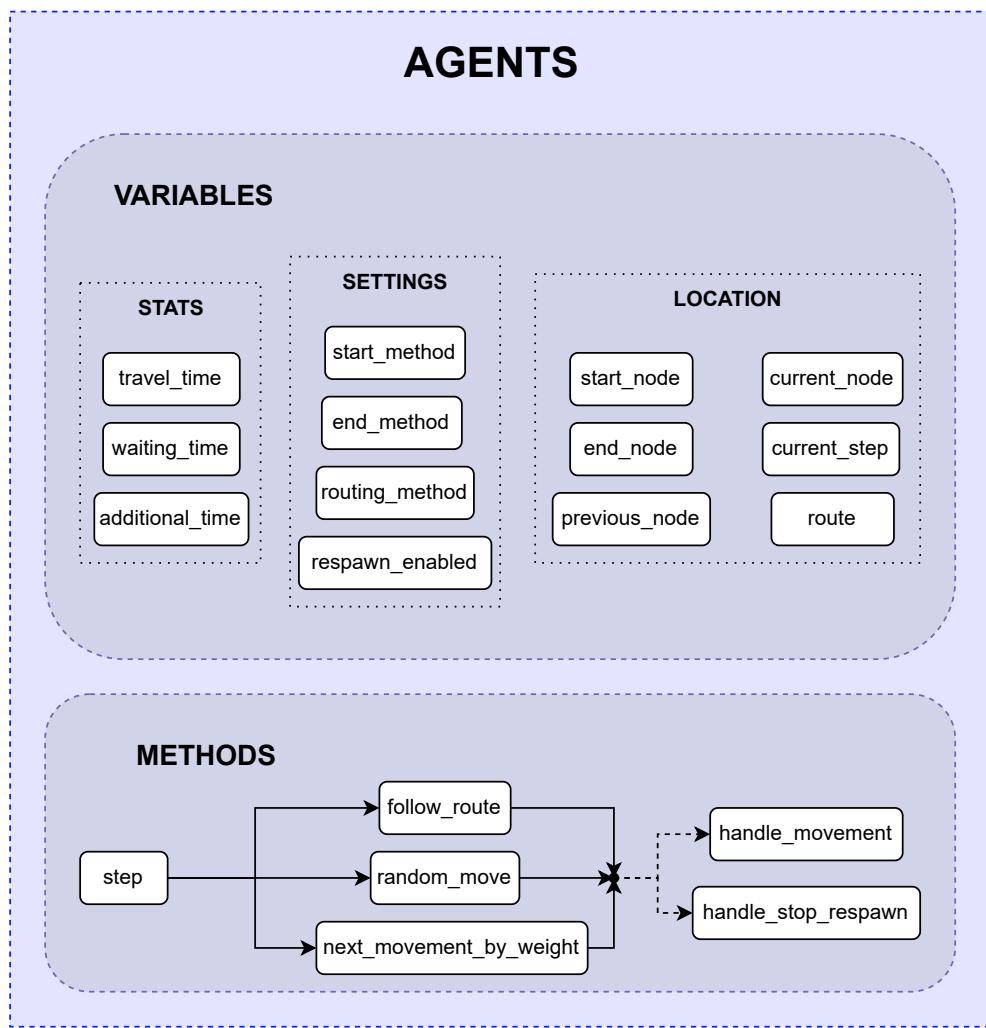


Figura 44: Diagrama de los agentes del modelo.

consultar el estado del tráfico del entorno en el momento de realizar un movimiento. Además, cuando el sistema de enrutamiento esté configurado para calcular el mejor movimiento en cada paso en función del tráfico en tiempo real, estos agentes tendrán la capacidad de observar todo el entorno disponible, a diferencia de otros agentes con sistemas de enrutamiento distintos que solo pueden observar o modificar la siguiente arista de su ruta predefinida..

Es importante distinguir en este modelo los dos tipos de agentes que tenemos según lo que vean del entorno, ya que nos puede ayudar a distinguir tipos de conductores que tendríamos en la vida real. Un usuario que simplemente se desplaza de un punto a otro, no tendría acceso a toda la carretera o posibles caminos alternativos a la hora de realizar el viaje, pero un usuario que esté siguiendo la ruta de una aplicación con conocimiento del estado de las carreteras en tiempo real, sí que tendría acceso a todas las carreteras. Esto será de gran utilidad a la hora de realizar estudios sobre las posibles consecuencias de que los usuarios utilicen aplicaciones con información en tiempo real a la hora de hacer viajes.

En la fig. 45 se muestra con más detalle el proceso de inicialización de los agentes, desde la configuración de parámetros basada en los ajustes definidos por los usuarios, hasta la asignación de su ubicación inicial. El diagrama también ilustra cómo los agentes interactúan con el entorno, realizando observaciones continuas sobre el estado de las carreteras, lo que les permite actualizar su información y ajustar sus decisiones en cada paso del proceso.

7.3.4. Salida

Con todo el modelo desarrollado, es importante analizar qué se mostrará como resultado cuando la simulación termine de ejecutarse. Ya se han definido los datos que serán recopilados tanto del entorno como de los agentes. Estos datos requieren de un procesamiento para facilitar al usuario la comprensión de los mismos. Se generarán gráficas con la siguiente información:

- Media de pasos necesarios para completar las rutas de los agentes a lo largo de la simulación.
- Tiempo adicional que los agentes consumen en la ruta a lo largo de la simulación.
- Tiempo de espera medio que los agentes están parados a lo largo de la simulación.
- Distribución de los tiempo de espera que los agentes deben estar parados.

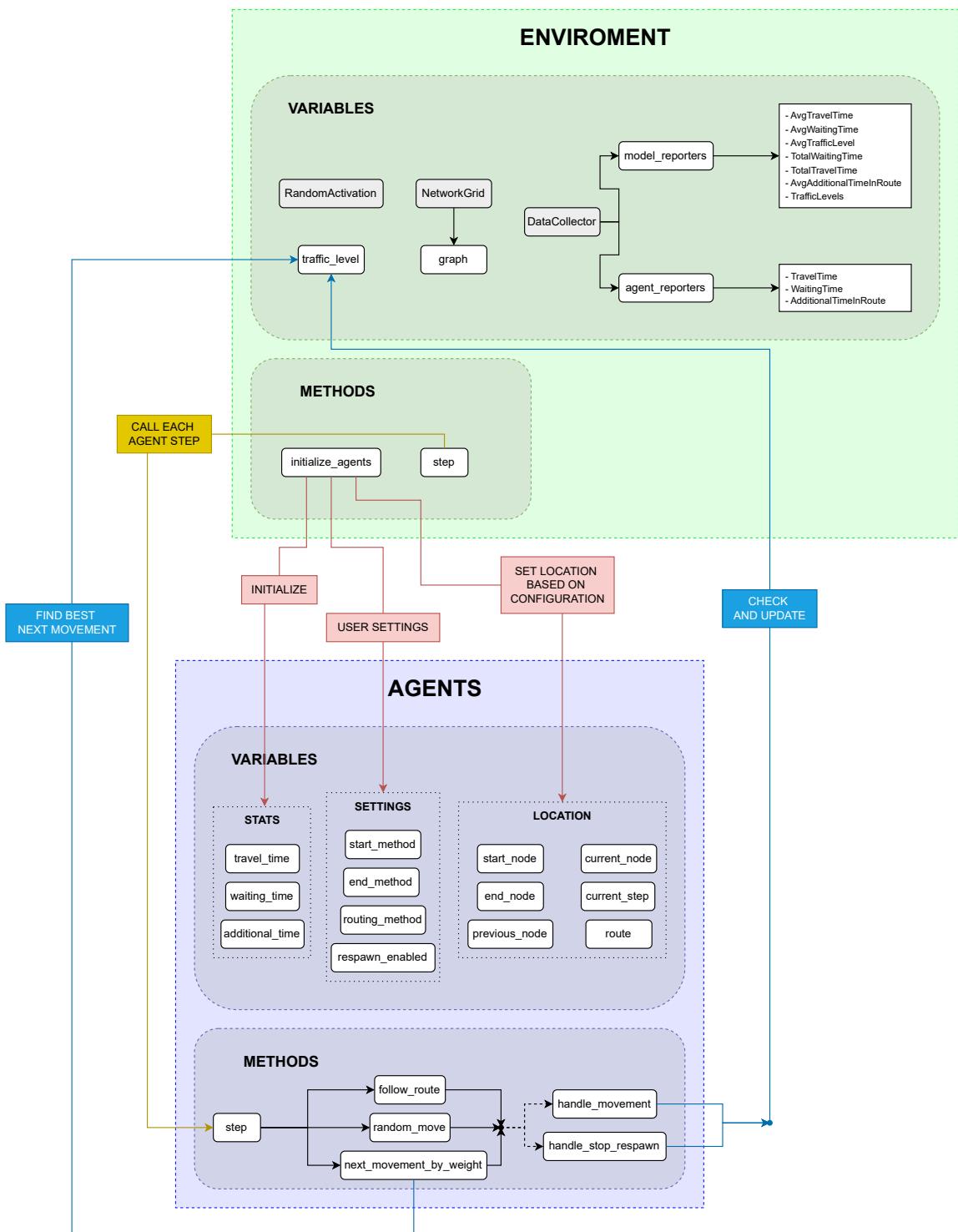


Figura 45: Diagrama del modelo.

Junto a estas gráficas se recogerán valores destacados de los datos obtenidos a partir de los recolectores de datos de los agentes, tales como media, mediana, mínimos y máximos del total de pasos consumidos en viajar, esperar y adicional en ruta.

También se aportará como salida de la simulación ejecutada un grafo con los niveles de tráfico de las aristas resultantes. Este grafo contendrá la información de las aristas en el último paso ejecutado de la simulación, y será de gran utilidad para estudiar la distribución real de los agentes en el mapa.

7.4. Interfaz de usuario

La interfaz de usuario seguirá el mismo estilo de la diseñada para la visualización y procesamiento de los datos extraídos, con un planteamiento de la interfaz en filas y columnas con tarjetas con diferentes propósitos. Aunque sean similares en estilo, se ha decidido separar ambas interfaces ya que persiguen objetivos completamente diferentes, y así se consigue evitar tener una sola interfaz confusa y sobrecargada de opciones para el usuario. Gracias a la alta cohesión y bajo acoplamiento de la interfaz desarrollada previamente, se ha podido reutilizar parte del código.

Los componentes para el mapa y la información de los nodos y aristas pulsados se han mantenido exactamente iguales, y el panel de la derecha con las opciones de visualización se ha visto modificado para recoger las opciones necesarias para esta interfaz, eliminando la opción para cambiar la fecha del nivel de tráfico mostrado en el mapa y el filtrado de información normalizada o directamente del proveedor.

Por lo tanto, quedaría por concretar el desarrollo de las siguientes partes de la interfaz:

7.4.1. Parámetros de la simulación

Esta sección será la que esté en la parte superior de la interfaz. Se trata de un panel que recoge todos **los parámetros disponibles para la simulación** que se comentaban previamente del modelo, de forma que el usuario pueda estudiar los efectos de las diferentes opciones (fig. 46 y fig. 47). El tablero se divide en cinco columnas con diferentes opciones:

- **Opciones de ejecución:** Establecer la cantidad de pasos y de agentes en la simulación y si los agentes reaparecerán una vez que alcancen su destino.

Figura 46: Captura del panel dedicado para la simulación.

Figura 47: Captura del panel dedicado para la simulación relleno.

- **Opciones de ruta:** Política para elegir el nodo inicial y final, que no tienen por qué ser la misma, y podrá ser aleatorio, puntos de interés o entrada/salida. También cuál será la ruta elegida por el agente para transitar de un punto a otro.
- **Tráfico inicial:** Al igual que en la otra interfaz se podía seleccionar un rango de fechas para procesar datos, en esta también se puede y actualizará el grafo del mapa, pero además el nivel de tráfico con el que comenzará la simulación.
- **Simulación:** Aquí el usuario podrá empezar la simulación, con una barra de carga que irá avanzando conforme los pasos de la simulación se vayan ejecutando. Además tendrá una tarjeta debajo donde podrá descargarse los datos extraídos o el grafo resultante de la simulación.
- **Visualización en el mapa:** Una vez que la simulación termine el usuario tendrá disponible un desplegable donde podrá elegir si mostrar el resultado de la simulación o los niveles de tráfico iniciales.

Una vez que la simulación termina, además de habilitar las opciones de descarga y despliegue del mapa que se comentaban, se mostrará un mensaje de alerta para que el usuario sepa que la simulación ha terminado (fig. 48).

Y como explica el mensaje, se habilitará de nuevo la opción para hacer más simulaciones y las gráficas se mostrarán al usuario en la parte inferior de la interfaz.

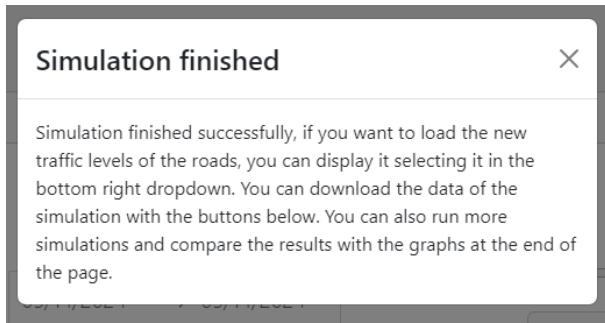


Figura 48: Captura de la alerta mostrada.

7.4.2. Resultados

Siguiendo el estilo de la otra interfaz, tenemos todo lo relativo a la simulación realizada en la sección inferior de la interfaz, gracias a la información recopilada desde la simulación en MESA.

Se mostrarán cuatro gráficas: tres serán distribuciones temporales conforme van ejecutándose los pasos de la simulación. Se muestra la media de movimientos que tardan los agentes en completar la ruta, la media de movimientos que no puede realizar cada agente por la congestión del tráfico y la media de movimientos adicionales que cada agente pierde en cada arista por el nivel de tráfico. La restante será una distribución con la cantidad de pasos bloqueados en total; esta se mostrará en escala logarítmica para poder apreciar mejor la información.

Volveremos a usar *Dash Plotly*, como con la otra interfaz. De esta forma el usuario que utilice la aplicación puede seguir usando las funciones relativas a las gráficas que esta librería ofrece y se mantiene la consistencia entre ambas interfaces.

A diferencia de la interfaz desarrollada para el procesamiento de los datos, si el usuario realiza múltiples simulaciones, irán apareciendo las nuevas colecciones de datos dentro de cada gráfica, junto a unos nombres que de forma comprimida. Esto facilitará al usuario hacer comparaciones probando diferentes parámetros de la simulación.

En la fig. 49 se puede observar cómo, según los parámetros de la simulación, se genera un nombre descriptivo que permite al usuario diferenciar fácilmente las distintas simulaciones en las gráficas. Indicando el nivel de tráfico inicial, pasos, agentes, política para el nodo inicial y final, método de enrutamiento y si la reaparición está habilitada o no.

Como información extra a las gráficas, se ofrece una tabla con algunos valores destacados

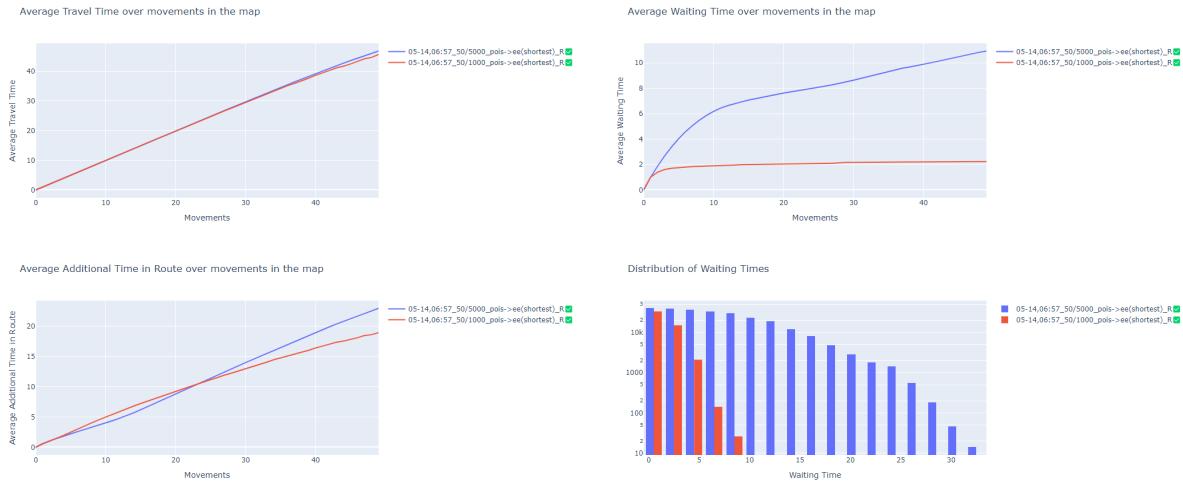


Figura 49: Captura de las gráficas con dos simulaciones.

Last simulation's summary statistics

Stat	TravelTime	WaitingTime	AdditionalTimeInRoute
count	50000	50000	50000
mean	23.76504	1.0227	10.431771670061757
std	14.414646351506121	1.2424071721909973	7.087404288770766
min	0	0	0
25%	11	0	4.799999999999999
50%	23	1	9.599999999999998
75%	36	2	15.09999999999994
max	49	9	36.59999999999994

Figura 50: Captura de la tabla tras realizar una simulación.

de la última simulación realizada (fig. 50).

7.5. Conclusiones

Este simulador no solo es una demostración técnica, si no que también representa una herramienta valiosa para urbanistas, ingenieros de tráfico y autoridades municipales, quienes pueden utilizarlo para mejorar la planificación urbana, la eficiencia de las redes viales o estudiar los focos de congestión del tráfico.

La sección final del proyecto, en la que se desarrolla el modelo basado en agentes y su correspondiente interfaz de usuario, queda resumida con el diagrama de la fig. 51.

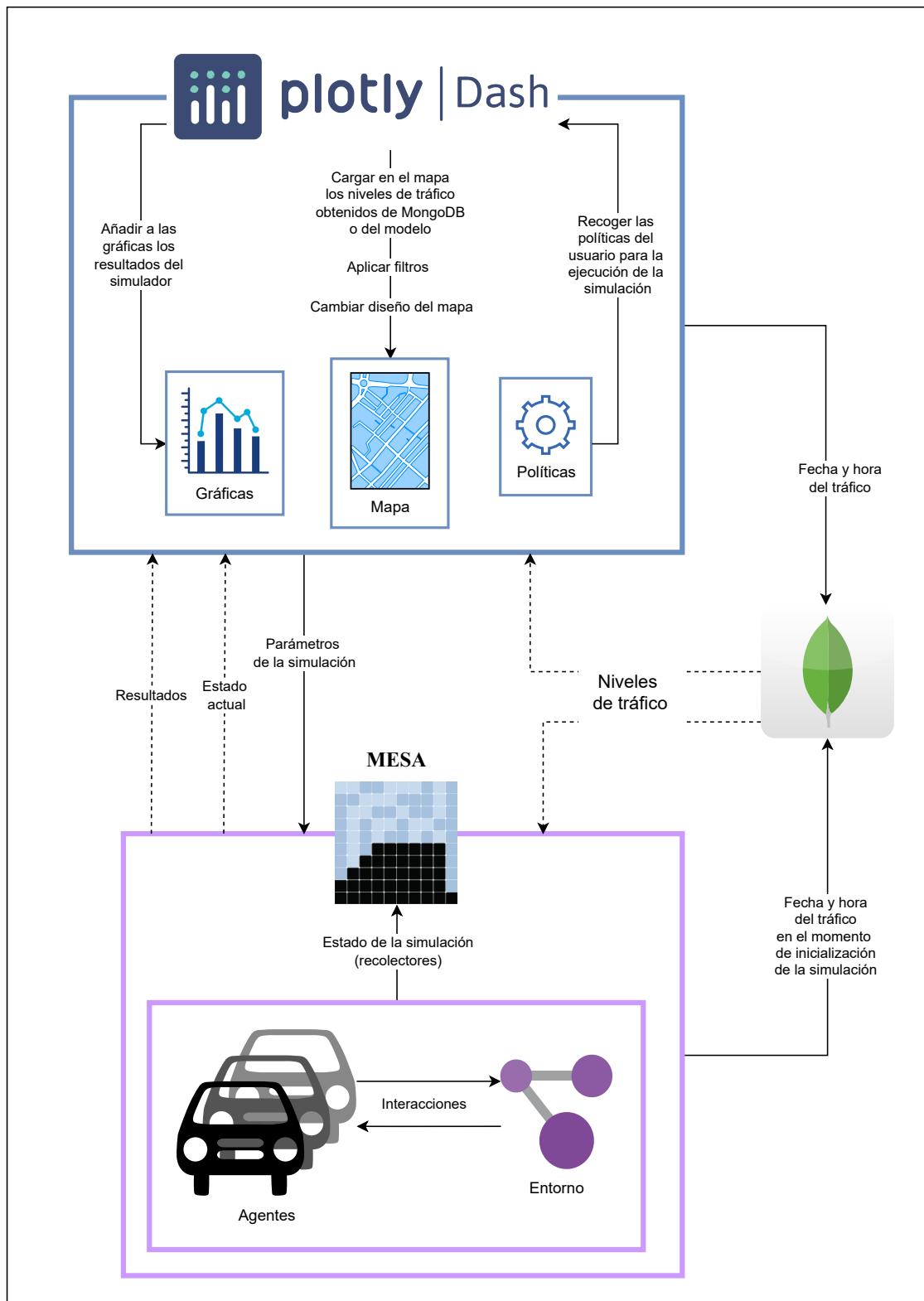


Figura 51: Diagrama general del modelo y su *dashboard*.

8

Pruebas de carga

8.1. Introducción

Las **pruebas de carga** son un tipo fundamental de **prueba no funcional** cuyo objetivo es evaluar el rendimiento de un sistema bajo condiciones específicas. En el contexto de este proyecto, se dividen en dos partes: una para la visualización y análisis de datos de tráfico en un rango de tiempo, y otra para la realización de simulaciones de tráfico con agentes.

Se ha llevado a cabo una **prueba de estrés** para evaluar la capacidad de cada sistema bajo condiciones extremas de carga. Este tipo de prueba asegura que la aplicación pueda gestionar picos significativos en la demanda, lo cual es fundamental para aplicaciones que requieren un alto rendimiento y fiabilidad en situaciones de alta carga. Esta se caracteriza por la **carga incremental**: se incrementa gradualmente hasta alcanzar o incluso superar los niveles esperados en situaciones de uso extremo. Esto permite observar cómo el sistema responde bajo condiciones de alta demanda. (fig. 52)

Los resultados de estas pruebas de carga proporcionan una visión integral del **rendimiento y la fiabilidad de las aplicaciones** bajo condiciones de uso intensivo. A través de estas pruebas, se garantiza que ambos sistemas pueden ofrecer una **experiencia de usuario eficiente**.

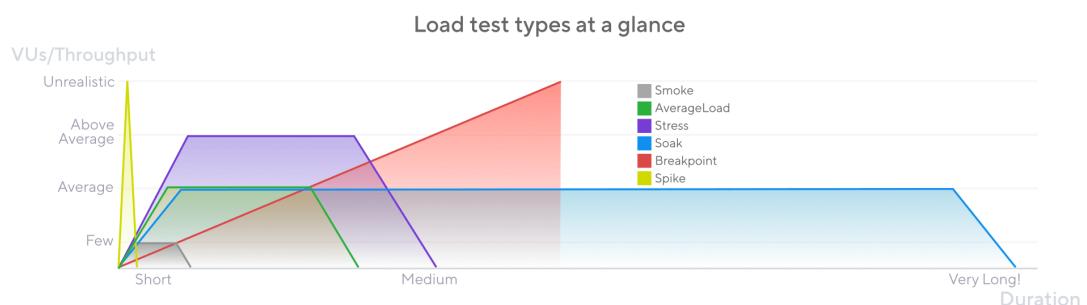


Figura 52: Diagrama de *Grafana* con los tipos de pruebas de carga [8].

Componente	Especificación
Procesador	i7-13700 2.10 GHz
Memoria RAM	16,0 GB (15,7 GB usable)
Tarjeta gráfica	Intel(R) UHD Graphics 770

Cuadro 13: Especificaciones del equipo de prueba

ciente y sin interrupciones, independientemente del volumen de datos procesados o la complejidad de las simulaciones ejecutadas.

Las pruebas de carga no solo proporcionan una medida clara de la capacidad del sistema para realizar operaciones bajo estrés, si no que también revelan aspectos críticos como:

- **Escalabilidad:** La capacidad del sistema para crecer y manejar mayores volúmenes de datos o simulaciones más complejas.
- **Robustez:** La capacidad del sistema para mantenerse estable y funcional incluso cuando es sometido a condiciones extremas.
- **Identificación de Cuellos de Botella:** Localización de componentes o procesos dentro de la aplicación que podrían limitar su rendimiento y que, por lo tanto, podrían necesitar optimización.

Para estas pruebas es muy importante tener en cuenta el equipo en el que se ejecutan (tabla 13). Tanto las aplicaciones web como la base de datos se encontrarán para todas las pruebas ejecutándose localmente. Por lo tanto, hay que remarcar que con un equipo de mejores prestaciones obtendríamos mejores resultados y viceversa. En este caso las pruebas han sido realizadas en el siguiente equipo:

8.2. *Dashboard de los datos de tráfico*

En este proyecto solo se han podido recopilar datos de aproximadamente 4 meses, entre mayo y septiembre. La cantidad de datos de estas fechas no son suficientes para hacer un análisis temporal de uso realista, ya que, tenemos una operación altamente costosa: el procesamiento de los datos para la generación de las gráficas.



Figura 53: Captura de pantalla de *MongoDB Compass* con las fechas con información disponible

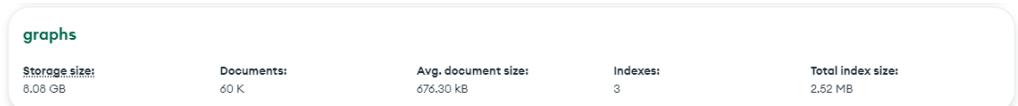


Figura 54: Captura de pantalla de *MongoDB Compass* con los grafos informados

Es necesario realizar algunas pruebas al sistema para garantizar unos parámetros temporales mínimos en función del rango de fechas seleccionado por el usuario. Para suplir la falta de información real con la que hacer las pruebas, se ha creado una colección adicional en *MongoDB*, a la que se han añadido datos reales del periodo comprendido entre el 8 y el 24 de mayo de 2024 multiplicados por 100, es decir, **procesar un día sería equivalente a procesar 100 días** dentro de esta colección. Con esta colección ficticia de aproximadamente 8 GB podemos hacer diversas pruebas de rendimiento (fig. 53 y fig. 54).

Se han ejecutado algunas agregaciones de *MongoDB* para comprobar que verdaderamente tenemos los datos multiplicados por 100 en este rango de fechas. Como reflejan las siguientes imágenes (fig. 55 y fig. 56), en la colección de grafos tenemos 596 fechas únicas con información, y cada una de estas repetida 100 veces, concretamente entre el 8 de mayo y el 24 de mayo.

Una vez verificado el conjunto de datos que usará el sistema, podemos realizar las siguientes pruebas:

- **Prueba de máxima carga:** Hemos realizado una prueba para estudiar cuánto tiempo tarda en procesar toda la información disponible. Tenemos 596 grafos únicos con infor-

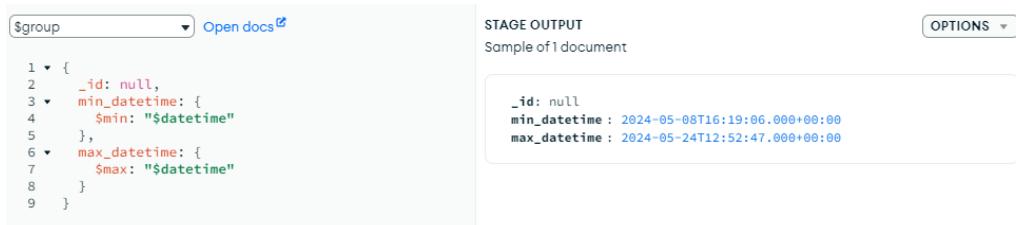


Figura 55: Captura de pantalla de *MongoDB Compass* con el rango de fechas disponible



Figura 56: Captura de pantalla de *MongoDB Compass* con el recuento de grafos con información única

mación (16 días), y cada uno multiplicado por 100 instancias repetidas, lo que hace un total de 596 000 grafos a procesar (1.600 días), o lo que es lo mismo, aproximadamente **4 años y medio** de información para procesar. El tiempo que tarde *MongoDB* en procesar esta información es crucial para saber si la plataforma es realmente útil en análisis de grandes volúmenes de datos. Los resultados de tiempo obtenidos han sido los siguientes:

- **Por nombre:** 701.30 segundos
- **Por rango de horas:** 606.53 segundos
- **Por día de la semana:** 443.38 segundos

Aunque a priori pudiera parecer mucho tiempo, **29 minutos** para procesar 4 años y medio, o como podemos observar en la fig. 54, 8 *Gigabytes* de datos, no es un mal resultado. Procesando estos volúmenes de información obtenemos un tiempo aproximando de un segundo por día.

- **Prueba de 2 años:** Siguiendo la misma lógica, para hacer una prueba en la que procesemos 2 años de información, deberemos coger 7 días de información ya que dos años son $365 * 2 = 730$ días y una semana de datos son realmente $7 * 100 = 700$ días. Se han obtenido estos resultados:
 - **Por nombre:** 303.26 segundos
 - **Por rango de horas:** 425.94 segundos
 - **Por día de la semana:** 238.18 segundos

Podemos observar cómo con la mitad de información obtenemos unos resultados muy similares en lo que a tiempo consumido por día respecta. Podríamos procesar 2 años

de información en **16 minutos**, lo cual, considerando el rango de tiempo que estamos procesando es más que aceptable.

- **Prueba de 1 año:** Para hacer una prueba en la que procesemos 1 año de información, deberemos seleccionar un rango de 4 días en la interfaz ya que un años son 365 días y cuatro días de datos son realmente $4 * 100 = 400$ días. Los resultados han sido:

- **Por nombre:** 147.94 segundos
- **Por rango de horas:** 142.88 segundos
- **Por día de la semana:** 122.84 segundos

Si se reduce el volumen de información a la mitad con respecto a la prueba anterior, podemos observar que el tiempo consumido es una tercera parte que el de la prueba anterior, algo menor a **7 minutos**. Por lo tanto, veremos el tiempo de procesado por día reducido a 0.88 segundos aproximadamente.

- **Prueba de 6 meses:** Finalmente, haremos una prueba con una cantidad de datos más reducida. Deberemos seleccionar un rango de 2 días en la interfaz ya podemos decir que dos meses son $6 * 30 = 180$ y dos días de datos equivalen a $2 * 100 = 200$ días. Con lo que obtenemos estos tiempos:

- **Por nombre:** 79.03 segundos
- **Por rango de horas:** 111.89 segundos
- **Por día de la semana:** 79.97 segundos

En la última prueba, hemos reducido de nuevo el tiempo de procesado por día a 0.66 segundos aproximadamente, ya que el tiempo total ha sido de **4 minutos y medio**, así que podemos asumir que a menor volumen de datos, mayor velocidad en el procesado de los datos. No tiene sentido hacer pruebas con tiempos menores, ya que podremos asumir que tardará aproximadamente medio segundo por día, lo cual es un buen tiempo de procesamiento de datos.

A modo de síntesis, podemos observar la cantidad de información procesada y el tiempo consumido para ello en la tabla **14**. Donde cada fila corresponde a una cantidad determinada

Procesado por tiempo	Nombre de vía	Rango de horas	Día de la semana	Total
4 años de información (aprox. 8 GB)	701.30	606.53	443.38	1.751.21
2 años de información (aprox. 4 GB)	303.26	425.94	238.18	967.38
1 año de información (aprox. 2 GB)	147.94	142.88	122.84	413.66
6 meses de información (aprox. 1 GB)	79.03	111.89	79.97	270.89

Cuadro 14: Tabla con los tiempos de procesamiento, en segundos, para cada caso

de información y cada columna corresponde a el tiempo en segundos dedicado a cada tipo de procesamiento, siendo la última el tiempo total de todos los procesamientos. Los resultados obtenidos han sido buenos, y podemos garantizar unos tiempos adecuados para los usuarios que usen la aplicación, y además, se tiene la posibilidad de mejorarlos con una máquina mejor o cuyo propósito sea únicamente este procesamiento de datos. Además en la gráfica de la fig. 57, podemos ver una representación más clara de los tiempos.

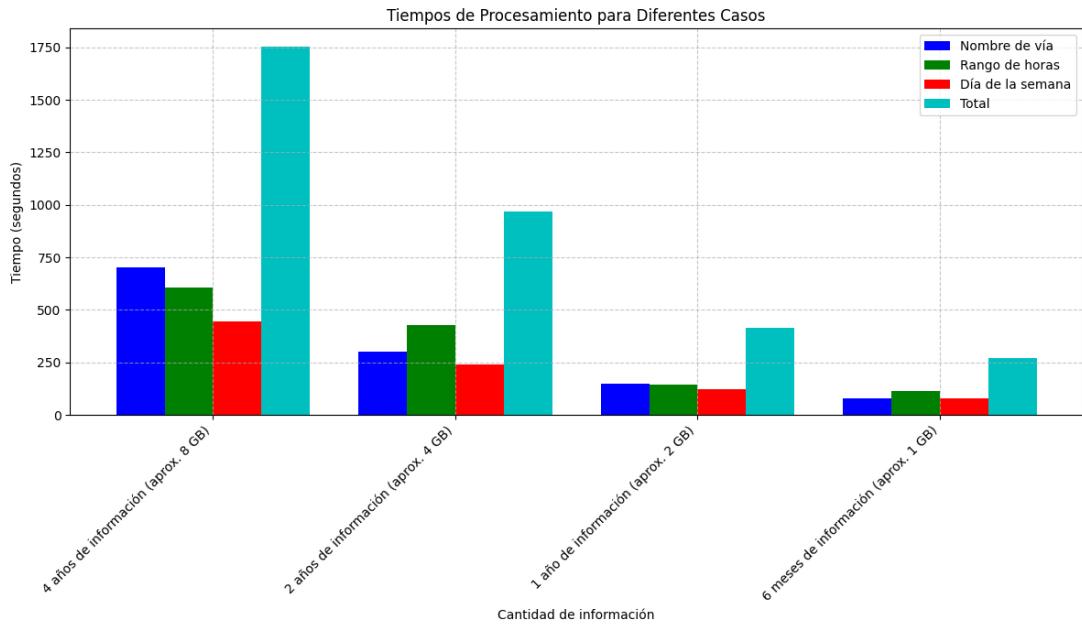


Figura 57: Gráfica con los tiempos de carga

8.3. Simulador de flujo de tráfico

En el simulador de flujo de tráfico, el usuario tiene la capacidad de **configurar diversos parámetros** que afectan el comportamiento y rendimiento del sistema. Entre estas configuraciones se encuentran el **número de agentes** (que representan vehículos en el flujo de tráfico) y el **número de pasos** (que simulan el tiempo transcurrido en la simulación). Para las pruebas que se describen a continuación, se ha decidido permitir la reaparición de los agentes, de modo que una vez que completen su ruta, se reinician desde su punto de partida, lo que permite una simulación continua y realista del tráfico.

Además, el simulador cuenta con **tres sistemas de enrutamiento** diferentes:

- **Sistema de enrutamiento 1 y 2:** Estos sistemas establecen la ruta de cada agente durante la inicialización, es decir, al comienzo de la simulación o en el momento de reaparición del agente. El coste computacional de esta operación es $O(1)$ para cada agente, ya que no requiere cálculos adicionales durante los pasos de la simulación.
- **Sistema de enrutamiento 3:** A diferencia de los anteriores, este sistema recalcula la ruta en cada paso de la simulación utilizando el algoritmo de Dijkstra. Este proceso es mucho más costoso computacionalmente debido a la complejidad del algoritmo de

Dijkstra, que es $O(V^2)$ en su implementación más simple, donde V es el número de vértices en el grafo. Esta complejidad podría aumentar significativamente el tiempo de simulación, teniendo en cuenta que además será ejecutada en cada paso una vez por agente.

Dada la naturaleza de los sistemas de enruteamiento, se espera que el sistema de enruteamiento 3, que recalculará la ruta en cada paso, sea considerablemente más costoso en términos de tiempo de ejecución. Por esta razón, se llevarán a cabo pruebas específicas para cuantificar la diferencia de tiempo entre los sistemas.

Para evaluar el rendimiento del simulador con diferentes configuraciones, se han diseñado las siguientes pruebas:

- **Pruebas con ruta inicial (Coste $O(1)$):**

- 1000 pasos con 50 agentes.
- 1000 pasos con 100 agentes.
- 1000 pasos con 1000 agentes.
- 1000 pasos con 2500 agentes.
- 1000 pasos con 5000 agentes.

- **Pruebas con recálculo de ruta en cada paso (Coste del algoritmo de Dijkstra):**

- 1000 pasos con 50 agentes.
- 1000 pasos con 100 agentes.
- 1000 pasos con 1000 agentes.
- 1000 pasos con 2500 agentes.
- 1000 pasos con 5000 agentes.

Justificación del número de pasos: No se considera necesario variar la cantidad de pasos en estas pruebas debido a la naturaleza de la simulación. Dado que las rutas de los agentes suelen oscilar entre 80 y 200 pasos, y que se permite la reaparición de los mismos, el sistema no

se verá afectado de manera significativa por un mayor número de pasos. En cambio, es mucho más relevante y efectivo analizar cómo varía el tiempo de ejecución en función del número de agentes, ya que esto proporcionará una visión más clara del rendimiento del simulador bajo diferentes cargas de trabajo.

Los resultados de las pruebas definidas han sido los siguientes:

Nº de Agentes	Tiempo (s)
50	3.75
100	6.13
1000	56.58
2500	127.39
5000	276.53

Cuadro 15: Resultados con enrutamiento en el momento de inicialización.



Figura 58: Gráfica con los tiempos de carga

- **Enrutamiento calculado en el momento de inicialización:**

Nº de Agentes	Tiempo (s)
50	75.72
100	149.66
1000	2026.42
2500	3950.79
5000	9470.61

Cuadro 16: Resultados con enrutamiento complejo basado en Dijkstra.

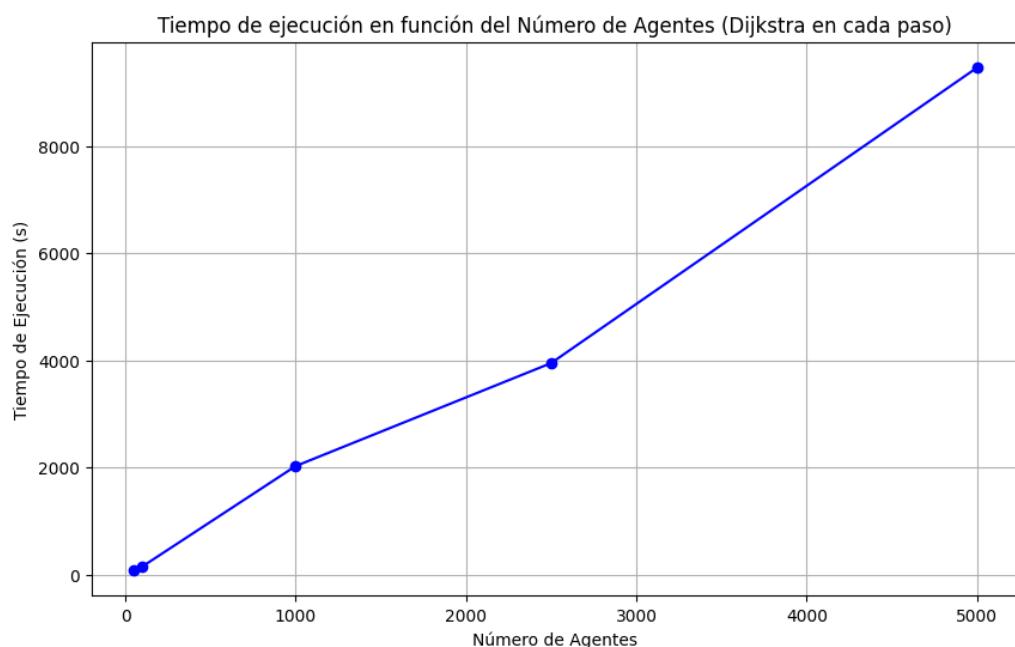


Figura 59: Gráfica con los tiempos de carga

- **Enrutamiento calculado para cada agente en cada paso:**

Simulation's stats with predefined route

Stat	TravelTime	WaitingTime	AdditionalTimeInRoute
count	2500000	2500000	2500000
mean	47.9075044	0.4491288	14.268342059898039
std	31.918690444394805	1.7202189699779156	12.251412403293791

Figura 60: Tabla de valores destacados de la simulación con ruta predefinida

Simulation's stats with routing in each step

Stat	TravelTime	WaitingTime	AdditionalTimeInRoute
count	2500000	2500000	2500000
mean	47.825186	0.407332	14.0977300088151
std	31.9242417377185	1.5711196100687972	11.942059707837792

Figura 61: Tabla de valores destacados de la simulación con ruta calculada en cada paso

A pesar de que la evolución de los tiempos es muy parecida (fig. 58 y fig. 59). Se puede apreciar el enorme incremento de tiempo que supone calcular Dijkstra para cada uno de los agentes en cada paso, llegando a un aumento de más de 30 veces el tiempo con una ruta predefinida en el momento de inicialización del agente (tabla 15 y tabla 16).

Con respecto a los resultados obtenidos en las simulaciones, es posible observar una mejoría con respecto a los tiempos adicionales y los de espera, en el caso de la ruta en cada paso, como podemos observar en la fig. 60 y la fig. 61 con 2.500 agentes.

Era de esperar que al calcular la mejor ruta posible en cada paso mejoraran los valores obtenidos, pero no lo suficiente como para tener que esperar 12 veces más la ejecución de la simulación. En parte, podemos explicar esta leve mejoría en los tiempos de espera por la estructura de las vías, ya que, a pesar de que la ruta sea la mejor, la red de carreteras se encuentra saturada cuando incorporamos 5000 agentes al sistema.

Las gráficas fig. 62 y fig. 63 también nos demuestran la mejoría que supone usar enrutamiento en tiempo real en lo que a tiempo adicional en el trayecto por congestiones se refiere, sobre todo podemos apreciarlo en la reducción de los picos, lo que quiere decir que este método de enrutamiento es mucho más útil en momentos en los que más agentes inicien su viaje simultáneamente, lo que se traduce a hora punta en el tráfico real.

Esto podría ser usado por alguna plataforma de enrutamiento conocida como *Google Maps* o *Waze*, que establecen una ruta para los usuarios antes de comenzar el viaje y

Average Additional Time in Route over movements in the map

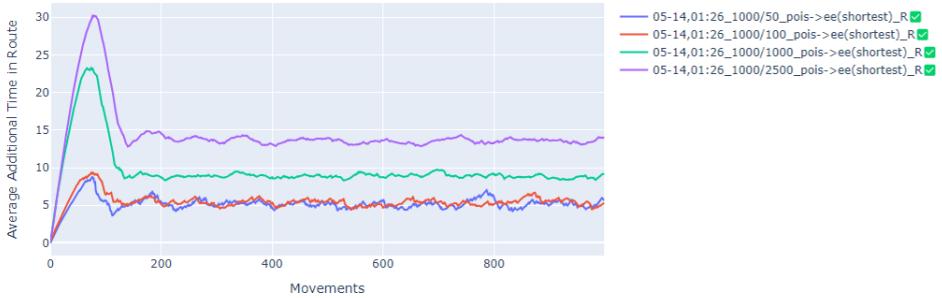


Figura 62: Gráfica con la media de tiempo adicional en las rutas de los agentes en la simulación con rutas predefinidas

Average Additional Time in Route over movements in the map

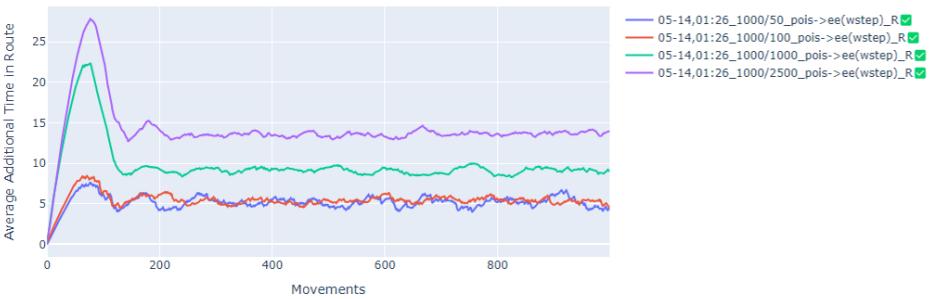


Figura 63: Gráfica con la media de tiempo adicional en las rutas de los agentes en la simulación con rutas calculadas en cada paso

quizás deberían dirigir su esfuerzo en buscar rutas dinámicas en horas puntas, analizando el mejor camino y reduciendo así las posibles emisiones y aglomeraciones masivas de coches.

9

Conclusiones y Líneas Futuras

9.1. Conclusiones

El proyecto realizado consta de **tres partes** bien definidas pero que comparten un mismo objetivo común, ofrecer una herramienta novedosa y útil para el análisis del flujo del tráfico, en nuestro caso centrada en la zona universitaria de Málaga, Teatinos.

En primer lugar, tras realizar un estudio exhaustivo, se identificó un proveedor de la **información del tráfico**, y se desarrolló una **aplicación para obtenerla y procesarla**. Esta información en bruto sirvió para de nuevo procesarla posteriormente y trasladarla a una estructura útil y eficiente como es un grafo. El refinamiento de la información fue exitoso, permitiendo trasladar la gran mayoría de la información disponible a la estructura, y gracias al diseño de un algoritmo de normalización de tráfico, se consiguió obtener una estructura con información detallada sobre el tráfico. Esta estructura de tipo grafo también permitió el almacenamiento de la información del tráfico de una manera eficiente en la base de datos.

Una vez que se obtuvieron y procesaron los datos y se definió la estructura que tendrían, se desarrolló una **aplicación para su visualización**, ofreciendo simultáneamente herramientas para el análisis de grandes volúmenes de datos con una serie de filtros personalizables. Con las pruebas realizadas se garantiza que cualquier usuario podrá analizar un año de información en un periodo de tiempo de 6 minutos aproximadamente, tiempo más que aceptable teniendo en cuenta lo que podría tardar una persona sin la herramienta.

A continuación, se desarrolló un **modelo basado en agentes** para realizar **simulaciones de tráfico**. Este modelo hará uso de los grafos almacenados previamente, utilizándolos como punto de partida. Las simulaciones colocarán una serie de agentes en diferentes puntos del

mapa, y estos se desplazarán hasta el punto que tengan como objetivo, representando el flujo de tráfico con el paso del tiempo en el entorno. Cada agente representa aproximadamente un vehículo grande, y el paso de tiempo del entorno es por pasos.

Gracias a seguir unas buenas técnicas de desarrollo software y diseño de interfaces, la aplicación desarrollada para el análisis de los datos sirvió como base para implementar la siguiente aplicación, enfocada en ofrecer una **interfaz de usuario** simple e intuitiva para hacer uso **del simulador de tráfico**, de forma que no haga falta acceder a consola para usar el simulador.

Ambas interfaces proponen la misma idea: **ofrecer los resultados en forma de gráficas** generadas automáticamente por la aplicación, pero ofreciendo al usuario la posibilidad de **descargar los datos** resultantes para realizar estudios más complejos o de otros valores diferentes. Permitir al usuario usar otras herramientas complementarias a las desarrolladas, hará que disfrute de una experiencia más completa y enriquecedora.

9.2. Dificultades encontradas

En esta sección se comentarán algunas de las dificultades encontradas a lo largo del desarrollo del proyecto y que soluciones se dieron.

9.2.1. Datos de tráfico

Desde el primer momento de desarrollo, se buscó cual era la mejor **forma de obtener la información** del proveedor. Hacer una petición por cada vía era la opción más sencilla, pero esta tenía una **escalabilidad** muy baja, aumentando significativamente el volumen de información recibida conforme se ampliaba la zona de estudio. Por lo tanto, se decidió elegir otro método, mediante *tiles* o azulejos en el mapa, que ofrecen información en secciones cuadradas del mapa de una manera más eficiente y reduciendo así la cantidad de peticiones considerablemente. Para la zona elegida eran necesarias dos *tiles* de información (fig. 8).

Esta información estaba **codificada**, y con la ayuda de algunas librerías se logró decodificar. El formato en el que venía la información era un *GeoJSON* con coordenadas relativas, es decir, ambas *tiles* se superponían y hubo que traducirlas a coordenadas reales teniendo en cuenta las coordenadas reales de las esquinas de cada *tile* y posteriormente fusionar ambos

ficheros.

Finalmente, para trasladar la información a una estructura, se extrajo un **grafo** gracias a OSMx. Este grafo fue procesado para asegurar que estuviera completamente cerrado, sin discontinuidades fuera de la zona de estudio, y con entradas y salidas claramente definidas para que los agentes pudieran navegarlo sin problemas. Para hacer el traslado fue necesario hacer una disección de los elementos del archivo *GeoJSON* en elementos más pequeños para aumentar la precisión, y con la ayuda de algunas librerías, se pudo encontrar que arista del grafo correspondía a cada uno de los elementos del fichero tipo *GeoJSON*. Se tuvieron en cuenta las coordenadas y la dirección en la que apuntaban los elementos para que no hubiera errores.

9.2.2. Representación del tráfico en el mapa

En primer lugar, **no existía un componente tipo mapa** proporcionado oficialmente por el *framework* donde se desarrollaron las aplicaciones **que permitiera el uso de grafos**. Se barajaron otras opciones de representación, pero ninguna conseguía ser tan eficiente, clara e interactiva como un grafo. Así que tras una larga búsqueda intensiva en foros comunitarios de la plataforma, se encontró una librería pública que permitía el uso de esta estructura en los mapas [4].

La librería no ofrecía ninguna facilidad a la hora de **representar las carreteras de doble sentido**, que aparecían superpuestas. Podría no parecer importante, pero se perdía mucha información al no poder visualizar la información de los dos sentidos, que en muchas ocasiones diferían entre sí. La mejor opción posible fue desarrollar dentro de la aplicación una función para que al pulsar sobre una arista del grafo, dibujase flechas correspondientes a cada sentido con el color del nivel de tráfico, de forma que el usuario no se pierda información (section 6.4.4).

9.3. Líneas Futuras

- **Automatizar el procesamiento de los datos previo a su almacenamiento:** Para procesar los datos, es necesario ejecutar un programa cada vez que se quiera añadir información a la base de datos. Una mejora que podría desarrollarse en un futuro es incluir este procesamiento de manera automática una vez que se haga la petición al proveedor, de forma que el usuario no tiene que preocuparse por el tratamiento de los

datos.

- **Añadir más zonas de Málaga para estudiar:** Esto no supondría gran dificultad, pero sí un significativo aumento en el uso de recursos del dispositivo para almacenar, procesar y analizar volúmenes tan grandes de información.
- **Mejoras en los análisis de datos realizados:** Las gráficas ofrecidas por la aplicación puede que no sean suficientes para realizar determinados tipos de estudios de las carreteras. Para añadir nuevas gráficas sería necesario reunirse con diferentes ingenieros civiles o equipos urbanistas para analizar qué gráficas son necesarias.
- **Capacidad para introducir nuevos grafos:** A lo largo del proyecto se planteó la posibilidad de poder modificar las carreteras existentes a la hora de hacer las simulaciones para poder estudiar cómo afectaría la creación o modificación de algunas vías. Aunque la idea fue descartada por la enorme complejidad que suponía, podría ser una línea futura de trabajo.
- **Mejoras al modelo:** Simulaciones con el tráfico en tiempo real, tener diferentes tipos de agentes (camión, coche, moto o bicicleta) o que los semáforos funcionen en el simulador (y poder estudiar diferentes políticas en ellos) son algunos cambios que podrían hacerse para mejorar el realismo del simulador.

Referencias

- [1] Ali Bazghandi. «Techniques, advantages and problems of agent based modeling for traffic simulation». En: *International Journal of Computer Science Issues (IJCSI)* 9.1 (2012), pág. 115.
- [2] Geoff Boeing. *Modeling and Analyzing Urban Networks and Amenities with OSMnx*. Working paper. 2024. URL: <https://geoffboeing.com/publications/osmnx-paper/>.
- [3] Jaume Ferré-Bigorra, Miquel Casals y Marta Gangolells. «The adoption of urban digital twins». En: *Cities* 131 (2022), pág. 103905.
- [4] Alberto Garcia-Robledo y Mahboobeh Zangiabady. *Dash Sylvereye: A WebGL-powered Library for Dashboard-driven Visualization of Large Street Networks*. English. Working-Paper. Re-submitted to IEEE Access on Aug. 11, 2023. The interpretation of the results in Section V has been corrected, as a more in-depth analysis unveiled that the prior results are attributed to the software (CPU) acceleration capabilities of Dash Sylvereye. Additionally, the manuscript now features a performance comparison with Kepler.gl and city-roads. ArXiv.org, mayo de 2021. doi: [10.48550/arXiv.2105.14362](https://doi.org/10.48550/arXiv.2105.14362).
- [5] GDAL/OGR contributors. *GDAL/OGR Geospatial Data Abstraction software Library*. Open Source Geospatial Foundation. 2024. doi: [10.5281/zenodo.5884351](https://doi.org/10.5281/zenodo.5884351). URL: <https://gdal.org>.
- [6] Sean Gillies et al. *Shapely: manipulation and analysis of geometric objects*. toblerity.org, 2007. URL: <https://github.com/Toblerity/Shapely>.
- [7] Inc. GitHub. *GitHub*. Platform for version control and collaborative development. 2024. URL: <https://github.com/>.
- [8] Grafana Labs. *Load Testing: Types, Tools, and Best Practices*. 2024. URL: <https://grafana.com/docs/k6/latest/types-of-load-testing/> (visitado 14-05-2024).
- [9] M Grieves. «Conceptual ideal for PLM». En: *Presentation for the Product Lifecycle Management (PLM) center, University of Michigan* (2002).

- [10] Michael Grieves y John Vickers. «Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems». En: *Transdisciplinary perspectives on complex systems: New findings and approaches* (2017), págs. 85-113.
- [11] Michael W Grieves. «Digital twins: past, present, and future». En: *The digital twin*. Springer, 2023, págs. 97-121.
- [12] Aric Hagberg, Pieter J Swart y Daniel A Schult. *Exploring network structure, dynamics, and function using NetworkX*. Inf. téc. Los Alamos National Laboratory (LANL), Los Alamos, NM (United States), 2008.
- [13] Mervi Hämäläinen. «Urban development with dynamic digital twins in Helsinki city». En: *IET Smart Cities* 3.4 (2021), págs. 201-210.
- [14] Paul Hendricks, Allan Enemark y Ajay Thorve. *Interactively Visualizing a Drivetime Radius from Any Point in the US*. <https://developer.nvidia.com/blog/interactively-visualizing-a-drivetime-radius-from-any-point-in-the-us>. 11 de mayo de 2021. (Visitado 14-03-2024).
- [15] MongoDB Inc. *MongoDB Community Server Installation Tutorial*. <https://www.mongodb.com/try/download/community>. 2024. (Visitado 15-07-2024).
- [16] Notion Labs Inc. *Notion*. Productivity and organization platform. 2024. URL: <https://www.notion.so/>.
- [17] JetBrains. *PyCharm: Python IDE for Professional Developers by JetBrains*. <https://www.jetbrains.com/es-es/pycharm/>. 2024. (Visitado 15-07-2024).
- [18] JGraph. *diagrams.net, draw.io*. Oct. de 2021. URL: <https://www.diagrams.net/>.
- [19] Kelsey Jordahl et al. *geopandas/geopandas: v0.8.1*. Ver. v0.8.1. Jul. de 2020. DOI: [10.5281/zenodo.3946761](https://doi.org/10.5281/zenodo.3946761). URL: <https://doi.org/10.5281/zenodo.3946761>.
- [20] Jackie Kazil, David Masad y Andrew Crooks. «Utilizing Python for Agent-Based Modeling: The Mesa Framework». En: *Social, Cultural, and Behavioral Modeling*. Ed. por Robert Thomson et al. Cham: Springer International Publishing, 2020, págs. 308-317. ISBN: 978-3-030-61255-9.
- [21] MongoDB, Inc. *MongoDB: The Document Database*. Version 7.0.11. URL: <https://www.mongodb.com/docs/manual/>.

- [22] MongoDB, Inc. *PyMongo: The Python MongoDB Driver*. Version 4.7.3. URL: <https://pymongo.readthedocs.io/en/4.7.3/>.
- [23] Marco Pagani et al. «Demographic scenarios for the new urban plan for Sarajevo Canton: an agent-based digital twin». En: *Conference on Urban Planning and Regional Development Proceedings*. Association of Consulting Engineers Bosnia y Herzegovina. 2023, págs. 30-40.
- [24] Plotly Technologies Inc. *Dash Framework*. Version 2.17.0. URL: <https://dash.plotly.com/>.
- [25] Python Package Index. *GeoJSON*. Version 3.1.0. 2023. URL: <https://pypi.org/project/geojson/>.
- [26] Python Software Foundation. *Python Programming Language*. Version 3.11. URL: <https://www.python.org/>.
- [27] Reitz, Kenneth and Benfield, Cory and Cordasco, Ian. *Requests: HTTP for Humans*. Version 2.x. 2023. URL: <https://docs.python-requests.org/en/latest/>.
- [28] TIOBE Software. *TIOBE Index*. Accedido en Agosto 2024. 2024. URL: <https://www.tiobe.com/tiobe-index/>.
- [29] Linus Torvalds et al. *Git*. Ver. 2.44.0. Distributed version control system. 2024. URL: <https://git-scm.com/>.

Apéndice A

Manual de

Instalación

A.1. Prerrequisitos

Antes de comenzar con la ejecución de las aplicaciones, es necesario tener instalado el siguiente software en el dispositivo:

- **Python:** Será necesario tener instalado Python en el dispositivo, concretamente se recomienda la versión 3.11 para asegurar el correcto funcionamiento de las aplicaciones.
- **MongoDB Community Server:** También necesario alojar localmente la base de datos de MongoDB para que funcione correctamente, siguiendo la guía oficial de MongoDB [15].
- **Cuenta en TomTom Developer:** Si se quiere extraer información será necesario tener una clave de acceso para hacer las peticiones al proveedor, que conseguiremos con la creación de una cuenta gratuita.
- **Cuenta en MapBox:** Para la correcta visualización de los mapas en las interfaces de usuarios necesitaremos una clave de acceso para conseguir los estilos.

Además de tener estas instalaciones se recomienda utilizar un *entorno de desarrollo* óptimo para Python. El más recomendado es **PyCharm** [17], por lo bien conectado que se encuentra al programa que se esté ejecutando y la comodidad que presenta a la hora de usar *entornos virtuales Python* (altamente recomendado para este proyecto) y la gestión e instalación de las librerías Python necesarias.

Cada aplicación del proyecto contiene un fichero «*requirements.txt*». Este fichero incluye todas las **librerías** necesarias para la ejecución del proyecto y sus versiones. El fichero suele ser reconocido instantáneamente por casi cualquier *IDE*. De no ser así, bastaría con ejecutar el

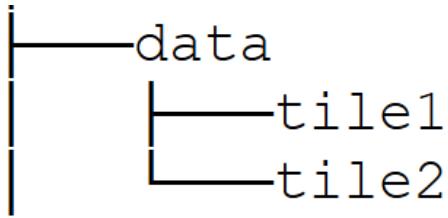


Figura 64: Carpetas donde se almacenarán los datos en bruto.

siguiente comando (con precaución de instalarlo en el entorno virtual si es que se está usando) en la consola: «`pip install -r /path/to/requirements.txt`».

A.2. Aplicación para obtener datos

Después de instalar las librerías necesarias localizadas en «`requirements.txt`», necesitamos **instalar** manualmente el software «`GDAL-3.4.3-cp311-cp311-win_amd64.whl`». Se encuentra incluido en el proyecto, y lo instalaremos usando el comando «`pip install dir/to/GDAL-3.4.3-cp311-cp311-win_amd64.whl`».

Además será necesario añadir una **variable de entorno** para que funcione adecuadamente. Se hará con un fichero en la dirección base del proyecto con el nombre «`.env`»; deberá incluir la clave de usuario de *TomTom* para hacer peticiones en la variable «`TOMTOM_API_KEY`». De esta forma los archivos del proyecto quedarán de la misma manera que en la fig. 65.

La información se almacenará en las carpetas indicadas en la fig. 64, que deberán haber sido creadas previamente.

Con todo esto podremos ejecutar el fichero «`get_tile_info.py`» a partir del *IDE*, o desde consola con el comando «`python get_tile_info.py`».

A.3. Aplicación para procesar los datos

Una vez instaladas las librerías necesarias localizadas en el fichero «`requirements.txt`» será necesario añadir la variable de entorno «`CONNECTION_STRING`» con el enlace a la base de datos de *MongoDB*. Si se ejecuta localmente, probablemente sea «`mongodb://localhost:27017/`».

Deberá añadirse una carpeta con los datos en bruto, de la misma manera que se encontraban a la hora de extraerlos y como puede observarse en la fig. 64; de ahí el programa recibirá los datos para traducirlos. Al igual que nos ocurría antes, deberemos crear una serie de carpe-

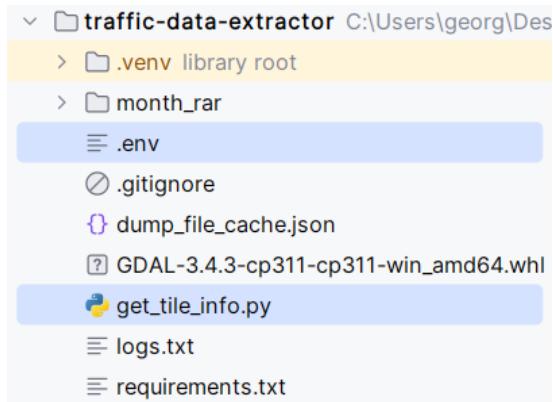


Figura 65: Captura del proyecto con los ficheros importantes destacados.

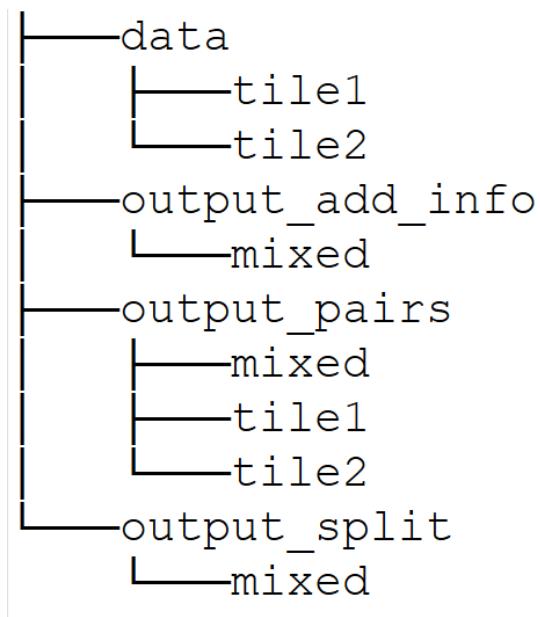


Figura 66: Carpetas que deberán crearse en el proyecto.

tas para asegurar el correcto funcionamiento del programa. Las carpetas creadas por nosotros deberán ser las que tenemos en la fig. 66. Los ficheros quedarán organizados como en la fig. 67.

Finalmente podremos ejecutar el fichero «main.py» a partir del *IDE*, o desde consola con el comando «python get_tile_info.py».

A.4. Inicializar la base de datos

Si se quiere probar alguna de las funciones de las dos aplicaciones diseñadas para el usuario, será necesario añadir datos reales. Para ello se proporcionan dos archivos *JSON* para las dos colecciones que son necesarias en MongoDB (fig. 68).

Deberá crearse una nueva base de datos llamada «TFG». Dentro de ella se crearán dos colecciones: «dates» y «graphs». Posteriormente, habrá que importar en cada una de ellas los ficheros correspondientes. Esto lo podremos hacer utilizando *MongoDB Compass* que se instalará junto a *MongoDB Community Server*.

Para mejorar la eficiencia de las consultas se recomienda crear los índices de la fig. 69, lo cual también faremos desde el programa *MongoDB Compass*. Esto mejorará la velocidad de las consultas pesadas drásticamente.

A.5. Aplicación para el análisis de datos

Después de instalar las **librerías** necesarias desde el fichero «requirements.txt» solo quedaría añadir las **variables de entorno** al directorio base del proyecto con el nombre «.env». La aplicación necesitará las variables «MAPBOX_PUBLIC_TOKEN» y «MONGO_URI», que tendrán la clave pública de *MapBox* con la que obtendremos los estilos de las *tiles* del mapa y la *URI* de la base de datos de *Mongo*, si se ejecuta localmente, probablemente sea «mongodbs://localhost:27017/». Los ficheros estarán dispuestos como en la fig. 70.

Con todo esto listo, solo tendríamos que **ejecutar** el fichero «app.py» ubicado en el directorio base del proyecto, bien con el *IDE* o con la consola de comandos: «python app.py»

A.6. Aplicación para realizar simulaciones

Siguiendo las mismas instrucciones que hasta ahora, se deben instalar las **librerías** ubicadas en el fichero «requirements.txt». Posteriormente se deben añadir de nuevo las variables

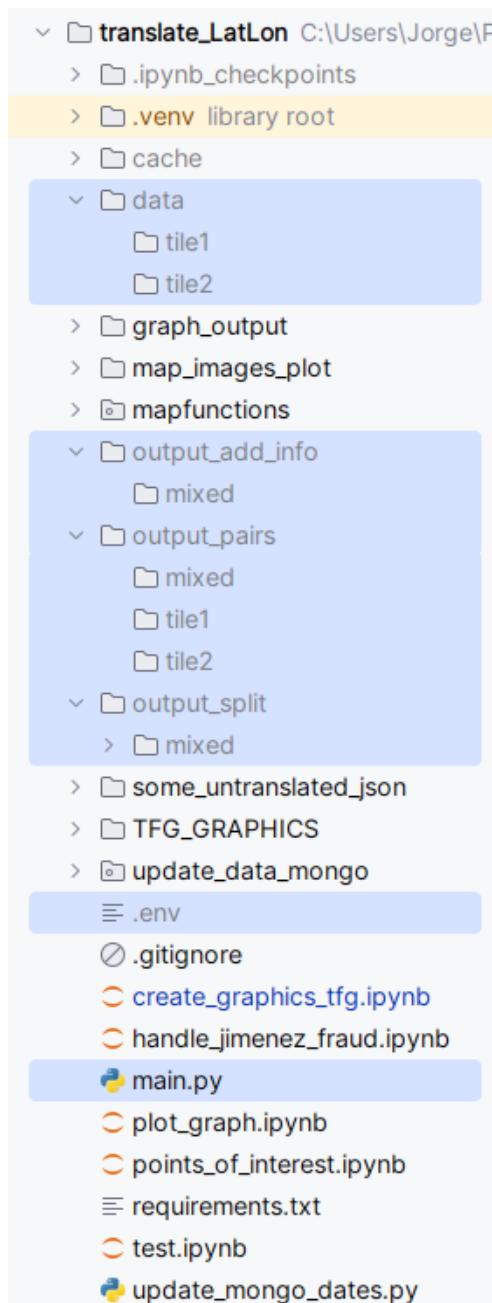


Figura 67: Captura del proyecto con los ficheros importantes destacados.



Figura 68: Captura desde *MongoDB Compass* con la base de datos y las colecciones.



Figura 69: Captura desde *MongoDB Compass* con la base de datos y las colecciones.

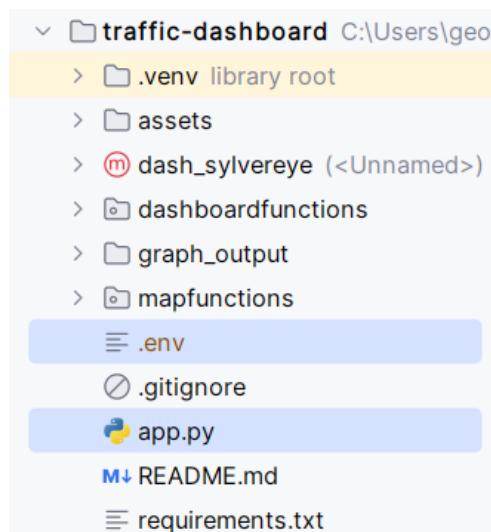


Figura 70: Captura del proyecto con los ficheros importantes destacados.

de entorno, ahora con la diferencia de que necesitaremos dos ficheros «.env». Esto se debe a que el simulador es un módulo aislado de la interfaz, y para asegurar su funcionamiento fuera de esta, es necesario tenerlo. Tendremos ficheros en los directorios:

- «traffic_model»: que solamente contendrá la variable «MONGO_URI».
- Directorio principal: que tendrá de nuevo la variable «MONGO_URI» y «MAPBOX_PUBLIC_TOKEN», de la misma forma que lo hacía la aplicación para el análisis de datos.

Por lo tanto, todo el conjunto de archivos de este proyecto quedarán dispuestos de la misma manera que en la fig. 71.

Finalmente, solo tendríamos que **ejecutar** el fichero «app.py» ubicado en el directorio «dash_gui», bien con el *IDE* o con la consola de comandos: «python app.py».

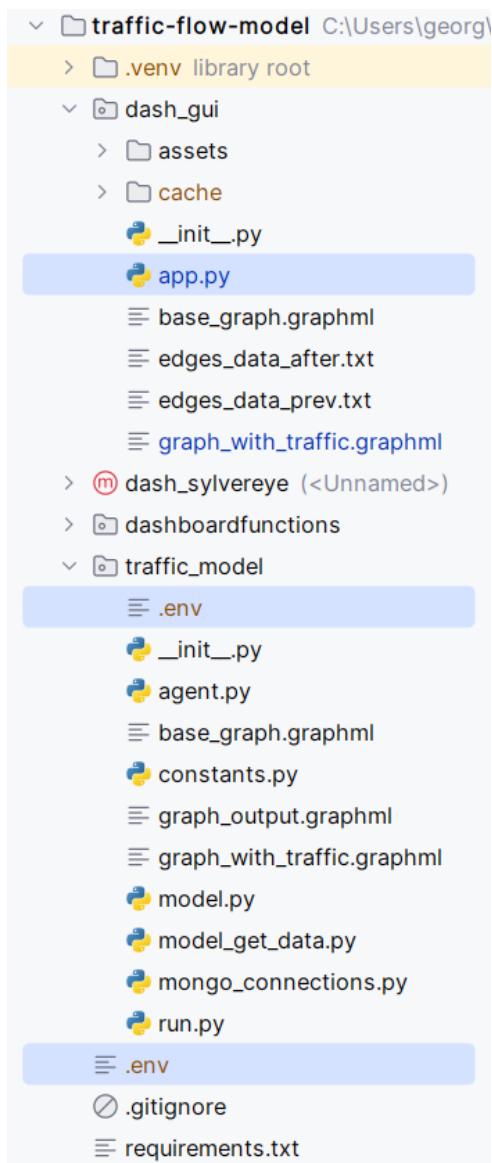


Figura 71: Captura del proyecto con los ficheros importantes destacados.

Apéndice B

Manual de Usuario

Este manual está diseñado para guiar al usuario a través del uso y funcionalidades de las aplicaciones de análisis de datos y simulación de tráfico urbano. Estas herramientas han sido desarrolladas para ofrecer una experiencia intuitiva y eficiente. Este documento proporcionará todas las instrucciones necesarias para aprovechar al máximo las capacidades de las aplicaciones.

B.1. Extractor y procesador de datos

Si se han seguido todos los pasos del manual de instalación (incluido la creación de carpetas necesarias), lo único que hay que hacer es ejecutar los ficheros indicados, que son «get_tile_info.py» y «main.py» respectivamente. No requiere de más instrucciones para su uso.

B.2. Aplicación para el análisis de datos

B.2.1. Editar estilo del mapa

En la zona central de la interfaz podemos ubicar el mapa, centrado en el grafo de Teatinos. A la derecha del mismo podemos ver tres paneles, los dos inferiores son los que están completamente enfocados en el estilo del mapa, para los nodos y aristas respectivamente. Se pueden modificar libremente todos los atributos, deslizables de tamaño u opacidad, el desplegable con la política para colorear, etc.

La última opción de personalización («Display») no hará nada aunque se modifique hasta que se haya cargado el histórico de los flujos de tráfico de una fecha pasada, pudiendo controlar la visualización de datos normalizados o reales.

En la fig. 72 podemos observar un ejemplo de uso sencillo, en el que se modifica el tamaño y transparencia de nodos y aristas, y además se cambia la política de coloreado en función de la velocidad máxima de la vía.

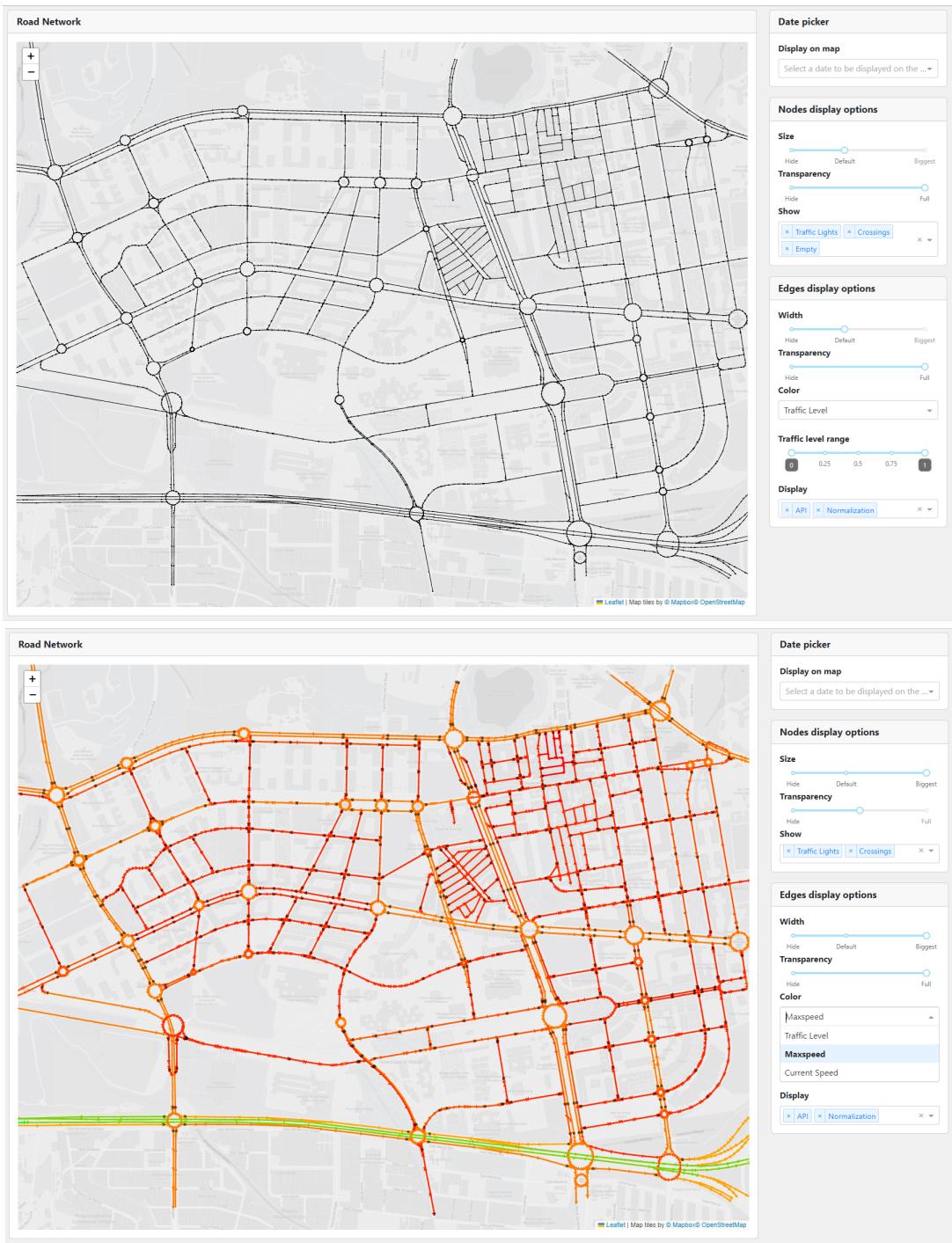


Figura 72: Capturas de pantalla con el antes y el después de cambiar el estilo del mapa

Node Information	
Property	Value
Latitude	N/A
Longitude	N/A
Street Count	N/A
Crossing	N/A
Traffic Light	N/A
OSM ID	N/A

Edge Information	
Property	Value
Name	N/A
Maxspeed	N/A
Highway	N/A
Traffic level	N/A
Current Speed	N/A
Lanes	N/A
OSM ID	N/A

Node Information	
Property	Value
Latitude	-4.4762669
Longitude	36.7174842
Street Count	2
Crossing	Yes
Traffic Light	No
OSM ID	5354846229

Edge Information	
Property	Value
Name	Calle Jiménez Fraud
Maxspeed	30
Highway	secondary
Traffic level	0.482
Current Speed	14.46
Lanes	2
OSM ID	284697342

Figura 73: Columna antes y después de pulsar sobre un nodo y una arista.

B.2.2. Ver información adicional

Para ver en detalle la información de algún nodo o arista el usuario solo deberá pulsar sobre el elemento que desea consultar la información, y las dos columnas actualizarán la información mostrada de manera automática.

Junto a los datos más destacados de los elementos, en las aristas, podemos observar cómo se dibuja una representación de la vía en cuestión. Esto será de especial utilidad para permitir la visualización de los niveles de tráfico correctamente para aquellas vías de doble sentido, que no tienen por qué tener el mismo flujo de tráfico (fig. 73).

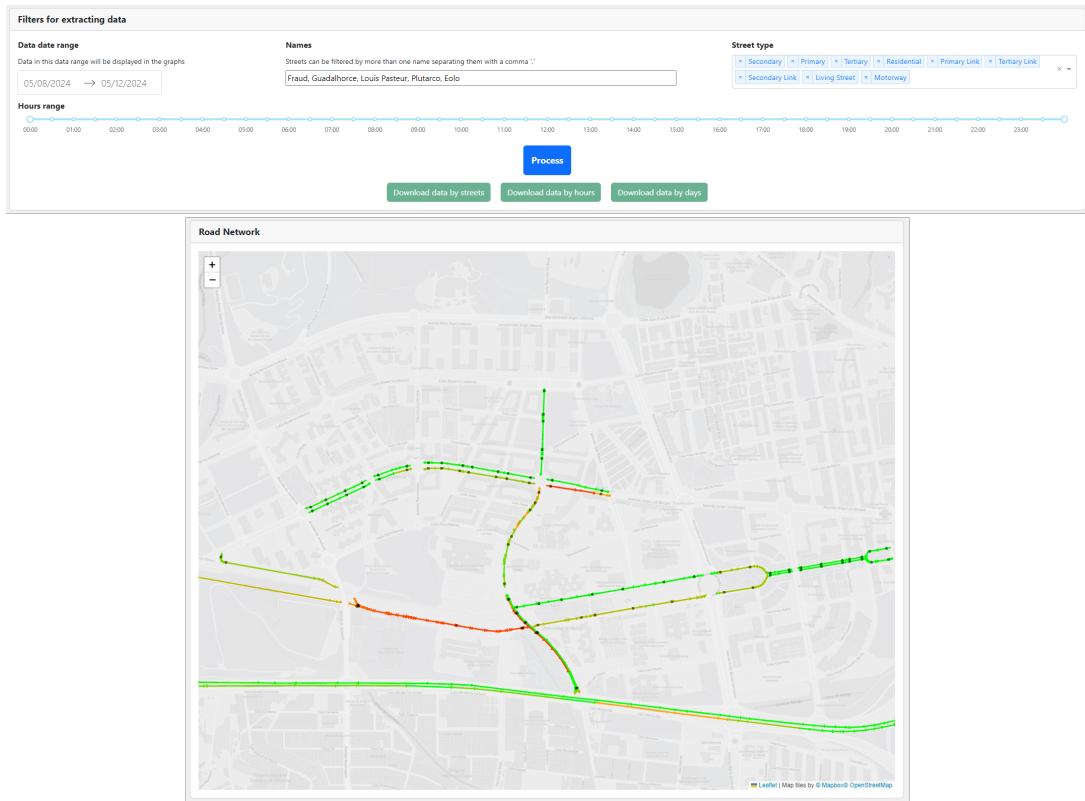


Figura 74: Captura con los filtros de las aristas y el resultado en el mapa.

B.2.3. Filtrar las vías del grafo

Si el usuario quiere filtrar las vías del grafo, bien para la visualización en el mapa o para un procesamiento de los datos más preciso, se debe usar el primer tablero en la zona superior de la interfaz de usuario. Los filtros reflejados en el mapa son el **nombre de las vías**, que podrán incluirse varios siempre y cuando estén separados por comas y el **tipo de vías** (fig. 74).

Es importante recalcar que los filtros son mutuamente excluyentes entre ellos, es decir, si el usuario filtra por nombre «Autowía del Guadalhorce», pero en el seleccionable de la derecha no se añaden las vías de tipo «Motorway», no se incluirán en el grafo.

B.2.4. Realizar análisis de los datos

Para realizar el análisis automático del histórico de datos se usará el mismo panel que para el filtrado. De hecho, el filtrado producido en el mapa, es el mismo que se hará en el análisis del histórico de datos. Por lo que puede servir al usuario como guía de las calles cuyo tráfico está analizando.

Figura 75: Captura antes y después de comenzar el análisis de los datos.

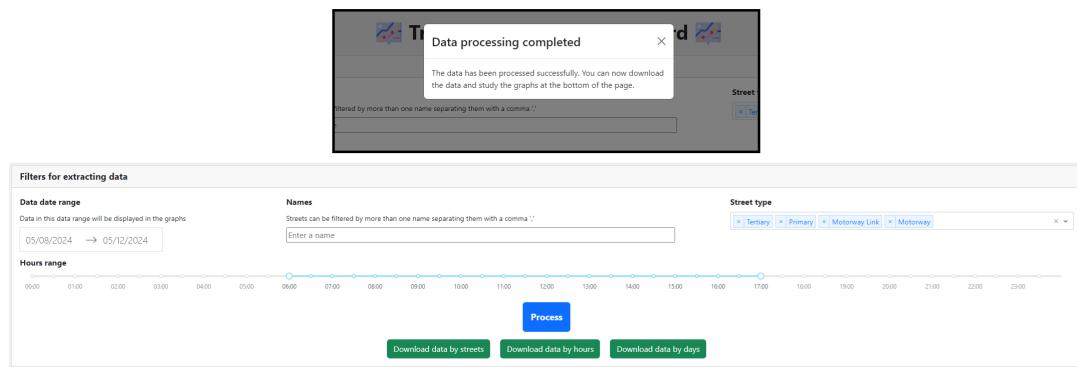


Figura 76: Captura de la interfaz una vez terminado el análisis.

Primero tendrá que seleccionar en el calendario el rango de fecha a analizar. Por defecto será el primer y último día de los que se tienen registro del tráfico. Después podrá filtrar el tipo de carreteras y nombre de las calles si lo desea (si deja las opciones en blanco se descartará el filtro) y finalmente, podrá descartar un rango de horas. Por ejemplo, si solo interesa estudiar las condiciones del tráfico a mediodía, por la noche o de madrugada.

Una vez el usuario esté satisfecho con el filtrado de las vías, puede pulsar el botón «Process» para comenzar con el procesamiento de los datos, y aparecerá un «spinner» giratorio indicando que está cargando (fig. 75).

Cuando el procesamiento termine, el usuario verá un mensaje notificando que está listo, y además se habilitarán los botones de descarga de los resultados obtenidos (fig. 76).

Como se le indica al usuario en la ventana flotante, las gráficas se encuentran disponibles en cuanto termine el análisis, y el usuario podrá acceder a ellas desplazándose hasta el último

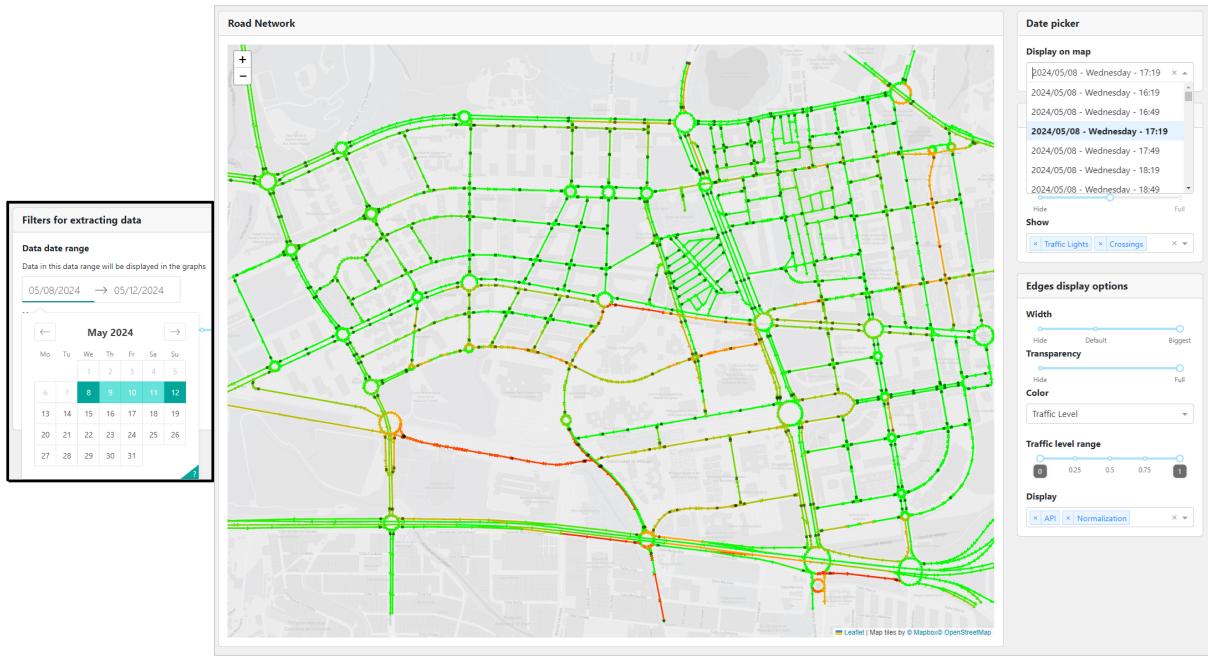


Figura 77: Captura seleccionando un rango de fecha y una hora del histórico de tráfico.

tablero inferior de la interfaz.

B.2.5. Cargar tráfico de una fecha en el mapa

Para cargar en el mapa el histórico con el nivel de tráfico de una fecha en concreto será necesario seleccionar de la zona de procesamiento de datos el rango de días en el que se encuentra el histórico deseado, y posteriormente, en el primer tablero disponible a la derecha del mapa, se podrá seleccionar la hora y el día que se quiere cargar en el mapa (fig. 77).

Una vez que el usuario pulse sobre un valor de histórico del desplegable, el mapa se actualizará automáticamente con los valores de tráfico de esa fecha. El estilo e incluso la política de coloreado podrán ser modificados sin problemas: el sistema recordará la fecha seleccionada para volver a colorearlo posteriormente.

B.2.6. Gráficas

Una vez que el análisis ha finalizado, las gráficas están disponibles para el usuario. Serán tres las gráficas que podrá visualizar el usuario, que representan un análisis en función de diferentes aspectos, concretamente en función de las horas, de la calle, y del día de la semana (fig. 78).



Figura 78: Captura de las gráficas una vez terminado el análisis.



Figura 79: Captura del desplegable de configuración de las gráficas.

El usuario dispone de diferentes configuraciones para las gráficas mostradas, todas ellas se configuran a partir del mismo desplegable. Podrá visualizar las gráficas en función de los niveles de tráfico, o la velocidad real de la vía. Además podrá visualizar la media de dichos valores, también podrá visualizar la mediana, y los valores mínimo y máximo (fig. 79).

Finalmente, las gráficas permiten hacer zoom y deshacerlo de manera dinámica arrastrando sobre ellas y pulsando en los botones de la esquina superior derecha de cada una. Entre estos botones se encuentra el de hacer una captura, que generará un archivo en formato *PNG* disponible para que el usuario lo guarde (fig. 80).



Figura 80: Captura con las funciones de las gráficas.

B.2.7. Interfaz completa

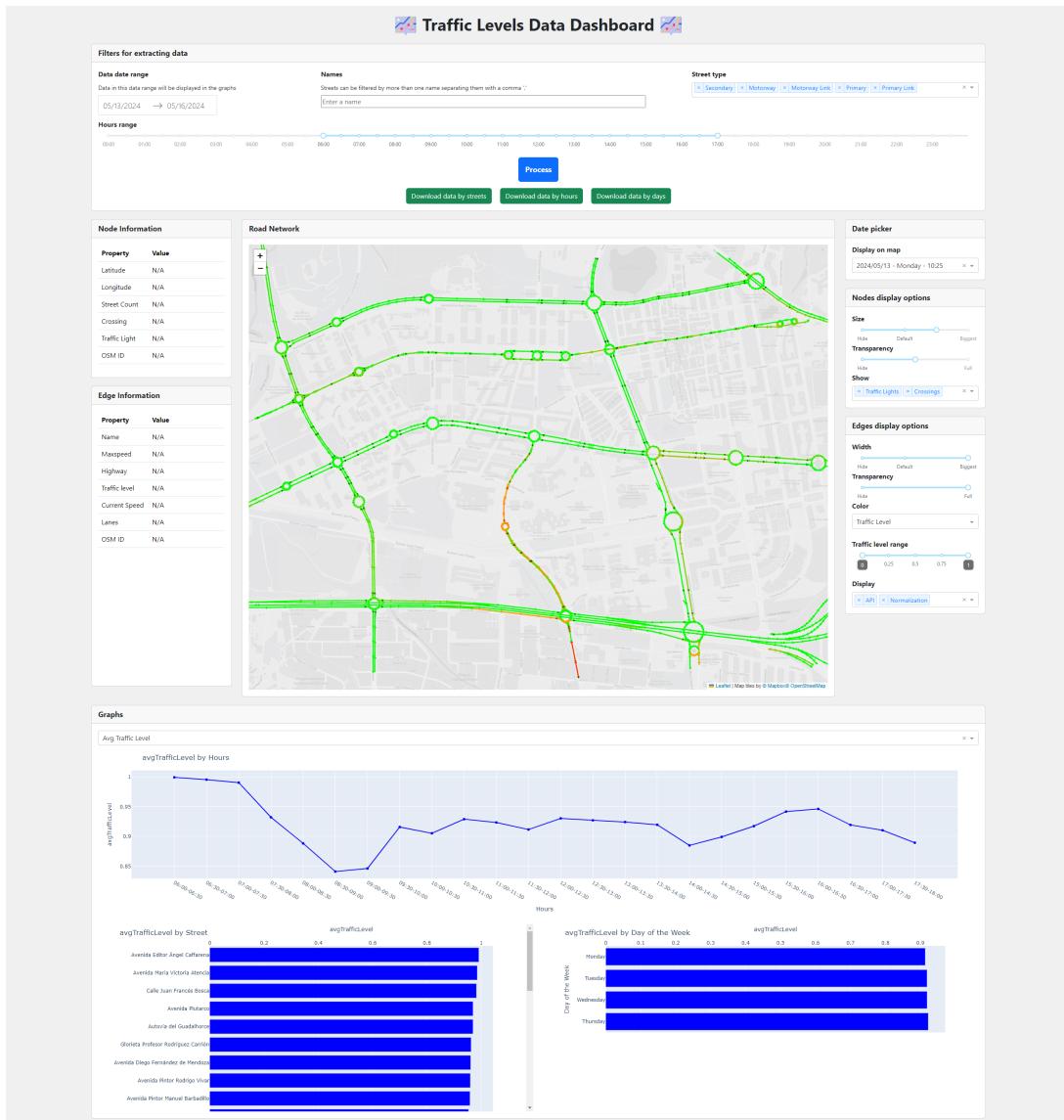


Figura 81: Captura de la interfaz completa

B.3. Aplicación para realizar simulaciones

Esta aplicación al estar basada en la anterior, comparte gran parte de las funcionalidades, tales como el cambio de estilo del mapa, la información adicional de los elementos. Por lo tanto, solo se explicarán en el manual aquellas funcionalidades que difieran en algo de la interfaz previa.

B.3.1. Filtrar las vías del grafo

El filtrado de aristas sigue igual, con la diferencia de que está desplazado con respecto a la otra aplicación, ahora se encuentra en la parte inferior de la columna de la derecha, pero el usuario podrá filtrar con los nombres y el tipo de vía de la misma manera (fig. 82).

Este filtro no afecta de ninguna manera al simulador, que funcionará en todo momento con el grafo completo.

B.3.2. Gráficas

Las gráficas contienen las mismas funciones dinámicas que en la anterior interfaz, permitiendo de la misma manera hacer capturas o zoom. Aunque ahora no tenemos disponible una opción para cambiar de variable a analizar, las gráficas son el resultado de las simulaciones, y conforme se vayan realizando simulaciones se irán agregando a las gráficas (fig. 83).

También se generará una tabla que recoge los valores más importantes de la última simulación que se realice.

B.3.3. Realizar simulaciones

Para realizar las simulaciones, el usuario seguirá un procedimiento similar al de la aplicación para el análisis de datos, ya que primero deberá completar todos los campos disponibles con la configuración deseada para la simulación. Estos campos recogerán:

- La cantidad de agentes que se desplegarán en el mapa.
- Cuántos pasos durará la simulación.
- Si los agentes que lleguen a su destino reaparecerán en otro punto o no.

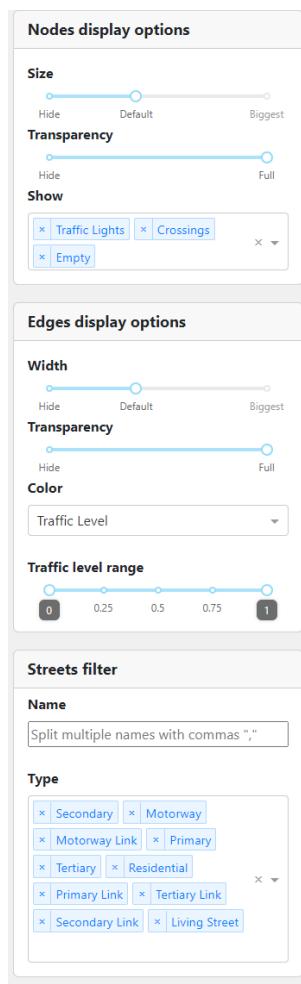


Figura 82: Captura con la columna derecha de la interfaz del simulador.

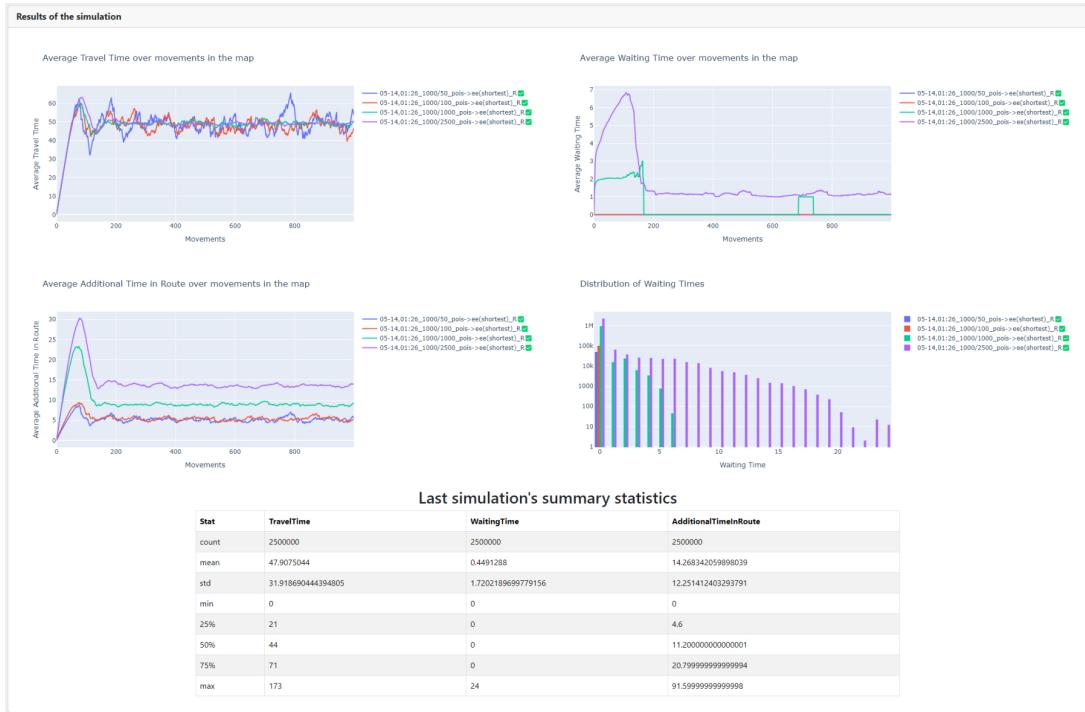


Figura 83: Captura con las gráficas después de ejecutar cuatro simulaciones.

- La política para seleccionar el nodo inicial y final de cada agente (pudiendo ser aleatorio, puntos de interés o de entrada/salida al grafo).
- La política de enrutamiento de los agentes (pudiendo ser aleatoria, predefinida al empezar teniendo en cuenta o no los pesos y calculando el siguiente movimiento con Dijkstra).
- La fecha y hora en la que la simulación comenzará.

Todos los campos deberán ser completados antes de que el usuario pulse el botón «Start simulation». De no ser así, el usuario vería una notificación indicándole donde se encuentra el error (fig. 84). La simulación conforme vaya ejecutándose irá completando una barra de carga que permitirá al usuario saber aproximadamente cuánto le queda. Cuando la simulación termine podrá descargarse los datos de la simulación realizada de la misma manera que se hacía con la aplicación anterior.

Adicionalmente, en esta tarjeta encontramos un desplegable donde podremos seleccionar si el nivel de tráfico mostrado en el mapa es el resultado de la simulación o de la fecha seleccionada. Este desplegable solo estará disponible una vez que se realice alguna simulación. Hasta ese momento el nivel de tráfico mostrado siempre será el de la fecha seleccionada (fig. 85).

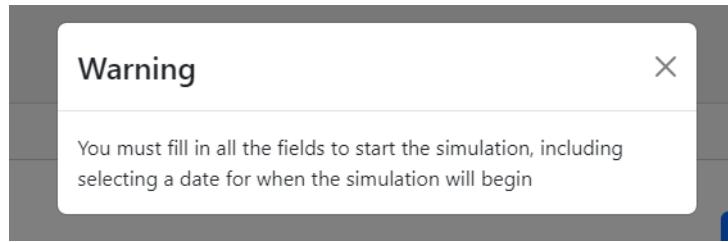


Figura 84: Captura de la notificación de aviso por campos vacíos

Display graph from:

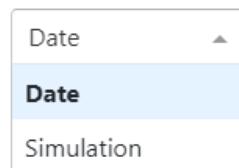


Figura 85: Captura del desplegable de la interfaz

B.3.4. Cargar tráfico de una fecha en el mapa

El usuario que quiera mostrar el nivel de tráfico de una fecha pasada en el mapa deberá proceder de una manera similar a la de la aplicación anterior, pero en la sección dedicada a realizar las simulaciones. Deberá seleccionar un rango de fecha en el calendario, del momento que desea mostrar el nivel de tráfico, y en el desplegable que está debajo seleccionar la hora (fig. 86).

En caso de que se haya realizado alguna simulación, deberá seleccionar específicamente que quiere mostrar el tráfico de la fecha seleccionada en el desplegable de la fig. 85.

B.3.5. Interfaz completa

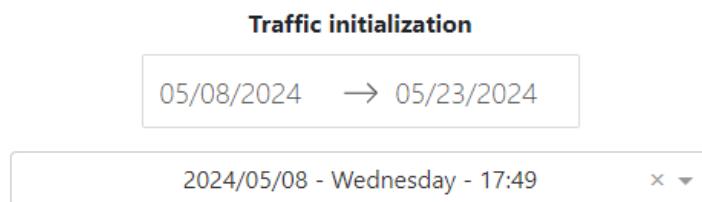


Figura 86: Captura del calendario y el desplegable de la interfaz

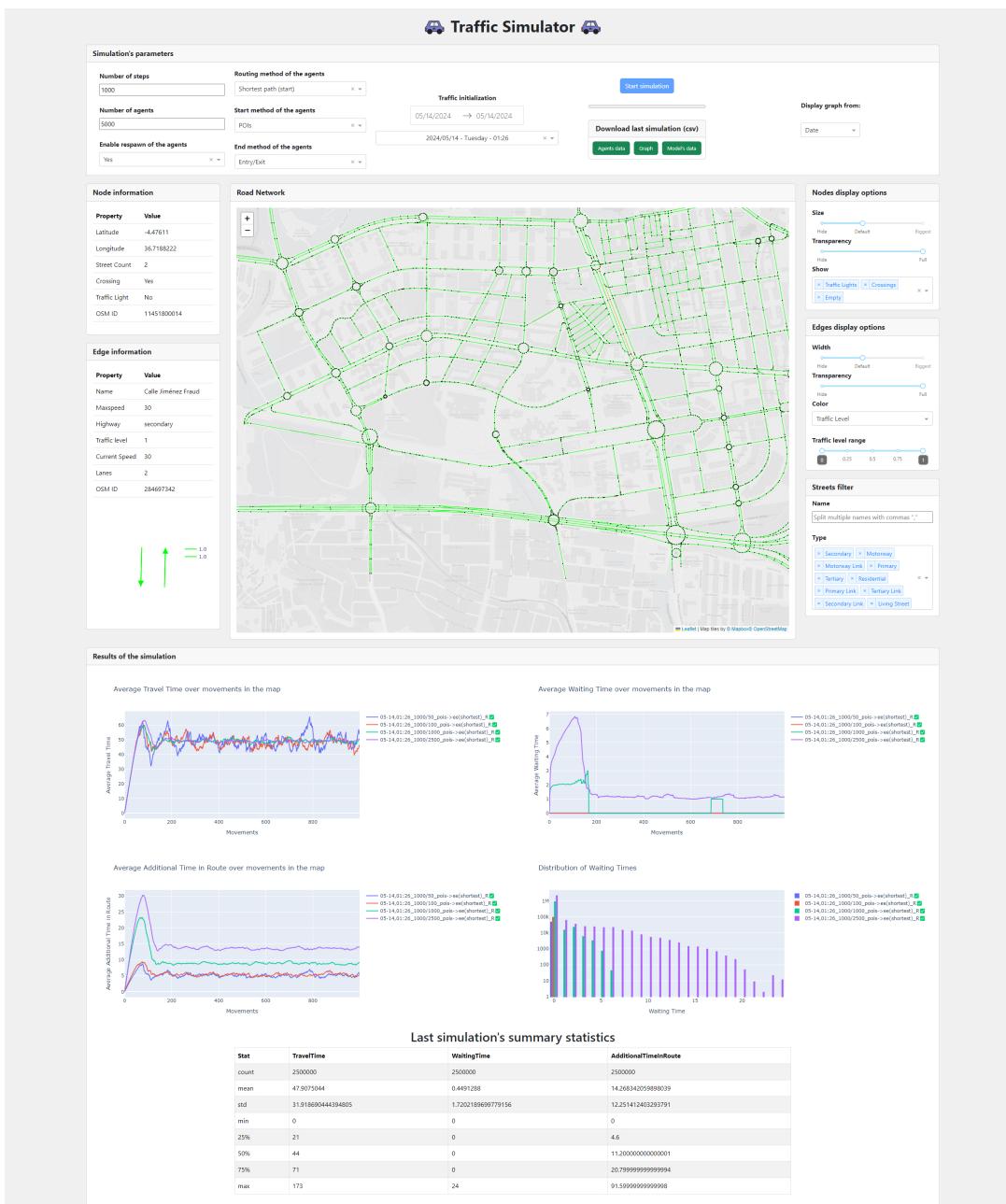


Figura 87: Captura de la interfaz completa

Índice de figuras

1.	Idea conceptual ideal del ciclo de vida del producto por Grieves [9].	11
2.	Concepto de gemelo digital de Grieves y Vickers [11].	12
3.	Diagrama con las fases e iteraciones del desarrollo.	20
4.	Diagrama general del proyecto.	21
5.	Medida del área utilizando la plataforma <i>FreeMapTools</i>	50
6.	Flujo de tráfico en una zona de Teatinos en formato PNG.	52
7.	Comparación de los formatos del nivel de tráfico de Raster Flow Tiling en la zona del Hospital Clínico	53
8.	<i>Tiles</i> necesarios en mapa de <i>MapTiler</i>	53
9.	Diagrama de secuencia para la extracción de información.	56
10.	Gráfica con el proceso de extracción de la información del proveedor.	57
11.	Ejemplo con dos elementos de un archivo geoJSON de <i>TomTom</i>	58
12.	Cuadrícula con las coordenadas de cada <i>tiles</i> solicitada.	60
13.	Imagen del mapa del <i>tile</i> norte de Teatinos con las longitudes y latitudes mínimas y máximas.	64
14.	Imagen del mapa del <i>tile</i> sur de Teatinos con las longitudes y latitudes mínimas y máximas.	64
15.	Imagen del grafo generado.	65
16.	Imagen del mapa con un GeoJSON postprocesado cargado.	67
17.	Comparación la información en los grafos en función de la medida de los elementos	68
18.	Representación del proceso que se debe realizar en las carreteras de doble sentido.	69
19.	Gráfica con el proceso de traducción y mezcla de archivos.	69
20.	Representación del grafo con la arista más cercana a un elemento del GeoJSON	70
21.	Imagen del grafo ampliado en Calle Jiménez Fraud	71
22.	Diccionario que se añadirá a cada una de las aristas del grafo	75
23.	Captura de MongoDBCompass con la estructura de cada documento	76
24.	Gráfica con el proceso de exportación de los datos al grafo y MongoDB.	77

25.	Gráfica con el proceso de extracción de la información.	78
26.	Diagrama general del proceso de extracción de la información.	79
27.	Imagen del grafo generada con la librería <i>OSMNx</i>	81
28.	Información disponible de una arista en el grafo base.	83
29.	Gráfica con el nivel de tráfico medio en el rango de 24 horas del día.	84
30.	Gráfica con los niveles mínimos de tráfico en las calles.	84
31.	Gráfica con la velocidad media en los días de la semana.	85
32.	Sección de código donde se genera el identificador por la franja de tiempo. . .	86
33.	Diagrama con las etapas de las agregaciones en MongoDB.	87
34.	Gradiente de color para las aristas del mapa.	88
35.	Gradiente de color para las aristas del mapa.	89
36.	Proceso que sufre la interfaz mientras se procesan los datos	90
37.	Columna antes y después de pulsar sobre un nodo y una arista	91
38.	Columna derecha de la interfaz.	93
39.	Captura de pantalla de la tarjeta con los grafos (vacía)	94
40.	Captura de pantalla de la tarjeta con los grafos	94
41.	Captura de pantalla del desplegable de la tarjeta con los grafos	95
42.	Diagrama general del <i>dashboard</i>	96
43.	Diagrama del entorno del modelo.	101
44.	Diagrama de los agentes del modelo.	104
45.	Diagrama del modelo.	106
46.	Captura del panel dedicado para la simulación.	108
47.	Captura del panel dedicado para la simulación relleno.	108
48.	Captura de la alerta mostrada.	109
49.	Captura de las gráficas con dos simulaciones.	110
50.	Captura de la tabla tras realizar una simulación.	110
51.	Diagrama general del modelo y su <i>dashboard</i>	111
52.	Diagrama de <i>Grafana</i> con los tipos de pruebas de carga [8].	113
53.	Captura de pantalla de <i>MongoDB Compass</i> con las fechas con información disponible	115
54.	Captura de pantalla de <i>MongoDB Compass</i> con los grafos informados	115

55.	Captura de pantalla de <i>MongoDB Compass</i> con el rango de fechas disponible	115
56.	Captura de pantalla de <i>MongoDB Compass</i> con el recuento de grafos con información única	116
57.	Gráfica con los tiempos de carga	119
58.	Gráfica con los tiempos de carga	122
59.	Gráfica con los tiempos de carga	123
60.	Tabla de valores destacados de la simulación con ruta predefinida	124
61.	Tabla de valores destacados de la simulación con ruta calculada en cada paso .	124
62.	Gráfica con la media de tiempo adicional en las rutas de los agentes en la simulación con rutas predefinidas	125
63.	Gráfica con la media de tiempo adicional en las rutas de los agentes en la simulación con rutas calculadas en cada paso	125
64.	Carpetas donde se almacenarán los datos en bruto.	136
65.	Captura del proyecto con los ficheros importantes destacados.	137
66.	Carpetas que deberán crearse en el proyecto.	137
67.	Captura del proyecto con los ficheros importantes destacados.	139
68.	Captura desde <i>MongoDB Compass</i> con la base de datos y las colecciones.	140
69.	Captura desde <i>MongoDB Compass</i> con la base de datos y las colecciones.	140
70.	Captura del proyecto con los ficheros importantes destacados.	140
71.	Captura del proyecto con los ficheros importantes destacados.	142
72.	Capturas de pantalla con el antes y el después de cambiar el estilo del mapa .	144
73.	Columna antes y después de pulsar sobre un nodo y una arista.	145
74.	Captura con los filtros de las aristas y el resultado en el mapa.	146
75.	Captura antes y después de comenzar el análisis de los datos.	147
76.	Captura de la interfaz una vez terminado el análisis.	147
77.	Captura seleccionando un rango de fecha y una hora del histórico de tráfico. .	148
78.	Captura de las gráficas una vez terminado el análisis.	149
79.	Captura del desplegable de configuración de las gráficas.	149
80.	Captura con las funciones de las gráficas.	149
81.	Captura de la interfaz completa	150
82.	Captura con la columna derecha de la interfaz del simulador.	152

83.	Captura con las gráficas después de ejecutar cuatro simulaciones.	153
84.	Captura de la notificación de aviso por campos vacíos	154
85.	Captura del desplegable de la interfaz	154
86.	Captura del calendario y el desplegable de la interfaz	154
87.	Captura de la interfaz completa	155

Índice de cuadros

1.	Requisitos funcionales	29
2.	Requisitos no funcionales	31
3.	Caso de Uso RF01: Mostrar tráfico del pasado	31
4.	Caso de Uso RF02: Información de nodos y aristas	33
5.	Caso de Uso RF03: Cambio de estilo del grafo	35
6.	Caso de Uso RF04: Filtrado de carreteras	37
7.	Casos de Uso RF05 y RF11: Procesado de información	39
8.	Casos de Uso RF06 y RF11 : Realizar simulaciones y Notificación	41
9.	Caso de Uso RF07: Alternar tráfico	44
10.	Caso de Uso RF09: Descargar resultados	46
11.	Caso de Uso RF10: Descargar gráficas	47
12.	Diccionario de datos de las etiquetas utilizadas en la información de tráfico	59
13.	Especificaciones del equipo de prueba	114
14.	Tabla con los tiempos de procesamiento, en segundos, para cada caso	118
15.	Resultados con enrutamiento en el momento de inicialización.	122
16.	Resultados con enrutamiento complejo basado en Dijkstra.	123



UNIVERSIDAD
DE MÁLAGA | **uma.es**

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga