Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN CIENCIA E ENXEÑARÍA DE DATOS

Análisis, diseño e implementación de una aplicación web para la consulta y visualización de los medios de transporte del municipio de Madrid.

Estudante: Brais Klein Otero

Dirección: Diego Seco Naveiras

Tirso Varela Rodeiro

A Coruña, June de 2023.

A mi familia y amigos.

Agradecimientos

Aún tengo muy presente un día de finales de verano de 2019 en el que me acerqué por primera vez a la UDC, para preguntar por estos estudios. La claridad y calidez con la que me recibió Berta Guijarro, generó en mi las ganas de formar parte del mundo de los datos y dedicarme a ello profesionalmente.

Siempre me he encontrado cómodo en el ambiente universitario de la facultad de informática e, incluso, he disfrutado y aprovechado proyectos complementarios, como han sido los cursos homologados de Inglés o mentor de estudiantes, así como actividades deportivas.

Quiero hacer un agradecimiento especial a mis ”colaboradores”, Tirso y Diego. Trabajar con ellos lo hace todo más fácil. Son dos grandes orientadores que me han permitido presentar este trabajo fin de grado de la mejor manera posible. Muchos aciertos se deben a ellos, los errores son solo míos.

También quiero agradecer a mis cómplices en este viaje.

Primeramente a mi madre Victoria Eugenia, origen, norte e inspiración, de cuya fuente he bebido su constancia. A mi padre Julio César por su pragmatismo y buenos consejos basados en la experiencia. A mi madrina Carmen, mi psicóloga particular. Le agradezco ser tan didáctica y estar siempre a mi lado. También a Elías, por la paciencia y por compartir la aventura de ser hermano.

Además de mi familia me gustaría hacer una breve mención a ciertas personas, las cuales han hecho de este etapa de mi vida algo inolvidable.

Gracias a Dei, por ser un gran amigo y confidente. Gracias a Lucía, por apoyarme siempre y ayudarme a ser mejor persona. Gracias a Castelo, por demostrarme que aunque pasen los años las buenas amistades prevalecen. Gracias a Román, por sacarme una sonrisa sin importar las circunstancias.

También quiero dar las gracias a Pérez, Temes, Alberto, Sara, Marta, Prieto, Chou, Mateo, Nolito, Canosa, JJ y Josiño. Cada uno de vosotros habéis aportado un granito de arena a este trayecto que hoy finaliza.

Como dicen los 184 años de mis abuelos: De los tiempos, los presentes y siempre, siempre con humildad y perseverancia.

Resumen

En este trabajo fin de grado el objetivo principal consiste creación e implementación de una aplicación web que permita la consulta y visualización de datos acerca del sistema de transporte público proporcionado por el municipio de Madrid. Se ha perseguido la idea de que la herramienta sea usada principalmente por parte de los trabajadores y directivos del Consorcio de Transportes de Madrid con el fin de lograr la mejora de la gestión del sistema, además de facilitar información del transporte público a los viajeros.

Con estas premisas, inicialmente se han llevado a cabo dos análisis, uno preliminar y otro exploratorio. Esto se hizo con el fin de conocer el conjunto de datos sobre los que se operaría. A continuación, se definieron los requisitos funcionales así como el alcance del proyecto. Posteriormente, se elaboraron una serie de bocetos de posibles interfaces de usuario que podría tener la aplicación, además de realizar el modelado conceptual definitivo de los que se usarían en el despliegue de la misma. Finalmente, se procedió al desarrollo e implementación de la plataforma.

En cuanto a las tecnologías empleadas durante el avance del proyecto, se utilizó PostgreSQL para almacenar el conjunto de datos. Por otra parte, se hizo uso del framework Django para definir los modelos de datos e implementar la aplicación web. A su vez, Django se combinó con códigos de JavaScript para dotar de mayor iteración a la vistas finales del proyecto. Por último, se emplearon tecnologías como HTML, CSS, Leaflet, Boostrap y PowerBI para la visualización del contenido.

Este trabajo fin de grado se realizó siguiendo una metodología iterativa incremental.

Abstract

In this end-of-degree project, the main objective is the creation and implementation of a web application that allows the consultation and visualization of data regarding the public transportation system provided by the Municipality of Madrid. The idea has been to make the tool primarily used by the employees and managers of the Madrid Transport Consortium in order to improve the management of the system and provide information to the travelers.

With these premises, two analyses were initially carried out, a preliminary one and an exploratory one. This was done to understand the dataset on which the project would operate. Subsequently, the functional requirements and the scope of the project were defined. Then, a series of sketches of possible user interfaces for the application were created, along with the definitive conceptual modeling that would be used in its deployment. Finally, the platform was developed and implemented.

Regarding the technologies used during the project's progress, PostgreSQL was employed to store the dataset. Furthermore, the Django framework was used to define the data models and implement the web application. Django was combined with JavaScript code to provide enhanced interactivity to the final views of the project. Lastly, technologies such as HTML, CSS, Leaflet, Bootstrap, and PowerBI were used for content visualization.

This end-of-degree project was carried out following an iterative incremental methodology.

Palabras clave:

- Transporte público
- Django
- Aplicación web
- Python
- Análisis y visualización de datos
- PostgreSQL
- Mejora del transporte
- Leaflet
- Consorcio de Transporte de Madrid
- PowerBI

Keywords:

- Public transportation
- Django
- Web application
- Python
- Data analysis and visualization
- PostgreSQL
- Transportation improvement
- Leaflet
- Madrid Transport Consortium
- PowerBI

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1
Introducción

En este capítulo se presenta tanto la motivación que subyace bajo la realización de este proyecto como los objetivos que el mismo debe cumplir.

1.1 Motivación

La utilización de tarjetas electrónicas ha supuesto una verdadera revolución en el uso del transporte público. El liberarse de billetes en formato físico hace que se gestione más rápido y eficazmente el transporte. Además, el disponer de tarjeta electrónica contribuye a la sostenibilidad del medio ambiente en la medida que no es necesario utilizar la impresión en papel. Esta notable evolución se puede observar en la Figura 1.1.

Los beneficios del sistema electrónico de billetes no solamente afecta a los usuarios del transporte sino también, y de una manera importante, a las empresas responsables del mismo, pues disponen de gran cantidad de datos para poder hacer análisis y gestión posterior de los usos que hacen los viajeros a todos los niveles. Gracias a esto se podría manejar más adecuadamente e introducir mejoras en toda la red de transporte público. De esta forma se solucionarían situaciones desfavorables en el sistema público de transporte en beneficio de los pasajeros, evitando esperas, masificaciones, o bien, adecuando los servicios en función de la demanda de los usuarios. A día de hoy, esta gestión de los datos está infrautilizada, básicamente por razón de no disponer de las herramientas centradas para su manejo y explotación.

El objetivo principal de este proyecto es trabajar sobre los datos tanto de la red del transporte del municipio de Madrid como del uso de la misma, para generar una aplicación de utilidad a través de la cual se puedan realizar análisis y extraer, de ellos, conclusiones adecuadas para los distintos interesados: clientes, operarios y directivos del organismo responsable del transporte en el municipio.

Para la consecución de dicho objetivo se creará una aplicación web mediante la cual se visualizarán y gestionarán datos de los pasajeros en sus trayectos, así como mapas y elementos

diversos de las rutas de los distintos medios de transporte. También se obtendrán datos sobre las secuencias, horas y jornadas con sus características como son: el total de usuarios y viajes, los lugares de mayor concentración de pasajeros o días de la semana con mayor afluencia, entre otros aspectos. A través de esta aplicación, los responsables del servicio tendrán elementos objetivos y precisos en los cuales apoyarse para la toma de decisiones, a corto y medio plazo, y poder planificar la evolución del servicio con sus necesidades futuras. Adicionalmente, el usuario público tendrá acceso a la herramienta para obtener información de ciertos aspectos de la topología de la red e, incluso, tendrá datos del histórico de sus trayectos.

Figura 1.1: Evolución del billete de transporte público del municipio de Madrid. [1]

1.2 Objetivos

Como objetivo principal del proyecto se establece la creación de una aplicación web que permitirá la gestión de la red de transporte del municipio de Madrid por parte de los trabajadores y directivos del Consorcio de Transporte del municipio. Se desarrollará esta aplicación *ad hoc* para que la puesta en marcha sea lo más adecuada posible y deberá mostrar los datos de viaje de los clientes y la estructura de la red de transporte.

Además, este objetivo central se desglosa en los objetivos secundarios que se presentan a continuación:

- **Realizar un completo análisis exploratorio de los datos recopilados a través de las tarjetas electrónicas de viaje.** En esta fase, los datos sin procesar se limpian, preparan y visualizan antes de examinarlos en busca de patrones y tendencias útiles. Este proceso permite comprender mejor el comportamiento de los usuarios mediante la exploración de datos.

- **Modelar los datos.** Usar técnicas de modelado de datos para transformar los datos recopilados en información detallada sobre los distintos medios de transporte público del municipio de Madrid. Para abordar esta transformación es imprescindible organizar

los datos de manera que se tengan en cuenta los aspectos más relevantes de la red del transporte.

- **Desarrollar una aplicación web que permita a los usuarios realizar consultas sobre los datos**. Es muy importante desplegar una interfaz que resulte ágil, práctica, intuitiva y sobre todo sencilla. En la búsqueda de este objetivo, la aplicación debe contar con una página en la que se refleje, de manera ordenada y detallada, información de los trayectos de los viajeros a través de la topología de la red de transportes a lo largo del tiempo. Esto permitirá adoptar decisiones fiables sobre el uso del sistema público. Por otra parte, sería de gran interés crear una página centrada en el conocimiento y exploración de la topología de la red por parte de los usuarios. Además de las páginas anteriormente mencionadas, el diseño e implementación de un visor del historial de viajes de los pasajeros proporcionaría un punto de gran interés para los clientes del transporte público.

Capítulo 2

Fundamentos tecnológicos

En este capítulo se lleva acabo un análisis sobre los proyectos existentes más relevantes que se alinean con los objetivos previamente establecidos en la sección 1.2, así como de las herramientas y tecnologías utilizadas para llevar a cabo el trabajo.

2.1 Estado del arte

El objetivo de esta sección consiste en la búsqueda de otros proyectos ya existentes cuyos objetivos sean convergentes con los expuestos en este proyecto. Se trata de analizar y aprovechar los conocimientos, experiencias y soluciones disponibles que aportan investigaciones previas dentro del mismo campo de estudio.

En este sentido, esta sección se va a centrar básicamente en dos plataformas: Snap4City [2] y Amsterdam Smart City Dashboard [3].

Snap4City es una plataforma que tiene como fin la mejora de la calidad de vida de las personas. Para ello focaliza sus esfuerzos en optimizar la gestión de la información relativa a diversos recursos urbanos como se observa en la Figura 2.1. Los beneficios que esta aplicación aporta a la ciudad, se basan en la obtención de información objetiva, precisa y dinámica, pues se actualiza constantemente en tiempo real. Esto permite a los directivos encargados de la gestión de la ciudad aportar soluciones a cuestiones como el tráfico y sus derivados: aglomeraciones de personas, calidad del aire, eficiencia energética, etc. Para ello, se utilizan diferentes recursos y técnicas como Big Data, Inteligencia Artificial (IA) y el Internet de las cosas (IoT), los cuales recogen información proveniente de diversos elementos técnicos como sensores o cámaras.

La plataforma *Amsterdam Smart City Dashboard*, desarrollada por Geodan, está en la misma línea que la plataforma anteriormente referida. Su objetivo es facilitar la gestión urbana, centrándose en el caso concreto de la ciudad de Amsterdam. Sigue el mismo patrón en el procedimiento de obtención, análisis y procesamiento de datos que la herramienta *Snap4City*. Su

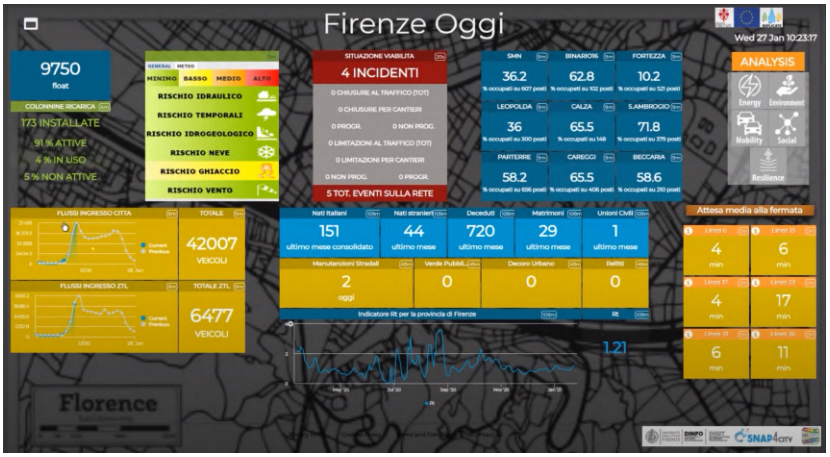


Figura 2.1: Página principal de la Snap4city aplicado a la ciudad de Florencia

objetivo principal es ofertar a las personas responsables de la gestión de la capital de los Países Bajos, información relevante para la toma de decisiones eficaces y eficientes en relación con los servicios públicos de la urbe. A su vez, destaca por el uso de información espacial, plasmada a través de mapas altamente visuales. Se puede observar su pantalla principal en la Figura 2.2.

Figura 2.2: Página principal del Amsterdam City Dashboard

Este TFG se centra en las necesidades de transporte público del municipio de Madrid, concretamente en su gestión y optimización. A través de este proyecto se pretende posibilitar la consulta y visualización de datos relativos a los medios de transporte, como son la información sobre las rutas, paradas y patrones de uso de los viajeros, así como la identificación de áreas susceptibles de mejora en la red de transportes. Tiene, por lo tanto, un enfoque más restringido y especifico que *Snap4City* y *Amsterdam Smart City Dashboard*, aplicaciones que aportan

soluciones más ambiciosas pues se refieren a diversos aspectos de una ciudad inteligente.

Este proyecto se diferencia a los previamente mencionados en que no solamente se podrán visualizar los datos y gestionar el transporte a través de un panel de control, sino que también se ofrece el diseño y desarrollo de una página web íntegra a través de la cual los usuarios podrán efectuar consultas relativas a la topología de la red de transporte o a su propio registro de viajes, con el objetivo de mejorar su experiencia en relación en el uso del servicio público.

2.2 Tecnologías utilizadas

- Python [4]. Lenguaje sencillo de programación de alto nivel orientado a objetos utilizado para el manejo de grandes volúmenes de datos. Además, destaca su recurrente uso en proyectos de Inteligencia Artificial.
- R [5]. Lenguaje de programación centrado en el análisis estadístico de los datos. R se encarga de la realización de múltiples tipo de análisis y visualizaciones gracias a la amplia gama de paquetes que posee.
- RStudio [6]. Aplicación que habilita el uso de R a través de una interfaz eficiente e intuitiva.
- PostgreSQL [7]. Sistema de gestión de bases de datos relacional que dispone de capacidades para el almacenado y manipulación de datos en tablas relacionales a través del uso de consultas en lenguaje SQL.
- PostGIS [8]. Extensión de PostgreSQL usada principalmente en aplicaciones SIG que permite almacenar datos de naturaleza geoespacial, sobre los que se pueden realizar una amplia variedad de consultas espaciales complejas.
- Leaflet [9]. Librería que permite la visualización de mapas interactivos usando otras tecnologías web como HTML, CSS y JavaScript de manera simple, flexible y personalizable.
- Django [10]. Framework de desarrollo web que posibilita construir aplicaciones web de manera eficiente y escalable, el cual está basado en Python.
- GeoDjango [11]. Módulo de Django orientado a la elaboración de aplicaciones web geográficas, de gran utilidad en este proyecto.
- PowerBI [12]. Herramienta, muy utilizada en el mundo empresarial, que está centrada en el análisis y visualización de datos. Permite combinar y transformar conjuntos de datos de varias fuentes para crear visualizaciones interactivas.

- Pandas [13]. Biblioteca de Python utilizada para el análisis y manipulación de datos a través de la cual es posible organizar y trabajar con los datos de manera sencilla e intuitiva a través de estructuras de datos flexibles y eficientes.
- Matplotlib [14]. Biblioteca centrada en la visualización de datos en Python que destaca por la creación de todo tipo de gráficos para representar información de manera efectiva y visualmente atractiva.
- Seaborn [15]. Biblioteca de visualización de datos basada en Matplotlib que proporciona una amplia variedad de funciones enfocadas en la creación de visualizaciones visualmente atractivas.
- Scipy [16]. Biblioteca de Python que se encarga de proporcionar una serie de herramientas y funciones para la realización de cálculos numéricos.

Capítulo 3

Metodología y planificación

En este capítulo se hace referencia a los aspectos metodológicos empleados a la hora de abordar el proyecto así como a la planificación y seguimiento realizado para la ejecución del mismo.

3.1 Metodología de desarrollo

La metodología empleada, ha sido interactiva e incremental, mediante la cual se ha dividido el proyecto en bloques reducidos y manejables llamados iteraciones. Así se aborda el trabajo de forma progresiva y adaptable, pues en cada iteración avanza poco a poco, posibilitando la identificación y resolución de posibles problemas en etapas tempranas. Al hacerlo paso a paso se puede afirmar que siempre se ha mantenido un seguimiento, comunicación y colaboración entre todos los actores: el equipo de desarrollo y los interesados, que en este caso han sido los tutores, y el alumno como ejecutor y desarrollador del proyecto.

También este sistema ha permitido la mejora continua, aplicando la técnica de PDCA (Plan/Do/Check/Act) [17] y redundando en mayor calidad y eficacia TQM [18] (Total Quality Management) al hacer la gestión de los cambios necesarios adecuadamente.

Cada bloque o iteración consta de planificación y diseño (Plan), implementación o aplicación (Do), prueba de un conjunto específico de funcionalidades (Check) y finalmente correcciones o mejoras (Act) que constituyen fases estructuradas que se repiten para avanzar con seguridad, eficiencia y sobre todo eficacia.

- **PLAN.** En la planificación se determinan los objetivos principales del proyecto, además de las metas a alcanzar en la primera iteración. También, en esta fase, se analizan de forma detallada los requisitos mediante una estrecha colaboración entre interesados y usuarios, con la finalidad de entender sus necesidades y expectativas. Una vez realizado lo anterior, se ha previsto el alcance y recursos necesarios para su realización.

- **DO.** Inicialmente, en la fase de diseño quedan definidas las estructuras de datos, la arquitectura del sistema y se planifican las pruebas de validación. Posteriormente, se realiza la implementación con la codificación y desarrollo del software según el diseño previamente establecido. Se construyen los componentes y se integran las funcionalidades dentro del sistema en desarrollo. Es importante seguir buenas prácticas de programación y realizar pruebas unitarias durante esta fase.

- **CHECK.** Sobre el software desarrollado se realiza una fase de pruebas y validación para garantizar el correcto funcionamiento y el cumplimiento de los objetivos del proyecto. Entre estas pruebas destacan la dedicadas a la integración y el rendimiento, las cuales se realizan para comprobar que las nuevas funcionalidades integradas cumplen con lo previsto.

- **ACT.** En la fase de evaluación y retroalimentación se realiza un profundo análisis de los resultados obtenidos. Para ello, se recopila el *feedback* de los usuarios y los interesados con la finalidad de ajustar y mejorar el plan para las iteraciones futuras.

Este mismo proceso explicado para la primera iteración, se repite con las posteriores, seleccionando nuevas funcionalidades a desarrollar además de ajustar los requisitos y diseños en base a la retroalimentación recibida. Se añaden nuevas funcionalidades y se mejora el software existente con cada paso hasta alcanzar el resultado final deseado.

Como se ha explicado este método incremental da como resultado una ganancia constante o incrementos que nos aproxima al objetivo perseguido. Todo esto se puede observar en las Figura 3.1 y Figura 3.2.

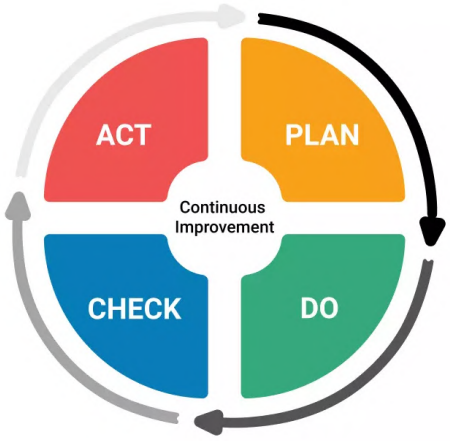
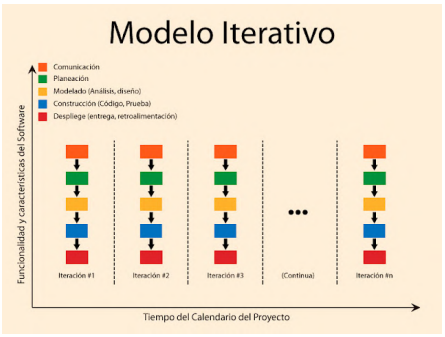
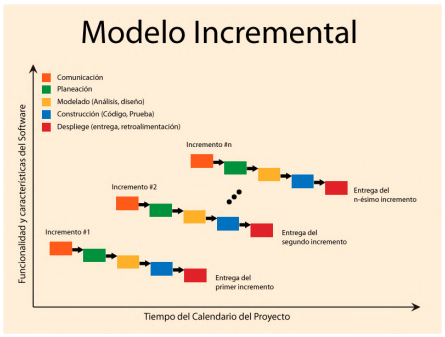


Figura 3.1: Ciclo PDCA
[17]



(a) Modelo Iterativo



(b) Modelo Incremental

Figura 3.2: Metodología Iterativa Incremental.
[19]

3.1.1 Herramientas de apoyo a la metodología

- Visual Studio Code (VSCode) [20]. Editor de código que permite el uso de mútiples lenguaje como R, Python o JavaScript. Sus puntos fuertes son su facilidad de uso y alto grado de personalización. Se utilizó en el proyecto tanto para el análisis exploratorio del conjunto de datos como para el diseño e implementación de la aplicación web.

- GitHub [21]. Plataforma web basada en Git, la cual se usa en el alojamiento, administración y colaboración de proyectos de software. En el caso particular de este proyecto, sirvió para realizar el seguimiento de las iteraciones del proyecto.

- Overleaf [22]. Aplicación en línea usada la creación y edición de textos, basado en el sistema de composición de textos *Latex*. Se ha empleado para la redacción de la memoria del proyecto.

- Draw.io [23]. Aplicación web que se usa para la creación de una amplia gama de diagramas. Se empleó para hacer tanto el diagrama de Gantt como para los modelos conceptuales del proyecto.

- Marvel [24]. Plataforma *online* de diseño que, entre otras funcionalidades, posibilira la creación de interfaces y experiencias de usuario. Es de gran utilidad para la visualización y prueba de ideas, previas a la implementación de un software. Su usó en el proyecto se basó en la creación de las mockups del apartado 4.5.

3.2 Planificación y seguimiento

En esta sección se describen tanto la planificación inicial planteada para el proyecto como el seguimiento real del mismo.

3.2.1 Planificación

Para lograr el éxito en un proyecto, es fundamental contar con una adecuada planificación y organización en etapas o fases claramente establecidas. Esto garantiza una ejecución eficiente y la consecución de los objetivos planteados.

En primer lugar, se realizaron dos reuniones preliminares entre el alumno y los directores con la finalidad de fijar el alcance y los requisitos del proyecto. Por otra parte, además de realizar un estudio para decidir las tecnologías que se emplearían durante la ejecución del trabajo, se tomó la determinación de buscar principalmente un enfoque centrado en la rama de ingeniería de datos principalmente, siendo apoyado, en menor medida, por aspectos más relacionados con la rama científica del grado.

A continuación, se llevó a cabo la división del tiempo estimado del proyecto en 7 iteraciones con una duración de entre 2 y 3 semanas para cada uno. Estos ciclos de desarrollo se abordarían con la metodología descrita en la sección 3.1.

Por último, se destinó un mes para la redacción y posterior revisión de la memoria de proyecto.

En lo que respecta a los recursos utilizados en la ejecución del proyecto, se pueden destacar los siguientes:

- **Recursos técnicos.** Se refiere al conjunto de herramientas, instrumentos o materiales o necesarias para una completa ejecución del proyecto. Se dividen en dos categorías: recursos hardware y recursos software. El único recurso hardware utilizado fue el ordenador portátil del alumno. Por otra parte, los recursos software usados se mencionan en la sección 2.2.

- **Recursos humanos**. Son los referidos a las personas con los conocimientos y destrezas necesarios para alcanzar la correcta consecución de los objetivos del trabajo. En este tipo de recursos incluyen tanto a los directores del proyecto, por su labor de organización, apoyo y seguimiento, como al alumno, por su papel de artífice principal en la elaboración de Trabajo Fin de Grado (TFG).

3.2.2 Seguimiento

Como se ha comentado en el apartado anterior, a mayores de la realización de dos reuniones previas para la determinación del alcance, requisitos y objetivos, se dividió el tiempo estimado del proyecto en 7 iteraciones. En este apartado, se describe en detalle lo que se ha hecho en cada uno de dichos intervalos.

- **Iteración 1.** Se realizaron sucesivas pruebas de carga del dataset de viajes completo en software orientado a la analítica de datos, pero debido a su gran tamaño (151,1 millones

de filas), se optó por la división de los datos en bloques más pequeños y manejables para su posterior exploración usando R. Por otra parte, se llevó a cabo la definición inicial del modelo conceptual de la tablas de datos que se usarían durante el proyecto.

- **Iteración 2.** Después de realizar una prueba inicial con R, se decidió rehacer el análisis utilizando Python. Además, se comenzaron a realizar desarrollos y pruebas iniciales de la aplicación web utilizando Django [25] [26]. Estas pruebas se hicieron a modo de tutorial, para comprender el funcionamiento del framework, tanto de manera general como concreta en el uso de datos espaciales. A su vez, se refinó el modelo conceptual realizado en la primera iteración.

- **Iteración 3.** Se hizo un análisis exploratorio completo en Python, con resultados detallados. Además, una vez que se comprendió el uso y funcionamiento de Django, se inició el desarrollo de la interfaz de administración de la plataforma. También, se configuró el modelo de datos del proyecto en el sistema gestor de bases de datos PostgreSQL y se creó la primera página de la aplicación. Esta página se enfocaría en la búsqueda del historial de viajes de los usuarios del transporte público.

- **Iteración 4.** En esta iteración, se propuso desarrollar la segunda página del sitio web y mejorar la primera. Para la segunda página, se creó un panel de control interactivo, conocido como *dashboard*, que combinaría los datos de viajes de los usuarios con la información sobre la topología de la red de transporte. Para lograrlo se utilizó el software de visualización PowerBI.

- **Iteración 5.** El enfoque se centró en completar la última página de la aplicación web, la cual se enfocaría en los datos de la topología de la red de transporte del municipio madrileño.

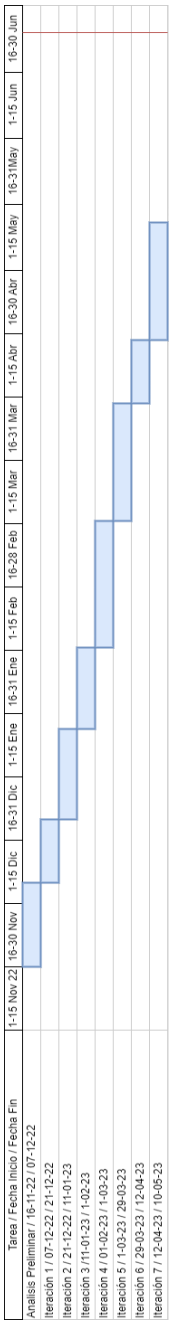
- **Iteración 6.** Se realizó un esfuerzo de refinamiento del trabajo, enfocándose en diversas tareas importantes. Entre estas labores se destacan la mejora y limpieza del análisis exploratorio, el aumento de la eficiencia en las consultas al servidor PostgreSQL, y la mejora y revisión tanto del Front-End como del Back-End de la aplicación web, entre otras acciones.

- **Iteración 7.** Se destinó el resto del tiempo a la redacción de la memoria.

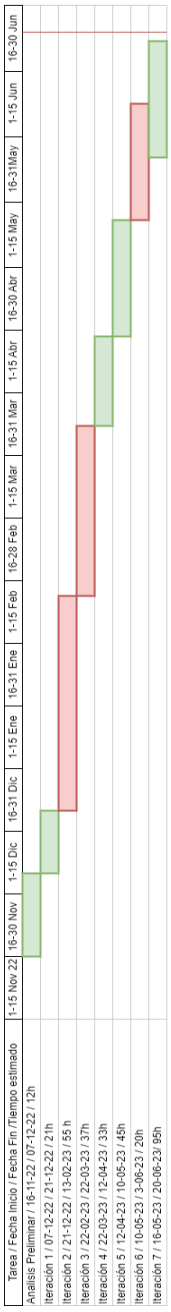
  La presencia de diversos problemas técnicos durante ciertas iteraciones, combinada con eventos como el período de exámenes del primer cuatrimestre en la facultad o el trabajo en empresa por parte del alumno, provocó que la planificación inicial no se pudiera cumplir. Entre dichos problemas destacan:

- Problemas con el gran tamaño del dataset de viajes en el análisis exploratorio. Para ello, como se comentó en su relativo *sprint*, se hizo la división en bloques más pequeños.
- Problemas con la instalación y configuración de Django y GeoDjango. Para solventarlos se llevó a cabo una exploración de información útil a través de diversas páginas de la web [27] [28].
- Problemas con la carga de datos en PostrgeSQL de manera general, y también de forma específica en el caso de los campos con formato temporal (*datetime*) y decimal en PostgreSQL. Para abordarlo se importaron los datos, con una sentencia *COPY*, en formato texto, y se realizó su modificación a los formatos apropiados a través de sentencias en el lenguaje de programación SQL.
- Fallos al pintar las líneas de transporte sobre el mapa en la última de las páginas de la aplicación web. Para solventar este problema se optó por el uso del la API RouteMachine.js de Leaflet [29].
- Problemas de eficiencia de consultas al servidor. Estos fueron solventados a través de la creación de índices.

  A continuación, en la Figura 3.3, se presenta el diagrama de Gantt de proyecto, que muestra la notable diferencia entre el tiempo estimado y el tiempo real. Se muestran en color rojo aquellos sprints que tomaron más tiempo del esperado.



(a) Planificación inicial



(b) Seguimiento real

Figura 3.3: Diagrama de Gantt del proyecto

Capítulo 4
Análisis

En esta sección se aborda lo relativo al análisis y documentación de todos los aspectos de la aplicación web previos a su posterior diseño e implementación.

4.1 Análisis Preliminar

Inicialmente, se llevó a cabo un análisis preliminar de los datos con el objetivo de familiarizarse con las bases de datos que se utilizarían. Para ello, los directores del proyecto proporcionaron al alumno una serie de conjuntos de datos de prueba que representaban lo que serían las bases de datos completas del proyecto. Gracias a este análisis previo, se pudieron comprender todos los campos y la estructura de los datasets.

Por una parte, se comenzó con el análisis de la base de datos de viajes. Este conjunto de datos contiene información detallada sobre cada clic realizado por los viajeros en su ingreso al medio de transporte público. El Consorcio de Transportes de Madrid recopila y almacena datos reales de cada pasajero al acceder al transporte público utilizando su tarjeta. De esta base de datos se puede obtener la siguiente información de sus campos:

- ***Tarjeta:*** Indentificador de tarjeta de viaje del usuario de transporte público.
- ***Fecha:*** Fecha y hora en la que el usuario realiza la transacción al pasar la tarjeta sobre la máquina de acceso al transporte.
- ***Título:*** Tipo de abono que posee el usuario. Dentro de estos tipos pueden encontrarse abonos anuales, mensuales y trimestrales, entre otros.
- ***Tusuario:*** Tipo de usuario que posee la tarjeta. Sus valores dependen de la edad del pasajero o del tipo de tarjeta que tenga.
- ***Descuento:*** Descuento asignado al viaje. Dentro de los descuentos posibles destacan los asignados a transbordos entre líneas.

- ***Dpaypoint:*** Brinda información sobre el tipo de transporte, la línea y la parada asociadas al inicio del trayecto.
- ***IDTLV:*** Consiste en un código que permite distinguir el tipo de máquina sobre la cual se hizo el clic con la tarjeta que permite contabilizar los datos de los viajes. Hay 4 tipos posibles:
  - *C0*: Corresponde con las máquinas de acceso al sistema de metro del municipio.
  - *C1*: Corresponde con las máquinas de acceso a los autobuses del municipio.
  - *C6*: Corresponde con las máquinas de acceso a los trenes del municipio.
  - *G2*: Corresponde con el sistema de validación a bordo del vehículo.
- ***Codval:*** Código de validez del paso de la tarjeta. Este código muestra información relacionada con el mensaje que el usuario visualiza en la pantalla al realizar el contacto de su tarjeta con la máquina de acceso al transporte.

Por otra parte se hizo un modelado conceptual inicial de las bases de datos del GTFS del metro de Madrid [30].

El GTFS es un estándar que se encarga de definir la información relacionada con el transporte público mediante un formato estructurado. Dentro de la información que comparte destacan los datos sobre rutas, horarios, paradas, etc. Sobre los datos del GTFS del metro de Madrid se llevó a cabo un diseño lógico, mostrado en la Figura 4.1, y un modelo entidad relación, presente en la Figura 4.2. A su vez, se realizó una descripción de los campos de cada una de las tablas del modelo, la cual se muestra a continuación:

- ***Rutas/ Routes***
  - *route id*: Identificador único de la ruta.
  - *agency id*: Identificador de la agencia responsable de la ruta.
  - *route short name*: Nombre corto de la ruta.
  - *route long name*: Nombre completo o largo de la ruta.
  - *route desc*: Descripción de la ruta.
  - *route type*: Tipo de ruta (código numérico).
  - *route url*: URL o enlace relacionado con la ruta.
  - *route color*: Color asignado a la ruta.
  - *route text color*: Color del texto asociado a la ruta.
- **Paradas/ Stops**

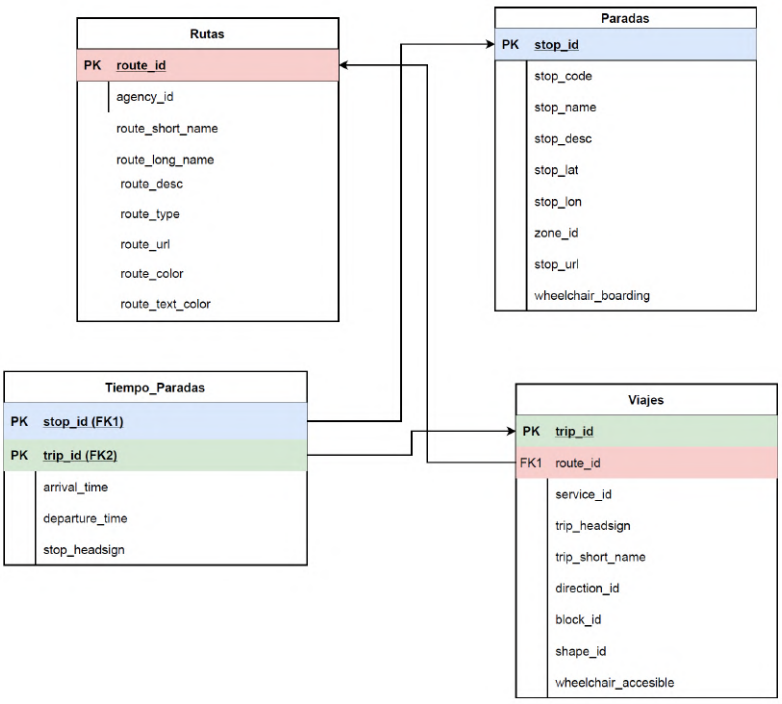


Figura 4.1: Diseño lógico preliminar del GTFS de Madrid

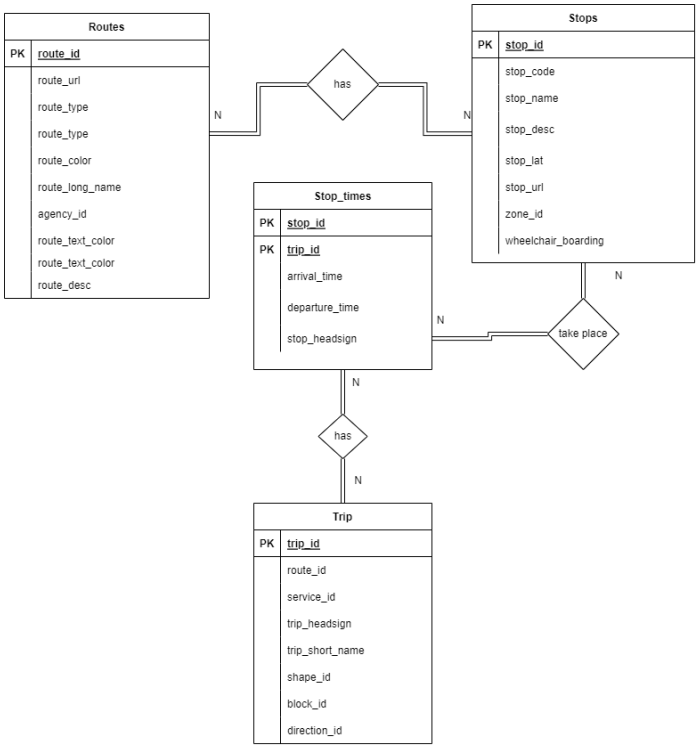


Figura 4.2: Modelo entidad-relacion preliminar del GTFS de Madrid

  - *stop id*: Identificador único de la parada.
  - *stop code*: Código de la parada.
  - *stop name*: Nombre de la parada.
  - *stop desc*: Descripción de la parada.
  - *stop lat*: Latitud de la ubicación de la parada.
  - *stop lon*: Longitud de la ubicación de la parada.
  - *zone id*: Identificador de la zona de la parada.
  - *stop url*: URL o enlace relacionado con la parada.
  - *wheelchair boarding*: Indicador de accesibilidad para sillas de ruedas.

- **Viajes/ Trips**

  - *route id*: Identificador de la ruta de transporte.
  - *service id*: Identificador del servicio de transporte.
  - *trip id*: Identificador del viaje específico.
  - *trip headsign*: Destino o nombre de la cabeza de línea del viaje.
  - *trip short name*: Nombre corto o abreviado del viaje.
  - *direction id*: Identificador de dirección del viaje (generalmente 0 o 1, que representan direcciones opuestas).
  - *block id*: Identificador del bloque de servicio.
  - *shape id*: Identificador de la forma o ruta geométrica del viaje.
  - *wheelchair accessible*: Indicador de accesibilidad para sillas de ruedas.

- **Tiempo_paradas/ Stop_times**

  - *stop id*: Identificador de la parada.
  - *trip id*: Identificador del viaje específico.
  - *arrival time*: Hora de llegada a una parada en particular durante el viaje.
  - *departure time*: Hora de salida de una parada en particular durante el viaje.
  - *stop headsign*: Indicador del nombre o dirección de la parada.

Para llevar a cabo este modelado conceptual, se omitieron ciertas tablas y campos proporcionados por el GTFS, ya que se consideró que eran irrelevantes para el desarrollo del proyecto. Por ejemplo, se omitieron tablas relacionadas con información del calendario de viajes del sistema público, o con información general de la agencia de transporte encargada del servicio.

Por otra parte, durante el análisis preliminar también se llevó a cabo el diseño de interfaces gráficas de usuario de la aplicación. Estos diseños se han documentado en la sección 4.5.

4.2 Análisis Exploratorio

Tal y como se mencionó en la sección 3.2, el siguiente paso después de la fase de análisis preliminar del proyecto, fue realizar un análisis exploratorio del conjunto de datos de viajes. Para llevarlo a cabo, se siguió la metodología descrita en el apartado 3.1.

En primer lugar se realizó la descarga de los datos, los cuales fueron proporcionados por los directores del proyecto en formato de compresión '.zip'. Como los datos eran demasiado grandes se decidió trabajar solo con los datos de enero de 2019, debido a que en este conjunto de datos no existirían anomalías causadas por la pandemia. Estos datos de un solo mes ya ocupaban un espacio en disco próximo a los 12 *GigaBytes*.

El hecho de que solamente los datos de enero tuviesen ese gran tamaño, permitió obtener la idea de que la futura aplicación debería estar preparada el manejo de volúmenes de datos todavía superiores. Esto se dedujo debido a la futura necesidad de incluir datos de otros meses del año o, incluso, de otros años distintos. Por lo tanto, se infirió que la escalabilidad debería ser un característica indispensable a tener en cuenta en el desarrollo de la aplicación web.

Posteriormente, se procedió a la carga de los datos usando la herramienta de análisis estadístico RStudio. Se utilizó la función *read_csv*, donde se especificó la ruta del archivo. Sin embargo, debido al gran volumen de datos, esta operación no lograba completarse.

Como solución a este inconvente se decidió hacer una divisón del dataset completo en bloques de menor tamaño y, así, poder realizar el análisis de manera efectiva. Para lograr este objetivo, se hizo el código sencillo en Python que se muestra a continuación [31] [32]:

```
import pandas as pd
chunk_size=500000
batch_no=1
for chunk in
    pd.read_csv(r'C:/Users/34610/Documents/TFG/ENERO_VIAJES_2019.txt'
,sep='#',chunksize=chunk_size):
    chunk.to_csv('chunk'+str(batch_no)+'.csv',index=False)
    batch_no+=1
```

En primera instancia, se optó por la división en bloques de 5 millones de filas cada uno, como se observa en el código. Sin embargo, esto generaba una gran cantidad bloques lo cual impedía una análisis eficiente y práctico. Para solucionar este problema, se decidió dividir el dataset en bloques de 60 millones de filas. Como resultado de este proceso, se obtuvieron 3 bloques. Los dos primeros con 60 millones de filas y el tercero con 31,1 millones aproximadamente. Gracias a este proceso de segmentación, se pudo conocer que el conjunto de datos estaba formado por 151,1 millones de filas.

Una vez se tenía la división hecha se pasó con la carga de datos y posterior ejecución del análisis en RStudio. Se comenzó el análisis con funciones generales para conocer el dataset

como las siguientes:

```
str(datos) # Estructura de la base de datos

head(datos)# Primeras filas

tail(datos) # Últimas filas

hist(datos$CODVAL)# Distribucion de la variable

plot(datos$CODVAL, datos$DESCUENTO)# Relaciones entre las variables
```

Sin embargo, como se comentó previamente, tras un corto contacto con la plataforma se consideró la realización del análisis exploratorio con Python. Este cambio se debió a la naturaleza cualitativa de los datos del conjunto de viajes. Se consideró que el alumno tendría una curva de aprendizaje mucho más rápida con Python que con R en el análisis de este tipo de datos, ya que en el grado, el uso del lenguaje R, se centró principalmente en la exploración datos cuantitativos o numéricos.

Para comenzar el análisis en Python, se importaron los paquetes necesarios, incluyendo Pandas, Seaborn y Matplotlib. Las etapas seguidas en el análisis son las siguientes: carga de datos, exploración inicial, limpieza y preprocesado, análisis de frecuencia de las variables y análisis con un subconjunto aleatorio.

4.2.1 Carga de datos

En esta etapa, se utilizan las herramientas proporcionadas por el paquete Pandas para cargar los datos desde una fuente externa, como un archivo CSV, Excel, una base de datos, etc.

Se usó la función $pd.read\_csv(\text{"}ruta\text{"})$ para la carga. Como resultado de este proceso se obtuvieron los siguientes mensajes de advertencia para cada uno de los bloques:

- **Chunk 1 y Chunk 2:** *DtypeWarning: Columns (2) have mixed types.*
- **Chunk 3:** *DtypeWarning: Columns (2,4) have mixed types.*

Estos mensajes indican que en ciertas columnas del dataset se encuentran tipos de datos distintos para la misma variable. Por ejemplo, una variable 'FECHA' tendría tipos mezclados si alguna de las filas de dicha variable tuviesen un *dtype='datetime'* y otras un *dtype='object'*.

En nuestro caso encontramos que los tipos de datos mezclados se encuentran en las columnas 'Tusuario' (chunks 1, 2 y 3) y 'Descuento' (chunk 3). La gestión de este inconveniente se realizó en pasos posteriores del análisis.

4.2.2 Exploración inicial

En esta etapa, se realizó la visualización de las primeras y últimas filas del dataset, así como la obtención de información básica del mismo.

Para el primero de los pasos se usaron las funciones *.head()* y *.tail()* como se muestra a continuación:

```
print('PRIMER CHUNK : \n Primeras filas \n',df.head(),'\n Últimas
    filas \n',df.tail())
```

El código de la Figura 4.3 produjo la siguiente salida:

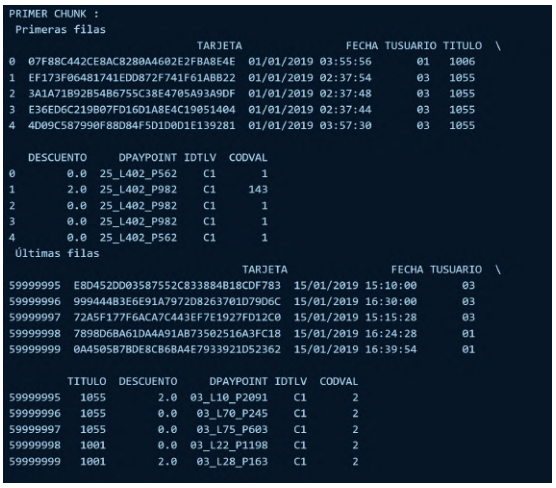


Figura 4.3: Primeras y últimas filas del primer bloque.

Se realizó el mismo proceso para los siguientes 2 bloques. La exploración y visualización de las primeras y últimas filas del dataset nos brindó una visión general del conjunto de datos. Por otra parte, se empleó la función *info()* para obtener información básica de cada uno de los bloques. Esto se puede observar en la Figura 4.4.

Se pudo observar que los objetos introducidos en la función son dataframes que poseen 60 millones de filas cada uno, en los cuales se muestra información acerca de 8 variables. En el tercer chunk existen 31,1 millones de filas debido a que a la hora de dividir el dataset entero en 3 chunks de tamaño de 60 millones de filas por bloque este último no se llegó a completar. Por lo tanto, el número exacto de filas de dataset es 151.094.796. Además, se observó que los tipos de datos de las variables *Fecha*, *Descuento* y *Codval* no son los correctos. Por otra parte, esta función permitió hacer un análisis individualizado de cada una de las variables:

- Tarjeta: La variable 'tarjeta' es de tipo 'obj', lo cual implica que consiste en vectores de caracteres o cadenas de texto. En este caso particular, la variable 'tarjeta'se compone de un string de 32 caracteres alfanuméricos.

```
print(df.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60000000 entries, 0 to 59999999
Data columns (total 8 columns):
 #   Column      Dtype
---  ------      -----
 0   TARJETA     object
 1   FECHA       object
 2   TUSUARIO    object
 3   TITULO      object
 4   DESCUENTO   float64
 5   DPAYPOINT   object
 6   IDTLV       object
 7   CODVAL      int64
dtypes: float64(1), int64(1), object(6)
memory usage: 3.6+ GB
```

(a) Primer bloque

```
print(df2.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60000000 entries, 0 to 59999999
Data columns (total 8 columns):
 #   Column      Dtype
---  ------      -----
 0   TARJETA     object
 1   FECHA       object
 2   TUSUARIO    object
 3   TITULO      object
 4   DESCUENTO   float64
 5   DPAYPOINT   object
 6   IDTLV       object
 7   CODVAL      int64
dtypes: float64(1), int64(1), object(6)
memory usage: 3.6+ GB
```

(b) Segundo bloque

```
print(df3.info())
[15]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31094796 entries, 0 to 31094795
Data columns (total 8 columns):
 #   Column      Dtype
---  ------      -----
 0   TARJETA     object
 1   FECHA       object
 2   TUSUARIO    object
 3   TITULO      object
 4   DESCUENTO   object
 5   DPAYPOINT   object
 6   IDTLV       object
 7   CODVAL      int64
dtypes: int64(1), object(7)
memory usage: 1.9+ GB
None
```

(c) Tercer bloque

Figura 4.4: Información básica sobre el conjunto de datos

- Fecha: La variable en cuestión es de tipo 'obj'. Los datos de esta variable están formateados como 'dd/mm/yyyy hh:mm:ss'. Dado que representa una combinación de fecha y hora, podría ser beneficioso cambiarla al tipo de datos 'datetime'. Al convertir la variable al tipo 'datetime', se podrá realizar operaciones y cálculos específicos de fechas y horas, como extraer componentes individuales (día, mes, año, hora, etc.), calcular diferencias de tiempo y realizar análisis temporal más avanzado.

- Título: Esta variable también es de tipo 'obj'. En este caso, es un string compuesto por dos dígitos numéricos. Dado que el título se trata como una variable cualitativa, no tiene sentido convertirla a tipo numérico o entero.

- Tusuario: La columna 'Tusuario' muestra el tipo de usuario asociado a la tarjeta metropolitana. Esta variable es cualitativa y de tipo 'obj'. Consiste en un string compuesto por dos dígitos numéricos, los cuales se tratarán como una cadena de texto.

- Descuento: Esta es la primera y única variable de tipo 'float64'. Este tipo de dato sirve para representar números reales. En este caso en particular, podríamos convertir dicha variable a string (*tipo obj*) ya que se trata de una variable teóricamente categórica que se encarga de clasificar los distintos viajes según los tipos de descuento aplicados en ellos.

- Dpaypoint: Es una varible de tipo 'obj'. Se trata de una serie de strings con el siguiente formato: 'nn_Lnnn_Lnn'. Se tiene en cuenta que n hace referencia a un número entero y L a una letra mayúscula. Se encuentan en el formato óptimo para su utilización.
- IDTLV: Se trata de otra variable de tipo 'obj'. Es un string con el formato 'Ln' teniendo en cuenta que n hace referencia a un número entero y L a una letra mayúscula. Se encuentran en el formato óptimo para su utilización.
- Codval: La variable en cuestión es de tipo 'int64', lo que indica que trata números enteros. Sin embargo, dado que se trata de una variable categórica que representa el tipo de código de validación devuelto por un sensor de metro al pasar una tarjeta, podría ser interesante convertir esta variable al tipo 'obj' (objeto) o 'string' (cadena de caracteres).

A su vez, se utilizó la función *.unique()* sobre los campos para obtener la cantidad de valores distintos que puede tomar cada variable, como se ve en la Figura 4.5.

```
print('Cantidad de valores distintos que puede tomar la columna TARJETA :',len(df.TARJETA.unique()))
print('Algunos de ellos: \n',df.TARJETA.unique())

Cantidad de valores distintos que puede tomar la columna TARJETA : 3577486
Algunos de ellos:
 ['07F88C442CE8AC8280A4602E2FBA8E4E' 'EF173F06481741EDD872F741F61ABB22'
 '3A1A71B92B54B6755C38E4705A93A9DF' ... 'A805181B1AEA1676775591125A0BD525'
 '5CBFC2B767C8237B70815B705FABAEF2' '99A6B4D5DD4647C48C736BA78B6ECFED']
```

Figura 4.5: Uso de la función unique sobre la variable *Tarjeta*

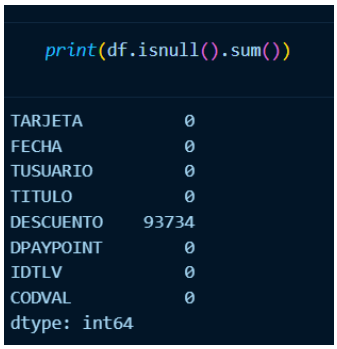
De esta exploración se obtuvieron la siguientes conclusiones:

- La variable con más valores distinto es TARJETA con cerca de 4,6 millones de identificadores diferentes (4.591.961 millones).
- La variable con menos valores distinto es IDTLV (4 valores).
- A la hora de realizar gráficos de distribución de densidad sólo los efectuaremos sobre aquellas variables con pocos valores únicos como Tusuario (16 valores), DESCUENTO (16 valores) y IDTLV (4 valores).

4.2.3 Limpieza y preprocesado

En esta sección, se lleva a cabo el tratamiento de posibles valores nulos presentes en el conjunto de datos. Por otra parte, también se realizó la corrección de problemas relacionados con tipos de datos incorrectos o mezclados que se identificaron en fases anteriores.

Para determinar la presencia y cantidad de valores nulos en nuestro conjunto de datos, se aplicaron las funciones *.isnull()* para identificar cada valor nulo y *.sum()* para obtener el total de valores nulos. Este proceso es observado en la imagen 4.6.



(a) Primer bloque

(b) Segundo bloque

(c) Tercer bloque

Figura 4.6: Análisis de nulos en el conjunto de datos.

En sendos bloques o 'chunks' existe un denominador común: existencia de nulos en la variable 'Descuento'.

- Chunk 1: Encontramos 93.734 nulos en la variable 'Descuento'. Porcentualmente al tamaño del bloque (60M), poseen un 0,16% de valores nulos en sus filas.
- Chunk 2 : Encontramos 89.929 nulos en la variable 'Descuento'. Porcentualmente al tamaño del bloque (60M), poseen un 0,15% de valores nulos en sus filas.
- Chunk 3: Encontramos 48.569 nulos en la variable 'Descuento'. Porcentualmente al tamaño del bloque (31M), poseen un 0,16% de valores nulos en sus filas.

Esta casuística se puede gestionar de dos maneras posibles:

- Sustituir el nulo por un código 0, significando la aplicación de ningún descuento.
- Asumir que el nulo fue un error en el sistema y crear un nuevo código de 'no se conoce'

en alusión al desconocimiento de la aplicación de algún tipo de reducción en el coste del viaje.

En esta ocasión se optó por la segunda de las opciones, tal y como se puede observar en la imagen 4.7.

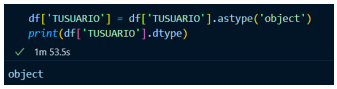
Figura 4.7: Adecuación del tipo de datos y tratamiento de nulos en la variable descuento

En este extracto de código se convierte la columna 'Descuento' al tipo de dato 'object', se imprime el tipo de dato resultante, se rellenan los valores nulos en la columna 'Descuento' con la cadena 'No se conoce', y finalmente se imprime la cantidad de valores nulos en esa columna. Como se puede comprobar la cantidad de nulos resultantes es 0. El proceso se repite para los otros dos bloques.

A continuación, se ajustan los tipos de datos de los campos Codval, Tusuario y Fecha a los tipos más apropiados. El proceso se repitió para los 3 bloques.

(a) Campo *Codval*

(b) Campo *Fecha*



(c) Campo *Tusuario*

Figura 4.8: Asignación de los tipos de datos adecuados para los campos *Codval*, *Tusuario* y *Fecha*.

Se logró realizar con éxito el cambio de tipo de datos en los campos *Codval* y *Tusuario*. No obstante, en el caso de la columna Fecha, no fue posible completar la operación debido a su tiempo de procesamiento excesivamente largo, lo que impidió su finalización. Este hecho, imposibilitó el aprovechamiento de la componente temporal del campo Fecha, ya que al no conseguir cambiar su tipo de datos se estaría tratando con una cadena de texto, la cual no aporta la información de manera adecuada. Sin embargo, este problema fue resuelto más adelante como se comenta en la sección 6.1.2.

4.2.4 Análisis de frecuencia de las variables

En esta etapa, se lleva a cabo la exploración y análisis de los valores más frecuentes presentes en cada una de las variables del conjunto de datos. A través de esta exploración se busca conseguir un mayor conocimiento sobre los patrones de repetición en los datos.

Para llevar a cabo este análisis, se utilizó la función *value_counts* en cada uno de los campos del conjunto de datos. Posteriormente, se aplicó la función *sort_values* para ordenar los valores de forma descendente. Finalmente, se utilizó la operación *head(5)* para seleccionar únicamente los cinco valores más frecuentes.

Dentro de los análisis realizados destacan los siguientes:

- **Tarjeta.** Los 5 pasajeros que más han utilizado el metro durante el mes de Enero son los siguientes:
  - 3E0410513BF6580F1E8ED912B6314AF7 con 2.019.545 viajes
  - 8CA04B9A8A7C315A38A1707EFF2F585D con 1.533 viajes
  - 0E0AE809A93FB2FF6AFBA13C31735B7F con 1.299 viajes
  - E49759E7547150ECDF57F186594D8BD4 con 1.042 viajes
  - 09E652F6CF4DC3300F4263CE7AF57E13 con 936 viajes

  Las cadenas de texto mostradas corresponden a identificadores de tarjeta de viaje anonimizados.

  A partir de sus frecuencias de apariciones, se puede inferir que el primer usuario ha realizado una cantidad significativa de viajes, ya que su tarjeta está posiblemente asociada al colectivo de trabajadores de limpieza del metro. Es probable que este usuario o usuarios accedan regularmente a las instalaciones, lo que explicaría la frecuencia de sus viajes.

  Por otra parte, el resto de la información se podría utilizar para establecer algún tipo de descuento a mayores sobre aquellos pasajeros que hacen mayor uso de este servicio público.

- **Tipo de usuario.** El gráfico de la Figura4.9, realizado mediante la función *countplot()* de Seaborn, revela información sobre los tipos de usuarios más frecuentes en el transporte público del municipio de Madrid. Los resultados obtenidos son los siguientes:
  - El tipo de usuario '01' cuenta con 88.275.023 apariciones, lo que representa un 58,42% del total. Este tipo de usuario corresponde a la categoría ***Normal***.
  - El tipo de usuario '03' cuenta con 40.929.099 apariciones, equivalente al 27,09% del total. Este tipo de usuario se asocia a la categoría ***Joven***.

- El tipo de usuario '02' cuenta con 17.531.120 apariciones, lo que representa el 11,60% del total. Este tipo de usuario corresponde a la categoría ***3º edad***.

- El tipo de usuario '06' cuenta con 2.528.655 apariciones, lo que equivale al 1,67% del total. Este tipo de usuario se clasifica como ***Tarjeta Azul*** .

- El tipo de usuario '04' cuenta con 1.189.507 apariciones, lo que representa el 0,79% del total. Este tipo de usuario se refiere a la categoría ***Infantil***.

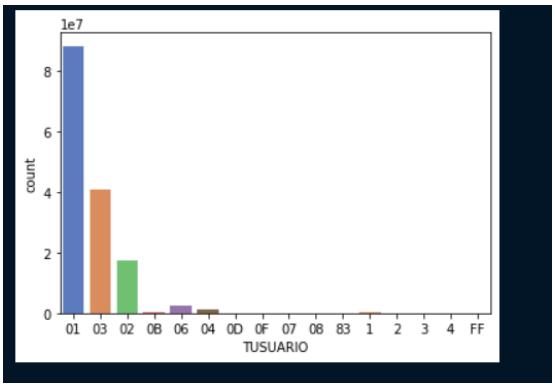


Figura 4.9: Análisis de la frecuencia de cada tipo de usuarios

- **Código de validez**

  Los resultados obtenidos son los siguientes:

  - Categoría '1': Esta categoría tiene 67.110.199 apariciones, lo que representa un 44,42% del total. Se refiere a una categoría que implica una validación completa con una tarjeta temporal con descuento de viaje.

  - Categoría '2': Se registra en 55.303.758 apariciones, lo que representa un 36,60% del total. Esta categoría es una variante de la categoría '1' y está relacionada con el aviso de que el abono está a punto de caducar.

  - Categoría '9': Se encuentra en 9.599.385 apariciones, lo que equivale al 6,35% del total. Esta categoría implica una validación completa con una tarjeta temporal sin descuento de viaje al salir de las instalaciones, excluyendo los autobuses.

  - Categoría '4': Tiene 4.444.294 apariciones, lo que representa un 2,94% del total. Esta categoría indica una validación completa con descuento de viaje (con multiviaje) al subir a un medio de transporte, cuando el viajero se desplaza desde o hacia una zona tarifaria no incluida en su abono.

  - Categoría '44': Registra 3.254.560 apariciones, lo que equivale al 2,15% del total. Esta categoría implica una validación completa al salir de las instalaciones con una tarjeta temporal sin descuento de viaje, y con origen y destino dentro del mismo ámbito.

- **IDTLV**

A partir del Figura 4.10, generado con el paquete *Seaborn*, pudimos obtener las siguientes conclusiones:

- El tipo de máquina de validación de tarjeta más utilizado es el correspondiente con las máquinas sistema de metro. Su uso representa un 61,2% del total.
- Las máquinas de validación presentes en los autobuses son las menos usadas con un 0,57% del total.

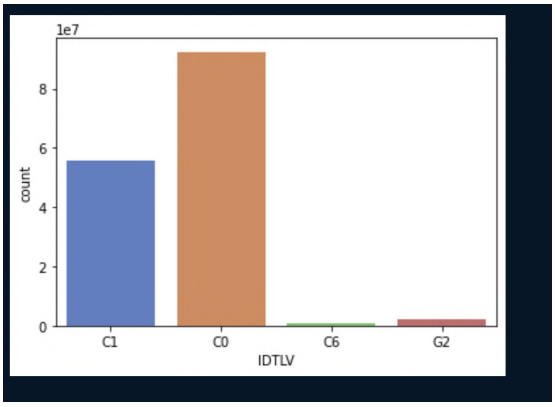


Figura 4.10: Análisis de la frecuencia del uso de los diferentes tipo de máquinas de acceso al servicio de transporte

4.2.5 Análisis con subconjunto aleatorio

Esta última sección del análisis exploratorio se destinó a hacer un análisis más exhaustivo de los datos a través de gráficos que muestran la combinación de diversas variables del conjunto. Entre estas variables destaca el uso de la variable *fecha* con el objetivo de hacer filtrados teniendo en cuenta la componente temporal. Para hacer este análisis se utilizó un conjunto reducido del dataset por dos motivos principales:

- Mejorar la eficiencia y rapidez del procesamiento para la visualización de los gráficos, lo que permitirá una mayor capacidad para efectuar distintas pruebas sobre los datos de manera ágil.

- Permitir el uso de la componte temporal. Como se había mencionado en la sección 4.2.3, debido a las altas necesidades computacionales, requeridas para cambiar el tipo de datos del campo *Fecha* a tipo *datetime* sobre volúmenes muy grandes de datos, el proceso no se pudo completar. Sin embargo, al usar un subconjunto más pequeño de datos, esta operación sí se pudo realizar de manera correcta.

Para realizar este análisis, se empleó un conjunto reducido de los datos totales. Se realizó un muestreo al azar de 400.000 filas para los bloques 1 y 2, y otro de 200.000 filas para el bloque 3. El tamaño de estos subconjuntos se decidió estableciendo una proporcionalidad entre el tamaño del bloque y el tamaño del dataset total. Después, se realizó una concatenación de los tres subconjuntos y una selección de una muestra aleatoria de 500.000 filas del mismo del bloque resultante del proceso de concatenado. Este último subconjunto fue sobre el que se realizó el análisis.

A continuación se muestra el código relativo a este proceso:

```
# tamaño de los subconjuntos
subset_size_1_2 = 400000
subset_size_3=200000

# Muestra aleatoria
subset_df1 = df.sample(n=subset_size_1_2, random_state=33)
subset_df2 = df2.sample(n=subset_size_1_2, random_state=33)
subset_df3 = df3.sample(n=subset_size_3, random_state=33)

#Concatenado
df_concat = pd.concat([subset_df1, subset_df2,subset_df3])

# Muestra aleatoria sobre el concatenado
subset_size=500000
subset_df = df_concat.sample(n=subset_size, random_state=33)
```

Como el conjunto de datos ya provenía del paso de limpieza, el único preprocesado que se le aplicó a mayores fue cambiar el tipo de datos del campo Fecha. Una vez completado este paso se comenzó con la realización de los gráficos. Estos gráficos se generaron usando *Seaborn* y el módulo pyplot de *Matplolib*. Los gráficos más destacados obtenidos se muestran a continuación.

Cantidad de viajes y pasajeros según día de la semana

En ambas gráficas de la Figura 4.11 se puede observar que los días con mayor afluencia en los servicios de transporte público del servicio madrileño son los miércoles y jueves, días que se encuentran en plena mitad de la semana laboral o estudiantil [33].

A su vez, como se podría esperar, el día de la semana con menor cantidad de uso del transporte público es el domingo, día no lectivo para la mayor parte de españoles.

(a) Distribución de los viajes según el día de la semana

(b) Distribución de pasajeros por día de la semana

Figura 4.11: Visualización del total de viajes y pasajeros ordenado por el día de la semana

Cantidad de viajes según la hora

En el gráfico de la Figura 4.12 refleja la cantidad de viajes realizados en las distintas horas del día [34]. Con una simple observación se puede conocer que los tramos horarios con más trayectos realizados son, por orden, las 8 de la mañana, las 2 del mediodía y las 6 de tarde. La lógica nos indica que el primer puesto de este *ranking* es motivado por el comienzo de la jornada laboral y académica, mientras que los otros dos horarios de mayor afluencia, como consecuencia, coinciden con la finalización de ambas jornadas respectivamente.

Por otra parte las horas con afluencia casi inexistente son las 3 y las 4 de la madrugada. Esto se debe a que en estos horarios, gran parte de los servicios públicos de transporte permanecen cerrados, además de que gran parte de la población se estima que puede estar durmiendo.

Figura 4.12: Distribución de los viajes según la hora del día

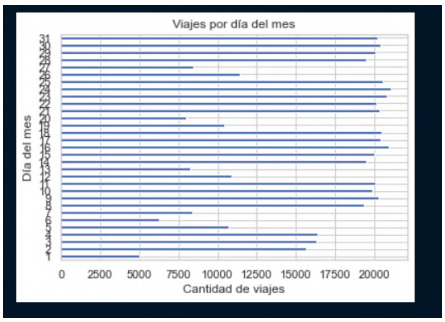
Cantidad de viajes según día del mes

Debido a que solo tratamos con los datos del mes de enero de 2019, mostramos gráficas únicamente de dicho mes, cuyo calendario se puede ver en la imagen 4.14.

En las imágenes de la figura 4.13 se puede ver una distribución que sigue un patrón regular y lógico mostrando menor afluencia los fines de semana y un pico de uso los días centrales de la semana [35].

Dentro de los días con mayor cantidad de viajes destacamos los días 24 y 26 del mes. Sin

embargo, aquel día con menos cantidad de viajes es el 1. Esto se puede deber a que es el primer día del año, y puede que los pasajeros tiendan a usar menos transporte o que, incluso, haya algún medio de transporte público que no esté en funcionamiento en dicho día.



(a) Distribución de los viajes según el día del mes

(b) Tendencia temporal de los viaje según el día del mes

Figura 4.13: Visualización del total de viajes ordenado por el día del mes

Figura 4.14: Calendario Enero 2019
[36]

Proporción de tipo usuarios de usuarios los fines de semana

El gráfico 4.15 se centra en la distribución de los distintos tipo de usuarios de transporte público durante los sábados y domingos. Solo tenemos en cuenta los 5 más frecuentes.

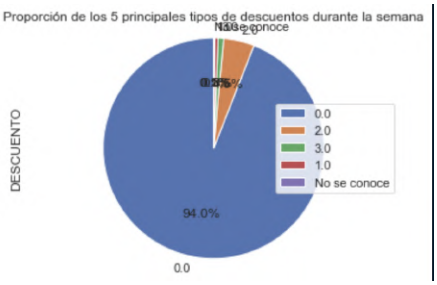
A través de esta visualización, podemos observar que la proporción durante el fin de semana es muy similar a la proporción del análisis de frecuencias generales mostrado en el apartado 4.2.4. No existen variaciones notables.

Figura 4.15: Distribución de tipo de usuarios organizados por día de la semana

Comparación de tipos de descuento entre días laborables y no laborables

En la gráficas de la figura 4.16, se muestran los 5 descuentos más aplicados a los usuarios en sus viajes realizados durante la semana, separándolos en dos grandes bloques: días lectivos y días no lectivos.

Gracias a esta visualización, se puede concluir que el tipo de descuento que predomina en ambos grupos es el '0.0', que significa la no aplicación de ningún descuento sobre el viaje.



(a) Proporción de descuentos en días laborables

(b) Proporción de descuentos en días no laborable

Figura 4.16: Porcentaje de descuentos según tramos de la semana

4.3 Análisis de Requisitos

Este trabajo se centra en la gestión del sistema de transporte público del municipio de Madrid. Con ese objetivo, se realizó un análisis, diseño e implementación de una aplicación web que nos da como resultado las funciones y requisitos comentados en la sección 4.3.2. Basándose en las mismas, los responsables del Consorcio de Transporte de Madrid, estarán en mejor disposición para adoptar las medidas más adecuadas de cara mejorar el servicio. De manera indirecta, esto ayudará a incrementar el uso del transporte público en la ciudad.

4.3.1 Actores

Las capacidades y características de cada uno de los roles dentro de la aplicación web se definen y detallan en esta sección.

- **Usuario público:** Este usuario hace referencia a los pasajeros del transporte público del municipio. La funcionalidades asociadas a este tipo de usuario son las relativas al uso de la primera, la segunda y la cuarta página de la aplicación web.

  La primera de ellas es la página de bienvenida que muestra una breve presentación permitiendo al usuario conocer el objetivo de la aplicación.

La segunda página le permite al usuario consultar su propio historial de viajes simplemente introduciendo el identificador relativo a su tarjeta de viaje. A su vez, el usuario tiene la posibilidad filtrar los resultado por fecha. En cuanto al resto de páginas de la aplicación, este usuario solo tendrá permisos de visualización.

Por último, la cuarta, y última, página de la aplicación le permite al usuario tratar con un mapa del municipio de Madrid y conocer información sobre las paradas y líneas cercanas a cualquier punto que el usuario seleccione sobre el mapa. Proporcionará al usuario mayor conocimiento sobre la red de transporte del municipio.

- **Usuario consorcio:** Este usuario es el adecuado para aquellos trabajadores del consorcio. Además de las funcionalidades mencionadas en el caso del usuario público, este usuario tendrá acceso al uso de las tercera página de la web.

  Dicha página le permitirá tratar con los datos de los viajes de todos los usuarios del transporte del municipio, además de los datos de la topología de la red de manera interactiva. Podrá usar esa página para sacar conclusiones informadas sobre el uso del servicio, las cuales podrán ser útiles para tomar decisiones orientadas a la mejora del mismo.

- **Administrador**: Por último, este rol, a mayores de todas las funcionalidades anteriores, tiene acceso a la página de administración de la aplicación.

  Esta interfaz permite al administrador gestionar usuarios y grupos de usuarios presentes en la aplicación, además de la administración y edición del conjunto de datos cargados en el sistema de gestión de bases de datos empleado en el proyecto.

4.3.2 Requisitos funcionales

En esta sección, se recogen los requisitos funcionales de la aplicación, los cuales consisten en aquellas funciones o servicios que la aplicación web pone a disposición del usuario. Todas estas características están recogidas en la tabla 4.1.

Cuadro 4.1: Tabla de historias de usuario de la aplicación web

ID	Nombre	Descripción
HU-01	Conocer historial de viajes (a)	Obtener todo el historial de viajes de un pasajero.

Cuadro 4.1 – *(continúa en la página siguiente)*

ID	Nombre	Descripción
HU-02	Conocer historial de viajes (b)	Obtener el historial de viajes de un pasajero permitiendo filtrar por intervalo de días.
HU-03	Información detallada de las paradas	Visualizar información sobre el tipo de transporte, número de línea y nombre de parada en la que el pasajero se subió a un vehículo de transporte público.
HU-04	Contador de subidas	Conocer la cantidad de veces que cada usuario subió a cada parada.
HU-05	Contador de pasajeros	Obtener la cantidad de pasajeros que usan el transporte público. Acción solamente permitida para administradores y usuarios del consorcio.
HU-06	Contador de viajes	Visualizar la cantidad de viajes que se realizan en el transporte público. Acción solamente permitida para administradores y usuarios del consorcio.
HU-07	Flitrado por tipo de transporte	Obtener toda la información sobre viajes de los usuarios y topología de la red filtrada por el medio que utilicen. Acción solamente permitida para administradores y usuarios del consorcio.
HU-08	Flitrado por linea	Obtener toda la información sobre viajes de los usuarios y topología de la red filtrada por la línea que utilicen. Acción solamente permitida para administradores y usuarios del consorcio.
HU-09	Consultar afluencia en días festivos	Visualizar la cantidad de pasajeros y viajes los días no laborables. Acción solamente permitida para administradores y usuarios del consorcio.
HU-10	Consultar paradas más transitadas	Descubrir las 5 paradas con más afluencia de pasajeros. Acción solamente permitida para administradores y usuarios del consorcio.

Cuadro 4.1 – *(continúa en la página siguiente)*

ID	Nombre	Descripción
HU-11	Flitrado por franja horaria	Conocer toda la información sobre viajes de los usuarios y topología de la red filtrada por el medio que utilicen. Acción solamente permitida para administradores y usuarios del consorcio.
HU-12	Flitrado por intervalo temporal	Obtener toda la información sobre viajes de los usuarios y topología de la red filtrada por el día o la franja de días en los que se llevaron a cabo los trayectos. Acción solamente permitida para administradores y usuarios del consorcio.
HU-13	Consultar información por tipo de usuario	Obtener toda la información sobre viajes de los usuarios y topología de la red filtrada por el tipo de usuario que viaje. Acción solamente permitida para administradores y usuarios del consorcio.
HU-14	Visualizar mapa de calor	Observar la densidad de viajes en diferentes áreas de Madrid. Acción solamente permitida para administradores y usuarios del consorcio.
HU-15	Comparar la afluencia en días laborables y no laborables	Realizar una comparación entre la cantidad de pasajeros en días lectivos y no lectivos. Acción solamente permitida para administradores y usuarios del consorcio.
HU-16	Comparar la afluencia en días de la semana	Realizar una comparación entre la cantidad de pasajeros y cantidad en los diferentes días de la semana. Acción solamente permitida para administradores y usuarios del consorcio.
HU-17	Exportar datos	Descargar los datos, en forma de tabla, de los elementos visuales del panel de control o *dashboard*. Acción solamente permitida para administradores y usuarios del consorcio.

Cuadro 4.1 – *(continúa en la página siguiente)*

ID	Nombre	Descripción
HU-18	Marcar ubicación sobre el mapa	Seleccionar una ubicación exacta en el mapa sobre el que se efectuarán diversas acciones.
HU-19	Mostrar paradas cercanas	Obtener marcadores en mapa que muestren las paradas cercanas al punto seleccionado por el usuario.
HU-20	Mostrar el recorrido de la línea	Visualizar el recorrido de la línea de transporte público correspondiente a la parada seleccionada por el usuario.
HU-21	Autenticación	Proceso de autenticación en el panel de administración de la aplicación. Acción disponible solo para administradores.
HU-22	Crear usuarios	Añadir nuevos usuarios al sistema. Acción disponible solo para administradores.
HU-23	Eliminar usuarios	Suprimir usuarios del sistema. Acción disponible solo para administradores.
HU-24	Crear grupo de usuarios	Generar grupos de usuarios en el sistema. Acción disponible solo para administradores.
HU-25	Eliminar grupo de usuarios	Borrar grupos de usuarios del sistema. Acción disponible solo para administradores.
HU-26	Asignar permisos	Otorgar permisos y configuraciones específicas a usuarios individuales o grupos de usuarios del sistema. Acción disponible solo para administradores.
HU-27	Gestionar permisos y roles	Administrar permisos y roles sobre usuarios individuales o grupos de usuarios del sistema. Acción disponible solo para administradores.

Cuadro 4.1 – *(continúa en la página siguiente)*

ID	Nombre	Descripción
HU-28	Administrar bases de datos	Gestionar las bases de datos cargadas en el sistema de administración. Acción disponible solo para administradores.
HU-29	Ver registros de actividad	Visualizar los registros de actividad de los usuarios de la página de administración. Acción disponible solo para administradores.

4.3.3 Requisitos no funcionales

Los requisitos no funcionales, son los encargados de definir las características o propiedades que debe tener el sistema en cuanto a aspectos como la seguridad y el almacenamiento, entre otros.

Los principales requisitos no funcionales del proyecto son los siguientes:

- Diseño de una interfaz fácil de usar e intuitiva para la aplicación web. Esto puede ayudar a favorecer la compresión por parte de todo tipo de usuarios.
- Las consultas realizadas en la aplicación deben ejecutarse de manera eficiente, reduciendo, en la medida de lo posible, los tiempos de espera en las ejecuciones de la plataforma. Para lograr este objetivo, una estrategia muy importante es el uso de índices.
- La escalabilidad de la aplicación debe estar garantizada, soportando cada vez bases de datos más grandes y operaciones y consultas de una complejidad computacional superior.

4.4 Arquitectura del sistema

El sistema se fundamenta en una arquitectura de tres capas, que se caracteriza por ser lineal y tener una jerarquía estricta, donde solo se tiene conocimiento de la capa inmediatamente inferior. La interacción entre estas capas generalmente se realiza de manera asíncrona, lo que permite una mayor escalabilidad del sistema. Esto se puede observar en la figura 4.17.

1. **Capa de base de datos.** Esta capa funciona como un repositorio de datos en la plataforma. Incluye un sistema de gestión de bases de datos que está exclusivamente conectado al controlador. Su objetivo principal es almacenar y recuperar los datos del controlador mediante esta conexión establecida.

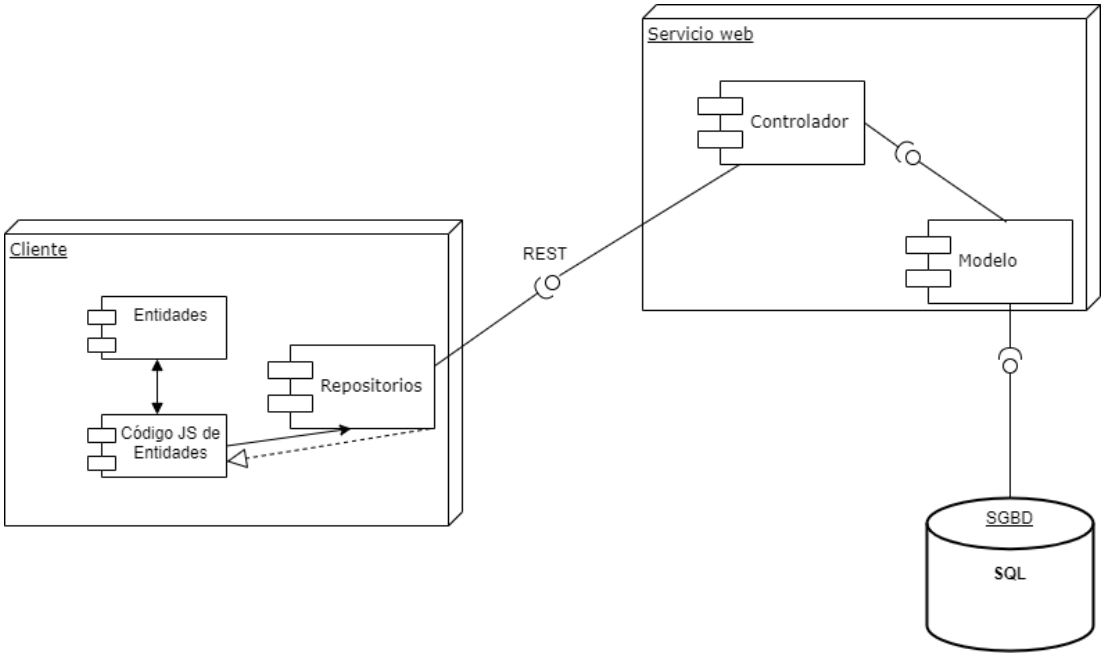


Figura 4.17: Diagrama de componentes general de la arquitectura del sistema

2. **Capa de servicio web.**

   (a) **Modelo.** El modelo representa la capa de acceso a los datos y la lógica de negocio. En esta capa se definen las clases y estructuras que interactúan con la base de datos. Además, se llevan a cabo operaciones tanto de lectura como de escritura. Por otra parte, el modelo también se encarga de servir los datos obtenidos al controlador.

   (b) **Controlador.** En esta parte de la capa de servicio web se define la lógica de control del sistema. El controlador se encarga de recibir las solicitudes del cliente, y las redirige a un lugar apropiado para su procesamiento. Se utilizan tecnologías y servicios *RESTful* para acceder a los datos almacenados en los repositorios.

3. **Capa de cliente**

   (a) **Repositorio.** Habilita el acceso a los datos a nivel de cliente. Se establecen métodos y funciones con el objetivo de obtener y enviar datos mediante una comunicación constante con el servidor, más concretamente con el controlador. Entre sus funcionalidades, destaca la inclusión de métodos para la realización de solicitudes *HTTP* (Hypertext Transfer Protocol) al servidor, y su posterior manejo de la respuesta obtenida.

   (b) **Entidades.** Se trata de las plantillas que se sirven para mostrar la información de

cara al usuario final de la aplicación web. Este conjunto de plantillas también se pueden denominar interfaces de usuario.

(c) **Código JS de Entities.** Esta parte de la capa de cliente contiene la implementación en JavaScript de la lógica asociada a las entidades. Poseen todo el código necesario para mostrar las información en las plantillas o *templates* de la aplicación.

4.5 Interfaz de usuario

Durante la fase preliminar del proyecto, una de las principales actividades llevadas a cabo consistió en el diseño de una serie de interfaces de gráficas para la aplicación web. El propósito fundamental de estos diseños fue proporcionar una idea general de la apariencia y disposición de los elementos de la aplicación web. No obstante, a medida que el proyecto avanzaba, se realizaron ciertos ajustes y mejoras de las interfaces, siguiendo la metodología incremental descrita en la sección 3.1.

La primera de las interfaces desarrolladas fue la relativa a la página de inicio de la aplicación. La idea considerada para esta página, consistió en la introducción de una imagen relacionada con algún medio de transporte del municipio madrileño, y la redacción de un breve texto en el cual se describiría, de manera sencilla, la funcionalidad de la aplicación web como se observa en la Figura 4.18. Esta página no está pensada para la realización de ninguna consulta, solamente tiene funcionalidad informativa.

La página relativa al historial de viajes, tenía como objetivo principal brindar al usuario la posibilidad de consultar los trayectos que ha realizado utilizando el sistema de transporte público de Madrid. Con el fin de permitir esta operación, se mostraba un caja de texto donde el usuario podía introducir el identificador de su tarjeta de viaje, además de un botón a su derecha que permite ejecutar la búsqueda y obtención de los desplazamientos del pasajero en cuestión. Esto se puede ver en la Figura A.1. Una vez realizada la operación por parte del usuario, este podía ver la tabla con los viajes llevados a cabo, ordenados según la fecha de realización de los mismos, mostrando en primer lugar aquellos más recientes. En caso de realización de otra consulta, como la introducción de un identificador de tarjeta diferente, el usuario debería retroceder a la página anterior a través de un botón con el mensaje 'Volver', como en la Figura 4.19. Sin embargo, finalmente esta página experimentó varias modificaciones. Dentro de ellas destaca la inclusión de un filtro temporal y la presentación de un mapa en el cual se indica, a través de marcadores, las paradas en las que ha subido el usuario durante ese intervalo temporal.

La siguiente página de la aplicación, mostrada en la Figura 4.20, tenía como propósito principal brindar un panel de control intuitivo sobre la información de los viajes de los usuarios y de la topología de la red, que permitiría a los trabajadores del consorcio de transporte

de Madrid tomar decisiones informadas para la mejora del sistema público de movilidad. Para ello, se usaría una plataforma orientada a la visualización de datos. Entre las vistas existentes, estaría la obtención de la información relativa a la afluencia de pasajeros y cantidad de viajes según ciertos filtros como tipo de transporte, franja horaria o línea, entre otros. Otra vista planteada, fue la visualización de un mapa de calor del municipio de Madrid, el cual permitiese visualizar las zonas de mayor y menor afluencia de pasajeros en la capital. A medida que se fue avanzando en el proyecto, se fueron incluyendo más vistas de interés.

Como última página, se propuso la creación de un mapa interactivo que permitiese al usuario adquirir conocimiento sobre la topología de la red de transporte. Esta página posibilitaría al usuario seleccionar un punto en el mapa como se puede observar en la Figura A.2. Posteriormente, se visualizarían las paradas cercanas al punto escogido, tal como se muestra en la Figura A.3. La principal ampliación que se realizó sobre esta página, fue brindar la opción de observar la ruta de las distintas líneas de transporte sobre el mapa.

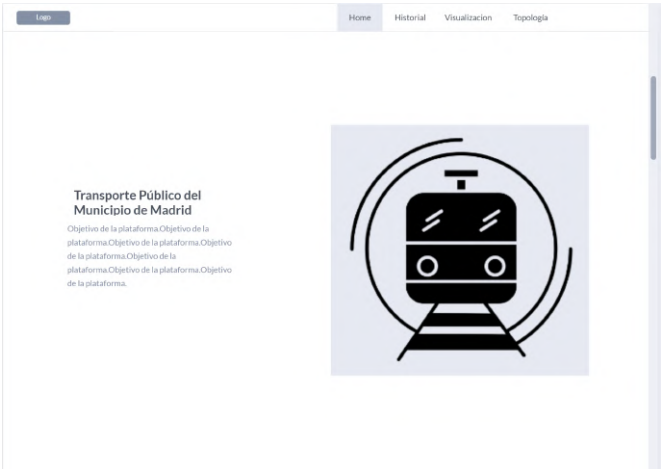


Figura 4.18: Mockup de la pantalla de inicio de la aplicación

4.6 Modelo conceptual de datos

Como se mencionó en el análisis preliminar de los datos en la sección 4.1, se realizó,en primera instancia, un diagrama entidad-relación sobre los datos del GTFS del metro de Madrid con el objetivo de comenzar a conocer los datos con los que se podrían trabajar durante la realización del proyecto. Sin embargo, finalmente el estudio se amplió a todos los medios de transporte del municipio.

Con el fin de lograr una mayor simplicidad y eficiencia, se recopilaron todos los datos relacionados con la topología de la red de transportes públicos de Madrid en una única base de datos. Esta base de datos se integró con la de viajes, con el propósito de aprovechar la

Figura 4.19: Mockup de la consulta de historial de pasajeros

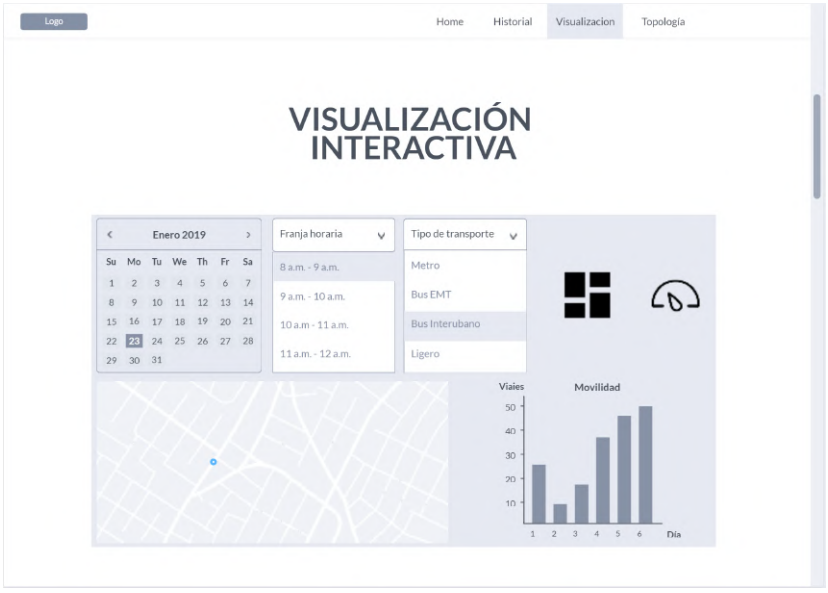


Figura 4.20: Mockup de la página de visualización interactiva de los datos

información de uso del transporte y de la topología, para obtener conclusiones que pudieran aportar valor sobre el movimiento alrededor de toda la geografía de la capital.

A continuación, se muestra el modelado conceptual definitivo en el que se basó el proyecto. Por una parte, se encuentra el modelo Entidad-Relación, en la Figura 4.21, mientras que por otro lado encontramos el diseño lógico del modelo conceptual, a través de la Figura 4.22. Como se puede observar, este modelado conceptual es una simplificación del realizado en la etapa preliminar. Este hecho se debe a que la única información necesaria para el proyecto es la relacionada con los 'clicks' de los pasajeros, en su acceso a los medios de transporte, además de información básica acerca de la topología de la red.

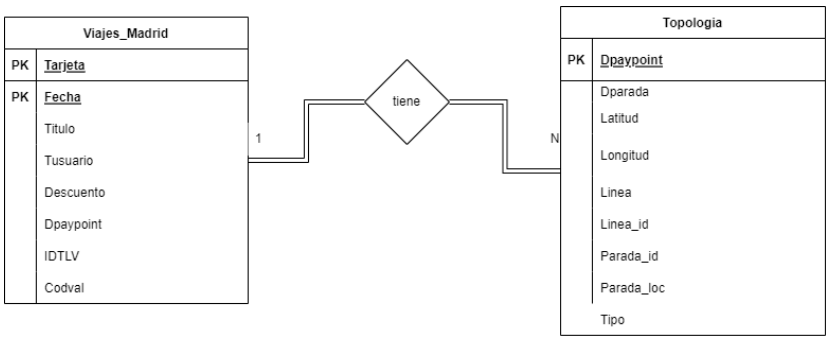


Figura 4.21: Modelo entidad-relación definitivo

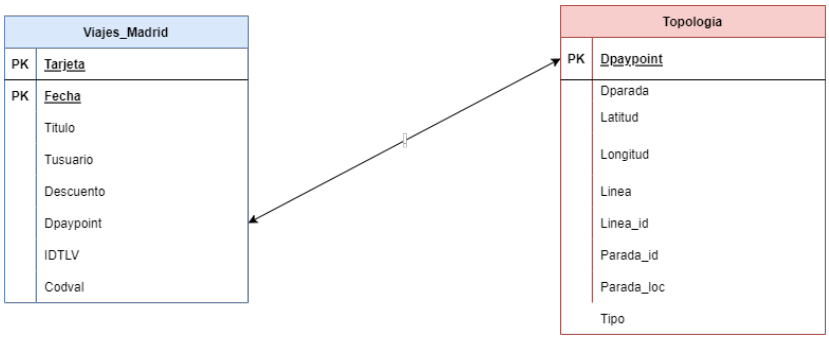


Figura 4.22: Diseño lógico definitivo

Topologías

Conjunto de datos que resumen toda la información relativa a la red de trasporte urbano de la capital española.

- *Dparada*: Nombre de la parada de transporte.
- *Dpaypoint*: Punto de pago asociado a la parada.
- *Tipo*: Tipo de transporte (cercanías, metro, autobus interurbano, etc.).

- *Linea*: Número o código de la línea de transporte.
- *Linea_id*: Identificador único de la línea.
- *Parada_id*: Identificador único de la parada.
- *Parada_loc*: Ubicación de la parada en formato geoespacial.
- *Latitud*: Valor de la latitud geoespacial de la parada.
- *Longitud*: Valor de la longitud geoespacial de la parada.

4.7 API REST

Una API consiste en una serie de reglas y protocolos enfocados a la comunicación entre sistemas informáticos. Permite compartir información de manera estructurada entre aplicaciones.

Los endpoints son URLs o puntos de acceso del servicio de una API que responden a las solicitudes. En el caso específico de nuestro proyecto, estos endpoints se encuentran definidos en el archivo urls.py generado por Django.

En esta sección se proporciona una documentación detallada de los endpoints desarrollados para implementar diversas funcionalidades en la aplicación web. Además, se incluyen los métodos correspondientes que se encargan de manejar estos endpoints.

Cuadro 4.2: EndPoints de la página de administración

Petición	**EndPoint**	**Acción**
GET	/admin/*func*/*Database*	Acceso a base de datos de la función propagada desde en Django.
GET	/admin/login	Acceder al portal de autenticación.
POST	/admin/login	Autenticarse.
POST	/admin/logout	Salir del portal de autenticación.

. . . . . *(continúa en la página siguiente)* . . . . . .

Cuadro 4.2 – *(continúa en la página siguiente)*

Petición	EndPoint	Acción
GET	/admin	Acceso al portal de administración.
GET	/admin/password_change	Acesso a página de cambio de contraseña.
GET	/admin/auth/user	Acceso panel de administración de usuarios.
GET	/admin/auth/user/add	Acesso al panel de añadir usuario.
POST	/admin/auth/user/add	Añadir usuario.
GET	/admin/auth/group	Acceso panel de administración de grupo de usuarios.
GET	/admin/auth/group/add	Acesso al panel de añadir grupo de usuarios usuario.
POST	/admin/auth/group/add	Añadir grupo de usuarios.
GET	/admin/*func*/*Database*	Acceso a base de datos de la función propagada desde en Django.
GET	/admin/*func*/*Database*/obj/change	Acceso a panel de modificación de un objeto de la base de datos.
POST	/admin/*func*/*Database*/obj/change	Modificar un objeto de la base de datos.
GET	/admin/*func*/*Database*/obj/change/h	Acceso al historial de cambios del objeto.
GET	/admin/*func*/*Database*/add	Acceso a panel para añadir un objeto nuevo.

. . . . . *(continúa en la página siguiente)* . . . . . .

Cuadro 4.2 – *(continúa en la página siguiente)*

Petición	EndPoint	Acción
POST	/admin/*func*/*Database*/add	Añadir un objeto nuevo.

Petición	EndPoint	Acción
GET	/	Acceso a la página de inicio.
GET	/visual	Acceso a la página de visualización de datos.
GET	/busqueda	Acceso a la página de historial de usuarios.
GET	/buscar	Obtención del historial de viaje.
GET	/topologia	Acceso a la página de topología de la red.
POST	/topologia	Selección de un punto en el mapa.
POST	/lineas	Obtención de la ruta de la parada seleccionada.

Cuadro 4.3: Endpoints de las páginas de inicio, historial de viajes, visualización interactiva y topología de la red.

Capítulo 5

Diseño

En este capítulo se describe la arquitectura del sistema, mostrada en la sección 4.4, haciendo hincapié en las tecnologías empleadas en cada capa de la misma. Por otra parte, se realiza un desglose de los diferentes componentes del sistema que fueron plasmados en la Figura 4.17.

5.1 Arquitectura tecnológica del sistema

En primer lugar, para la implementación de la **base de datos** se utilizó PostgreSQL. Gracias al uso de este sistema de gestión de bases de datos, se pudo llevar a cabo la carga eficiente de los mismos, además del establecimiento de las relaciones necesarias entre las diferentes tablas definidas en el modelo de datos del proyecto. Además, es necesario destacar el uso del la extensión PostGIS que se encarga de soportar el almacenamiento y las operaciones empleando información geográfica.

En cuanto a la capa del **servidor web**, se empleó el framework Django. Este entorno de desarrollo otorgó la posibilidad de llevar a cabo dos funcionalidades esenciales en el sistema:

- Creación y configuración del **modelo** de datos establecido para el proyecto. Para permitir esto, Django tiene el archivo *models.py*. Este fichero permite definir las clases que representan las tablas de la base de datos y sus campos correspondientes.

- Procesamiento de las solicitudes del cliente a modo de **controlador** del sistema. Estas operaciones se implementan a través de las vistas definidas en el archivo *views.py*, las cuales reciben las solicitudes del cliente y poseen la lógica de negocio adecuada para procesarlas.

A continuación, se encuentra la capa de **cliente web**. En ella se puede observar de nuevo el uso de Django a modo de **repositorio**, ya que se encarga de comunicarse con la base de datos mediante la conexión con el controlador. También, se empleó HTML, CSS, Leaflet,

Boostrap y PowerBI para la visualización y presentanción de las entidades propuestas a través del desarrollo de una serie de *templates* o plantillas. Finalmente, se usa JavaScript para el desarrollo del código de la entidades anteriormente mencionadas.

Todo la información previamente mencionada se plasma en la Figura5.1.

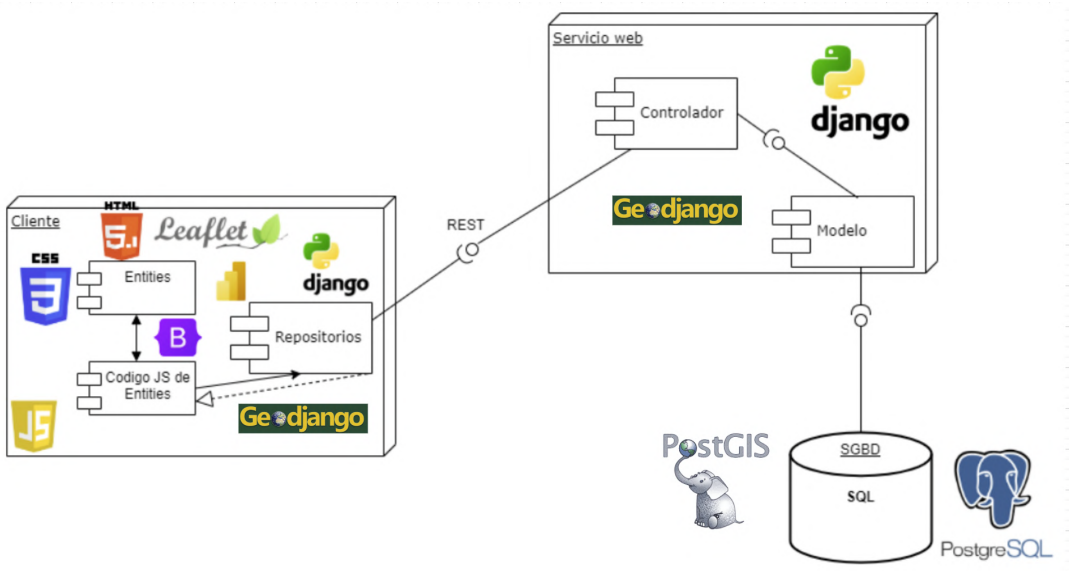


Figura 5.1: Diagrama de componentes de la arquitectura tecnológica del sistema

5.2 Diseño de la aplicación

Esta sección se centrará en mostrar aquellas aplicaciones, archivos o ficheros que permiten la implementación de cada uno de los componentes del modelo del sistema definido en la sección 4.4.

5.2.1 Servidor web

Como se comentó en la sección de arquitectura del sistema, el servidor web se divide en dos módulos diferenciados en los cuales se emplea Django [37]:

- **Modelos.** Para la definición de los modelos, Django utiliza un archivo llamado *models.py*. En este fichero se utlizan clases de Python para definir las tablas de las bases de datos. Los atributos de cada clase representan los campos de la tabla. Una vez generadas todas las clases, estas se propagan de cara a la base de datos usando las operaciones *makemigrations* y *migrate* descritas en la sección 6.1.1. Un ejemplo de definición de una clase en el fichero *models.py* se puede observar en la Figura 6.1.

- **Controlador**. En Django, la función de controlador se implementa a través del archivo *views.py*. En este archivo se definen una serie de funciones, las cuales se encargan de manejar y responder solicitudes entrantes. Estas vistas se mapean a una URL específica que se introduce en el archivo *urls.py*. Este archivo únicamente se encarga de definir esas rutas, y especificar a qué vista llamar cuando se accede a cada una de ellas.

5.2.2 Cliente web

- **Repositorio.** El papel del repositorio se lleva a cabo a través de la conexión entre las plantillas html (entities) y las vistas del archivo *views.py*. En este proyecto, estas conexiones se establecen a través de consultas HTTP usando la biblioteca JQuery. Esta biblioteca de JavaScript proporciona usa serie de métodos específicos para realizar solicitudes al servidor. Los métodos usados en nuestro proyecto son *$.get()* y *$.post()* , que realizan consultas GET y POST al servidor respectivamente.

- **Entities y Código JS de las Entities**. Las entities son una serie de plantillas que permiten definir la interfaz final que se le va a presentar al usuario por pantalla. Estas plantillas se encuentran en formato HTML y utilizan código en lenguaje JavaScript (JS) para llevar a cabo las operaciones deseadas. Se suelen almacenar en una carpeta llamada *templates*. Se hace uso de ciertas librerías para mejorar el diseño de la pantalla final como Bootstrap o CSS. La serie de plantillas utilizadas en proyecto se pueden observar en la Figura 5.2.

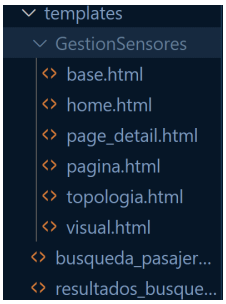


Figura 5.2: Lista de entities creadas en el desarrollo del proyecto

Capítulo 6

Implementación y pruebas

En este capítulo se aborda el proceso de importación y limpieza de datos en diferentes software utilizados en el proyecto. Además, se presentan ciertos algoritmos complejos implementados y se detallan las pruebas realizadas para su desarrollo final.

6.1 Implementación

6.1.1 Importación de datos

El proceso de importación de datos realizado durante el proyecto se divide en tres bloques bien diferenciados: importación en software de analítica de datos, importación en software de visualización de datos e importación en sistema gestor de base de datos.

En el primer caso, se aborda la importación de la base de datos de viajes en los softwares RStudio y VSCode (Python). Como se mencionó en la sección 4.2, se decidió crear un script para dividir los datos en tres bloques más pequeños debido a su gran tamaño. Una vez completado este proceso, se utilizaron las funciones "pd.read_csv" en VSCode y "read_csv" en RStudio para importar los datos.

En el segundo caso, se enfocó en la carga de datos de viajes y la topología de la red en el software de visualización PowerBI. Para llevar a cabo este proceso, se siguió el siguiente procedimiento:

1. En PowerBI, se hizo clic en la pestaña "Inicio" y se seleccionó la operación "Obtener Datos".

2. Apareció un desplegable con diversas opciones de tipo de datos para añadir. Se eligió la opción "Texto o CSV".

3. Finalmente, se abrió una pestaña en PowerBI que mostraba las tablas de datos importadas. Se seleccionó la opción "Cargar Datos" para completar el proceso de importación.

```
C:\Users\34610\Documents\TFG\ProyectoDjango>python -m django startproject MetroMadrid

C:\Users\34610\Documents\TFG\ProyectoDjango>cd MetroMadrid

C:\Users\34610\Documents\TFG\ProyectoDjango\MetroMadrid>python manage.py startapp GestionSensores
```

Figura 6.1: Creación del proyecto en Django.

Por último, se realizó la carga de datos en PostgreSQL. Para ello, en primera instancia se creó tanto el proyecto como la aplicación específica en Django que se usaría para desarrollar nuestra aplicación web. Esto se hizo, tras la instalación del framework, con los comandos *python -m django start proyect* y *python manage.py startapp*, respectivamente. Para esto se empleó VSCode y se puede observar en la imagen 6.1.

En siguiente lugar, se llevó a cabo la propagación del modelo desde Django a PostgreSQL. Para ello, en primera instancia, se definieron las clases relativas a las tablas que se querían importar en el archivo *models.py* de Django como se muestra a continuación:

```
class Pasajeros(models.Model):
    tarjeta=models.CharField(max_length=50)
    fecha=models.DateTimeField(
    default=datetime.now().strftime(("%d/%m/%Y %H:%M:%S")))
    tusuario=models.CharField(
    max_length=14,verbose_name="Tu Usuario")
    titulo=models.CharField(max_length=14)
    descuento=models.CharField(max_length=14)
    daypoint=models.CharField(max_length=50)
    idtlv=models.CharField(max_length=50)
    codval=models.CharField(max_length=14)
```

Listing 6.1: Fichero *models.py* de la base de datos de viajes

Este código muestra la clase relativa a la base de datos de viajes de los pasajeros.

Una vez definidos todos los modelos se realizó se propagación hacia PostgreSQL.

Inicialmente, se usó el comando *makemigrations*, el cual genera un archivo de migración que describe las operaciones necesarias para llevar la base de datos al estado definido por los modelos. A continuación, se empleó el comando *migrate* . Este comando se encarga de ejecutar el archivo creado, llevando a cabo la migración. La salida de ambos comandos se puede observar en la Figura 6.2 y la Figura 6.3 respectivamente.

En la primera migración, además de la creación de la tablas del modelo en PostgreSQL, se crean las tablas relativas a aspectos de configuración de Django como el almacén de usuarios y grupos registrados en el panel de administración del framework.

Esta acción se realizó para todas las bases de datos del proyecto, incluyendo las preliminares mencionadas en la sección 4.1 y las definitivas mostradas en la sección 4.6. Todas las tablas creadas en PostgreSQL se pueden observar en la Figura 6.4.

Figura 6.2: Operación *makemigrations* de la clase Pasajeros

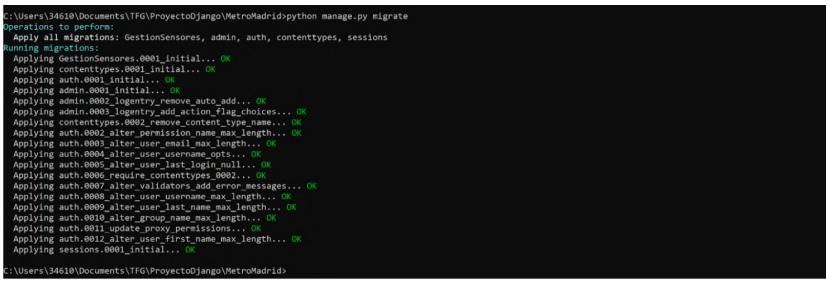


Figura 6.3: Operación *migrate* de la clase Pasajeros

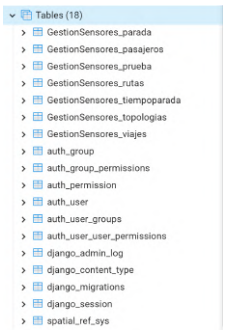


Figura 6.4: Tablas de datos propagadas desde Django a PostgreSQL

Una vez se obtuvieron los esquemas de todas las bases de datos en PostgreSQL, el siguiente paso que se llevó a cabo fue la carga de los datos.

Para importar los datos de manera correcta, se utilizó la función *Query Tool* de PostgreSQL [38] [39]. Se trata de una herramienta que permite ejecutar consultas SQL directamente en PostgreSQL a través de una interfaz gráfica.

La consulta que se empleó para cargar los datos de viajes fue la siguiente:

```
COPY public."GestionSensores_pasajero" FROM
    'C:\Users\34610\Documents\TFG\ENERO_VIAJES_2019.txt' DELIMITER
    '#' CSV HEADER
```

En este caso la sentencia COPY, de psql, carga los datos desde un archivo CSV, porporcionado por el usuario, en la tabla "GestionSensores_pasajero" del esquema "public" utilizando "#" como delimitador.

El mismo proceso se repitió para todas las tablas modificando la ruta del archivo, el nombre de la tabla y el delimitador, el cual depende del tipo de archivo.

Durante la importación de los datos en PostgreSQL, nos encontramos con un problema recurrente relacionado con los tipos de datos. El desafío era asegurarnos de asignar correctamente los tipos de datos a cada campo desde Django y luego cargarlos en PostgreSQL. Sin embargo, durante el proceso de importación, PostgreSQL generaba problemas debido a que el formato de nuestros datos no coincidía con los requisitos establecidos por el sistema de gestión de bases de datos, o se producían problemas en la segmentación de las columnas de los datasets importados. Los dos casos mas destacados fueron los siguientes:

- **Campo *Fecha* de la tabla de viajes.** El formato de fecha en nuestra base de datos sigue el patrón "dd/mm/yyyy hh:mm:ss". En dicho patrón "dd" representa el día, "mm" representa el mes, "yyyy" representa el año, "hh" representa la hora, "mm" representa los minutos y "ss" representa los segundos. Sin embargo, el formato de PostgreSQL para el tipo de datos *DateTime* es "yyyy-mm-dd hh:mm:ss". Inicialmente, para tratar de solucionarlo, se estableció en *models.py* de django la siguiente línea que también se pudo observar en el código 6.1:

  ```
      fecha=models.DateTimeField(
      default=datetime.now().strftime(("%d/%m/%Y %H:%M:%S")))
  ```

  En este fragmento de código, se utiliza el campo DateTimeField de Django para almacenar valores de fecha y hora. Se establece como valor predeterminado la fecha y hora actual utilizando la función datetime.now(). Además, se formatea el campo utilizando la función .strftime() para definir el formato deseado de la fecha.

  Sin embargo, a la hora de hacer las operaciones *makemigrations* y *migrate*, Django muestra un mensaje que indica que no podemos otorgarle ese formato a la fecha.

Para solucionarlo se optó por una opción simple pero eficaz. Se estableció el campo *Fecha* como un *CharField*, cadena de texto, y se importaron los datos. Una vez con los datos cargado en la base de PostgreSQL, se cambió el tipo de datos del campo fecha a tipo "timestamp without time zone" utlizando la interfaz gráfica del sistema gestor de bases de datos. Finalmente los datos se almacenaron con el formato apreciado en la Figura 6.5.

fecha timestamp without time zone
2019-01-02 11:51:12
2019-01-02 10:47:08
2019-01-02 14:04:12
2019-01-02 14:24:38
2019-01-02 11:52:14
2019-01-02 14:08:56
2019-01-02 14:01:46
2019-01-02 14:03:54
2019-01-02 13:38:02
2019-01-02 11:33:40
2019-01-02 12:24:54

Figura 6.5: Campo *Fecha* almacenado en PostgreSQL

- **Campos latitud y longitud en la tabla Topologías.** En este caso, el problema se presentaba con los datos de los campos latitud y longitud en la base de datos de la topología de la red. Estos datos eran números decimales en los que el archivo CSV de origen separaba la parte entera y decimal utilizando una coma.

  Debido a la naturaleza de estos campos se les establecieron el tipo de dato *DecimalField*, el cual se encarga de almacenar números decimales [40].

  No obstante, a la hora de importación de estos datos se produjo un error que indicaba que había más columnas de las esperadas. Esto ocurría ya que, a pesar de que habíamos establecido bien el tipo de datos, el separador utilizado para distinguir las columnas en un archivo CSV es la coma y, por lo tanto, la división que se hacía en los números era interpretada como una nueva columna.

  Para solucionar este problema se optó por la misma decisión que en caso anterior: cargar los datos como texto y modificarlos en la propia plataforma de PostgreSQL. La única diferencia es que en este caso, en vez de usar la interfaz gráfica proporcionada por pgAdmin4, se ejecutó una consulta SQL a través de la herramienta *Query Tool* de la siguiente forma:

```
ALTER TABLE public."GestionSensores_topologias"
ALTER COLUMN longitude TYPE double precision
USING replace(longitude, ',', '.')::double precision;
```

Esta consulta cambió las comas por puntos y modificó el formato de el campo longitud de cadena de texto a *double precision*, el cual trata numeros decimales con mayor precisión que el tipo *float*.

6.1.2 Limpieza de datos

Esta sección está dedicada a la limpieza y depuración del conjunto de datos empleado en el desarrollo de este TFG. La limpieza de datos es una etapa imprescindible en cualquier proyecto de Big Data. Gracias a ella, se puede mejorar la eficiencia en el análisis, además de la calidad y coherencia de los datos, entre otras aspectos relevantes.

En el desarrollo de este proyecto se llevo a cabo la limpieza de datos en dos plataformas distintas: VSCode (empleando Python) y PowerBI. Se emplearon ambas herramientas con el objetivo de enriquecer el aprendizaje personal, abordando eficazmente todos los desafíos relativos a la depuración de datos, para adquirir y perfeccionar las habilidades de programación y visualización intuitiva.

Python

Gran parte de la limpieza de datos llevada a cabo usando el lenguaje de programación Python, se mencionó previamente de manera extensa en la sección 4.2.3. Por lo tanto, en esta sección se mencionará dos casos concretos en los cuales esta limpieza ayudó a mejorar la calidad del conjunto de datos.

- **Campo Descuento.** Al emplear la función *.unique()*, sobre este campo sin realizar previamente ningún preprocesado sobre los datos, se pudo observar la salida mostrada en la Figura 6.6.

```
print('Cantidad de valores distintos que puede tomar la columna DESCUENTO : ',len(df_PWBI.DESCUENTO.unique()))
print('Algunos de ellos: \n',df_PWBI.DESCUENTO.unique())

Cantidad de valores distintos que puede tomar la columna DESCUENTO :  16
Algunos de ellos:
 [0.0 2.0 4.0 1.0 3.0 5.0 nan 30.0 13.0 '00' '02' '01' '03' '04' '05' '3E']
```

Figura 6.6: Función *.unique()* sobre el campo descuento

Como se puede apreciar en la figura, además de haber claramente valores que son de tipo texto y otros de tipo numérico, se puede determinar que hay una serie de valores

que transmiten la misma información pero aparecen desglosados como resultados diferentes. Estos son los casos de los valores 0.0 y “00”, 1.0 y “01“, 2.0 y “02”, entre otros. Esto demuestra que los valores no están estandarizados, y por lo tanto muestran información poco clara sobre el conjunto de datos.

Posteriormente, se realizó la operación mostrada en la Figura 6.7. Gracias a esa operación, se logra estandarizar los diferentes valores que toma el conjunto de datos para ese campo, lo cual provoca una mejora significativa en la calidad de la información que se va a obtener con el posterior análisis.

```
print('Algunos de ellos: \n',df.DESCUENTO.unique())
```

```
Algunos de ellos:
 [0.0 2.0 4.0 1.0 3.0 5.0 'No se conoce' 30.0]
```

Figura 6.7: Resultado de la estandarización del campo descuento

- **Campo Fecha.** Al intentar cambiar el tipo de datos del campo *Fecha* de “object” a “datetime” en el subconjunto aleatorio, nos enfrentamos a un problema significativo [41]. Dicho problema se producía por la existencia de dos formatos de fecha diferentes para el mismo campo: “dd/mm/yyyy hh:mm:ss” y “yyyy-mm-dd hh:mm:ss”. Por culpa de este hecho, se generaban ciertos registros falsos que indicaban que existía información de meses distintos a enero, ya que algunos casos se intercambia el día por el mes en el *parsing* del campo. Además, debido a este intercambio, se asignaba una afluencia desproporcionada de pasajeros y viajes al día 1 de cada mes, lo cual, junto con el hecho anterior, provocaba un análisis incorrecto de los datos. Todo esto ocurría a raíz de que el parseo de los datos solo se hacía con un formato de fecha específico, entonces todas las fechas que no tuviesen ese formato no se lograban interpretar correctamente.

  Para solucionarlo, se implementó la siguiente función [42]:

```
def parse_fecha(fecha_str):
    try:
        return pd.to_datetime(fecha_str, format='%Y-%m-%d %H:%M:%S')
    except ValueError:
        return pd.to_datetime(fecha_str, format='%d/%m/%Y %H:%M:%S')
```

  Esta función intenta transformar el campo de fecha utilizando uno de los formatos establecidos. En caso de que no tenga éxito, prueba a convertir el valor del campo utilizando el otro formato. Con esta operación se logró que ambos formatos estuviesen recogidos a la hora de transformar los datos, consiguiendo, así, una buena interpretación del campo.

PowerBI

Gracias a haber realizado previamente la limpieza y preprocesado de datos en Python, el proceso posterior en Power BI resultó más fácil y eficiente, ya que existían directrices y un proceso definido que facilitaron la tarea.

Es necesario destacar, que para esta depuración de los datos no se empleó el dataset entero sino un subconjunto representativo de los datos, el cual se obtuvo a través de la misma operación realizada en la sección 4.2.5, pero esta vez con un subconjunto con 100.000 filas más. Para llevar a cabo este proceso se utilizó el editor de *Power Query*. Este editor consiste en una interfaz visual interactiva, que permite realizar transformaciones y manipulaciones avanzadas en los datos de origen, con el objetivo de prepararlos para su posterior análisis y visualización.

En primer lugar, se procedió con el **cambio de tipo de datos de las variables**. Para ello, se seleccionó un recuadro pequeño al lado del encabezado de cada columna a través del cual, tras pulsarlo, se enseña un desplegable que muestra todos los tipos de datos a los que se puede modificar la columna, como se puede observar en la Figura 6.8. Se cambiaron todos los campos a tipo texto, menos el campo fecha. Este último, como es lógico se le estableció el tipo de datos "Fecha/Hora".

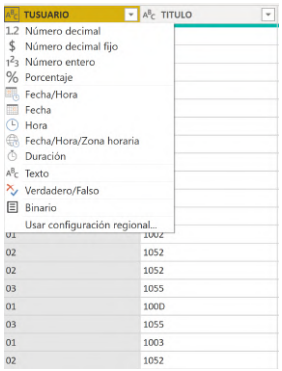


Figura 6.8: Desplegable de tipos de datos en PowerBI

El siguiente paso fue el **tratamiento de los nulos de la variable *Descuento***. Para ello se realizó la siguiente operación en lenguaje DAX [43] (que es el lenguaje de consultas empleado en PowerBI):

```
= Table.ReplaceValue(#"Tipo cambiado1","","No se
sabe",Replacer.ReplaceValue,{"DESCUENTO"})
```

En esta operación lo que se hace es cambiar los valores vacíos, " ", encontrados en la columna *Descuento*, por el mensaje "No se sabe".

A continuación, se realizó el **tratamiento del desglose de valores de la variable *Des-***

cuento comentados previamente en la sección 6.1.2. Mientras que en Python el problema se solventaba con aplicar *.astype()* sobre el campo Descuento, en PowerBI fue necesario ir modificando valor a valor de la siguiente manera:

```
1    = Table.ReplaceValue(#"Tipo
     cambiado4","00","0",Replacer.ReplaceText,{"DESCUENTO"})
```

Gracias a esta operación se obtuvo el resultado mostrado en la Figura 6.9.

DESCUENTO
0
1
2
3
4
5
No se sabe

Figura 6.9: Resultado de la estandarización del campo descuento en PowerBI

Es necesario destacar que en este caso, no se visualizan tantos valores en la variable *Descuento*, por el hecho de que se está tratando con un subconjunto reducido de los datos, tal como se comentó previamente.

Por último, a modo de generar una visualización más intuitiva para el usuario final del *dashboard*, se llevó a cabo la sustitución de los id's de ciertos campos del conjunto de datos por sus significados reales, entre otras operaciones. Esto facilitó, en gran medida, la comprensión de los visuales.

Un ejemplo de esta operación es el siguiente extracto de una consulta DAX:

```
1    = Table.ReplaceValue(#"Valor
     reemplazado13","3","Joven",Replacer.ReplaceText,{"TUSUARIO"})
```

En este código sobre el campo *Tusuario*, se remplaza el id "3" por el valor "Joven", el cual corresponde con el tipo de usuario asociado a ese id.

6.1.3 Implementación de las consultas en Django

Para realizar estas consultas se utilizó la información proporcionada por Django para la realización de *queries* [44].

Obtención de líneas de transporte

Para obtener la ruta de una línea en el mapa, el usuario puede seleccionar una de las paradas cercanas que se muestran en pantalla al hacer "clic" en el icono correspondiente.

En primer lugar, se establece que una vez se haga "click" sobre una parada cercana se obtenga tanto el tipo de transporte, como la línea correspondiente a la parada:

```
stop.on("click", (function(linea, tipo) {
                        return function() {
                            LineaT(linea, tipo);
                        }
                    })(data.paradas[i].Linea,
    data.paradas[i].tipo));

```

En este código se puede observar que una vez el usuario pulse en el objeto "stop", el cual es relativo a la parada en cuestión, se llama a una función con nombre "LineaT", a la cual se le pasa tanto la línea como el tipo de transporte relativo a esa parada.

Lo primero que hace la función "LineaT" es lo siguiente:

```
 function LineaT(linea, tipo) {
                $.post('/lineas/', {linea: linea, tipo: tipo}

```

Realiza una solicitud POST sobre la vista "líneas" utilizando una la librería jQuery Ajax [45]. Esta librería proporciona una serie de métodos y funciones que facilitan el manejo de solicitudes HTTP. Esta petición le envía, a la vista "líneas", la línea y el tipo de transporte obtenidos del proceso anterior.

En el archivo "views.py" de proyecto se define la siguiente función "líneas" :

```
def lineas(request):
    if request.method == 'POST':
        #recuperacion de campos
        lin= request.POST.get('linea')
        typ= request.POST.get('tipo')
        #consulta a la base
        lineas= Topologias.objects.filter(line=lin, TYPE=typ)
        #listado de puntos
        puntos = [[punto.latitud, punto.longitud] for punto in
    lineas]
        return JsonResponse({'linestring': puntos})

    return render(request, 'GestionSensores/topologia.html')

```

En esta vista se llevan a cabo los siguientes pasos [46] :

1. Obtención de la línea y parada provenientes del paso anterior utilizando una petición POST.

2. Filtrado en PostgreSQL, sobre la base de datos *Topologías*, de aquellas filas que coincidan con esa línea y parada en concreto. Para ello se usa la función *objects.filter()*, que sirve para realizar consultas y filtrar objetos en una base de datos de acuerdo con ciertos criterios de búsqueda.

3. Almacenado y serialización de las coordenadas de las paradas. El almacenado se lleva a cabo a través del campo *punto* de la tabla Topologías. Se almacena primero la componente y (latitud) y posteriormente la componente x (longitud). Posteriormente se lleva a cabo la serialización de los puntos en formato JSON a través de la función *JsonResponse*. Esta función devuelve una respuesta HTTP que contiene un diccionario con clave "linestring" y como valor la lista de puntos almacenada.

4. Envío del contenido de la solicitud POST. Para ello se usa la función *render*, la cual se encarga de renderizar la plantilla topología, relativa a la página, con el diccionario obtenido como respuesta a la solicitud POST.

Una vez obtenido el contenido solicitado, volvemos a la función LineaT:

```
function LineaT(linea, tipo) {
                $.post('/lineas/', {linea: linea, tipo: tipo},
   function(data) {
                    linep = L.Routing.control({
                        waypoints: data.linestring,
                        show: false,
                        createMarker: function(i, waypoint, n) {
                          return null;
                        } //para que no me muestre todos los
                        marcadores para cada parada
                      }).addTo(map); // dibujar la línea
                    map.fitBounds(linep.getBounds());
                });
            }

```

Se emplea la función *L.Routing.control*. Esta función pertenece a la biblioteca *Leaflet Routing Machine* [29], la cual se encarga de crear un control de enrutamiento sobre un mapa Leaflet, permitiendo a los usuarios planificar rutas entre distintos puntos geográficos. Los campos que se emplean en nuestro proyecto son los siguientes:

- **Waypoint**. Este campo sirve para especificar los puntos de la ruta. En nuestro caso, le pasamos un *linestring* de objetos [lat,long] que permitirá a la función calcular y mostrar la ruta en el mapa.

- **Show**. El campo *show* permite mostrar una serie de indicaciones en el mapa para guiar al usuario en la realización de la ruta, cercano a lo que hace la aplicación *Google Maps* o

Mapas, entre otras. Sin embargo, debido a que nuestro objetivo no es ese, desactivamos el funcionamiento de esa opción, estableciéndola como *false*.

- **CreateMarker**. Por último, se encuentra la funcionalidad *CreateMarker*. Esta característica permite personalizar la creación de marcadores para cada parada de la ruta. Con la finalidad de obtener un diagrama limpio, sin sobrecarga de marcadores, desactivamos también esta opción estableciéndola a estado *null*.

Finalmente, tras esta función se añade la línea al mapa y se emplea *map.fitbounds* para centrar la visualización del mapa en la ruta determinada.

Filtrado temporal de historial de viajes

El primer lugar, para obtener tanto la tarjeta de usuario como el intervalo temporal de búsqueda, hay que establecer una manera, desde el Front-End, que habilite al usuario la introducción de esos datos :

```
<form action="/buscar/" method="GET">
            <input type="text" name="pasaj"
    placeholder="Identificador de tarjeta" ><br><br>
            <label for="fecha_inicio" style="color: white;">Fecha
    de inicio:</label>
            <input type="date" name="fecha_inicio"
    id="fecha_inicio" min="2019-01-01" max="2019-01-31">
            <label for="fecha_fin" style="color: white;">Fecha de
    fin:</label>
            <input type="date" name="fecha_fin" id="fecha_fin"
    min="2019-01-01" max="2019-01-31">
            <input type="submit" value="Buscar">
        </form>
```

En esta sección de código se crea un atributo <*form*> [47] en el que se pueden observar los siguiente aspectos:

- Se realiza una consulta GET a la vista “buscar”.
- Se permite al usuario introducir un valor de tipo texto.
- Se establece un calendario interactivo en el navegador web que posibilita a los usuarios la selección de una fecha de inicio y de fin sobre el intervalo temporal sobre el que quieren obtener su historial de viajes. Debido a que los datos solamente pertenecen al mes de enero de 2019, se definen como fecha mínima y fecha máxima los días 1 y 31 de dicho mes, respectivamente.
- Finalmente, se crea un botón para enviar la información y dar inicio a la consulta HTTP.

Posteriormente se procede con el análisis de la vista "buscar" almacenada en el archivo *views.py*:

```
1  def buscar(request):
2      if request.method == 'GET':
3          usuariotren = request.GET.get("pasaj")
4          if len(usuariotren) != 32:
5              mensaje = "La tarjeta no está bien escrita"
6          else:
7              fecha_inicio = request.GET.get("fecha_inicio")
8              fecha_fin = request.GET.get("fecha_fin")
9
10             if fecha_inicio and fecha_fin:
11                 fecha_inicio = datetime.strptime(fecha_inicio,
12                 "%Y-%m-%d")
13                 fecha_fin = datetime.strptime(fecha_fin, "%Y-
14                 %m-%d")
15                 pasajerox =
16                 Pasajeros.objects.filter(fecha__gte=fecha_inicio
17                 ,fecha__lte=fecha_fin,
18                 tarjeta__icontains=usuariotren)
19
20             else:
21                 pasajerox =
22                 Pasajeros.objects.filter(
23                 tarjeta__icontains=usuariotren)
24
25             top_resul = Topologias.objects.filter
26             (DPAYPOINT__in=[pasajero.daypoint for pasajero in
               pasajerox])
27
28             return render(request, "resultados_busqueda.html",
29             {"pasajero": pasajerox, "query": usuariotren,
30             "top": top_resul})
31     else:
32         mensaje = "La búsqueda está vacía."
33     return HttpResponse(mensaje)
```

Listing 6.2: Vista de filtrado temporal del historial

Para desarrollar esta vista se cumplieron los siguiente puntos:

- Comprobación de que el usuario introdujo contenido en la caja de texto. Para ello, en la línea 2 se comprueba si se ha realizado el método HTTP. En caso de que haya sido así, se obtiene aquello que el usuario introdujo, mientras que, si el usuario no escribió nada, se mostrará por pantalla el mensaje "La búsqueda está vacía" (como se observa en la línea 32).

- Comprobación básica del elemento introducido. Para ello se hace una básica pero efectiva comprobación de la longitud del mensaje introducido por el usuario. Dado que la tarjeta es un identificador alfanumérico de 32 dígitos, se comprueba si el mensaje introducido tiene la misma longitud. En caso de no ser así, se mostrará el mensaje "La tarjeta no está bien escrita" por pantalla.
- Obtención de las fechas de inicio y fin (líneas 7 y 8).
- Casuísticas según la introducción de las fechas. Dado que la selección del intervalo temporal tiene un carácter opcional, se implementaron dos alternativas dependiendo de si los usuarios deciden introducir o no las fechas:
  - Si los usuarios proporcionan las fechas, se convierten, inicialmente, sus valores a tipo *datetime* a través de la función *datetime.strptime()*, con el objetivo de asegurarse de que estén en el formato adecuado. A continuación, se realiza una búsqueda filtrada en la tabla *Pasajeros*. En esta búsqueda se establecen las condiciones de que la fecha establecida sea mayor o igual a la fecha de inicio, y menor o igual que la fecha de fin. Para esto, se usan las funciones __gte y __lte respectivamente. Ambas operaciones, que se realizan en las filas correspondientes a la tarjeta introducida por el usuario, verifican si el campo "tarjeta" coincide utilizando la función __icontains.
  - Si los usuarios no proporcionan las fechas, la búsqueda se basa en obtener todos los viajes asociados a la tarjeta proporcionada sin ningún tipo de restricción adicional.
- Unión con la tabla de topologías. Una vez obtenidas las filas resultantes del paso anterior, se realiza un *join* con la tabla topologías a través del campo *Dpaypoint*. Esto se realiza con el objetivo de obtener información específica de las paradas en las que el usuario ha estado como el nombre, la línea, la ubicación geográfica, etc.
- Envío de los resultados de la consulta a la plantilla "resultados_búsqueda". Se realizará a través de la función *render*.

Finalmente, la plantilla *resultados_búsqued*a será la encargada de mostrar un mapa, en el cual se vean las paradas en las que el usuario ha estado, y el número de veces que las ha utilizado, entre otra información relevante.

6.2 Pruebas experimentales

Esta sección está dedicada a la mención de alguna de las pruebas llevadas a cabo durante el proceso de desarrollo de la aplicación web.

6.2.1 Experimentación general

Para facilitar la eficiencia y eficacia de este procedimiento se utilizó una base de datos de prueba. La tabla relativa a esta base de datos tomó el nombre de "Pruebas", y se puede observar en la Figura 6.4 junto con el resto de tablas del modelo. Esta tabla sirvió tanto para agilizar las consultas, como para mejorar la destreza en la gestión de diferentes tipos de campos, sobre todo los geográficos. Su contenido se resume a dos filas y contiene datos procedentes del dataset de viajes.

6.2.2 Pruebas sobre la consulta del trazado de líneas

Para llegar al estado final, mostrado en la interfaz de la aplicación web se siguieron los siguientes pasos a modo incremental:

1. Obtención de algún tipo de información al hacer clic sobre la parada cercana.

2. Adquirir los campos línea y tipo al pulsar sobre la parada.

3. Dibujar una única línea una vez se hacía "click" en cualquiera de las paradas. En este paso se pintaba una línea en concreto, la número 516, empleando un objeto GeoJSON [48], que unía todos los punto pero de manera aleatoria. Esto se debía a que el objeto unía los puntos en el orden en el que le llegaban [49]. Además, no tenía noción geospacial del sistema de carreteras del municipio de Madrid. El resultado fue el mostrado en la figura 6.10.

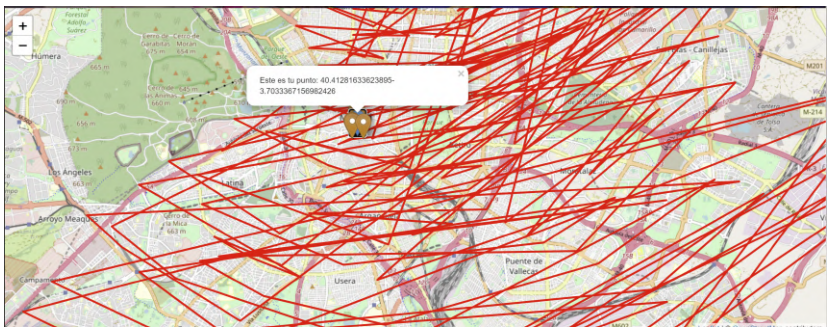


Figura 6.10: Resultado inicial de la creación de rutas
[50]

4. Dibujar el trayecto con el orden adecuado de las paradas ajustándose a los límites de las carreteras. Para ello, se empleó la API Leaflet Routing Machine tal como se menciona en el subapartado 6.1.3. El resultado final de esta iteración sección se muestra en la figura 6.11.

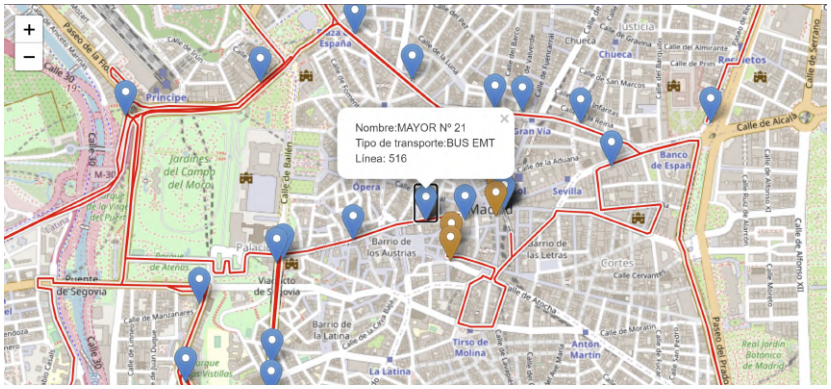


Figura 6.11: Resultado de creación de rutas usando *Leaflet Routing Machine*

5. Refinamiento de la visualización. En este último punto, se estableció la visualización del recorrido de cada línea acorde a la parada pulsada, la cual se mejoró notablemente.

6.2.3 Optimización de la consulta de historial de usuarios

La operación relativa a la consulta del historial de pasajeros tenía un gran inconveniente, la eficiencia. La consulta tardaba una gran cantidad de tiempo en ejecutarse. Este hecho se debía, en gran medida, a las consultas sobre el campo *tarjeta* y al "Join" entre las tablas *Pasajeros* y *Topologías* mencionado como penúltimo paso seguido en la subsección 6.1.3.

Como solución a este problema se propuso la creación de índices para lo cual se siguió el siguiente proceso:

- Cálculo del tiempo de la consulta sin índices. El tiempo resultante de la ejecución de la consulta sin ningún índice era de 6 minutos y 25 segundos.
- Índice sobre el campo *Tarjeta* de la tabla *Pasajeros*. Este proceso se llevó a cabo a través de la herramienta *Query Tool* de PostgreSQL y se realizaron las operaciones "create_index", para la generación del índice, y *vacumm analyze*, para limpiar y optimizar la base de datos. La creación de este índice consiguió reducir la duración a 2 minutos y 5 segundos. Ambas operaciones se pueden observar en la figura 6.12.
- Índice sobre el campo *Dpaypoint* de las tabla *Pasajero* y *Topologías*. La creación de estos índices no afectó en el rendimiento de las consultas.

Se puede concluir que se consiguió una reducción del tiempo de consulta a menos de la mitad del inicial. Además, se pudo conocer que la ralentización de la consulta se debía, en gran medida, a la iteración sobre el campo *tarjeta*, debido al gran tamaño de la base de datos de viajes.

(a) Create Index

(b) Vacumm Analize

Figura 6.12: Creación del índice sobre el campo tarjeta

Capítulo 7

Solución desarrollada

Este capítulo se centra en la realización de una visita guiada por las principales páginas y funcionalidades implementadas en el desarrollo de la aplicación web. Este proceso estará apoyado por una serie de capturas que permitirán visualizar el resultado final.

7.1 Página de Inicio

Esta página, tal y como se había pensado de manera inicial en la sección 4.5, se basó en mostrar una breve descripción del objetivo de la aplicación, y una imagen relativa a alguno de los medios de transporte incluidos en el sistema público de transporte del municipio de Madrid. La imagen escogida fue la relativa a una parada de la red del metro de Madrid, como se ve en la Figura 7.1. Tanto la imagen de la parada como la imagen de fondo de la página web son de dominio público.

Figura 7.1: Página de inicio de la aplicación.

7.2 Página de Historial de Viajes

En primer lugar, la página presenta una breve descripción del objetivo de la misma. A continuación, se dispone una caja de texto para que el usuario final pueda introducir el identificador de su tarjeta de viaje. En la parte inferior de la caja de texto, se habilitaron dos filtros opcionales relativos a la fecha de inicio y de fin del intervalo temporal sobre el que el usuario quiera realizar la consulta. Esta funcionalidad se presenta de una manera gráfica e interactiva, en la cual se despliega un calendario sobre el cual se puede seleccionar una fecha con un solo *click* sobre el número del día, como se observa en la Figura 7.3. Por último, se habilitó un botón para ejecutar la búsqueda. Todas las características mencionadas previamente se muestran en la Figura 7.2.

Figura 7.2: Página de historial de usuarios.

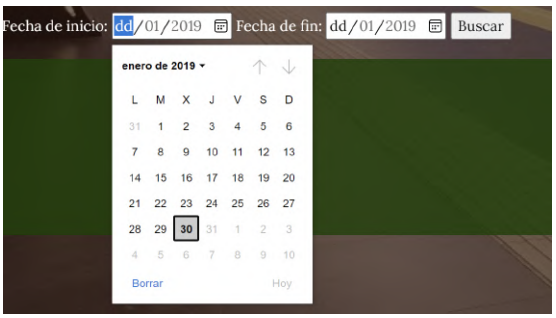


Figura 7.3: Interfaz de selección de fechas.

Después de ejecutar la consulta, los resultados se mostrarán en una nueva pantalla. Esta pantalla, además de permitir volver a hacer otra búsqueda sin necesidad de volver atrás en la página, enseña el total de viajes realizados por el usuario en el intervalo temporal seleccionado. Por otra parte, muestra un mapa centrado en el municipio de Madrid con una serie de marcadores sobre él. Estos marcadores indican la localización geográfica de las paradas a las que el usuario se subió en el intervalo de tiempo seleccionado, tal y como se observa en la

Figura 7.4. Como se observa en la Figura 7.5, si se pulsa sobre dichas paradas se obtiene la siguiente información sobre el tipo de transporte, la línea, el nombre de la parada y un contador con el total de veces que el usuario utilizó la parada en ese intervalo de tiempo.

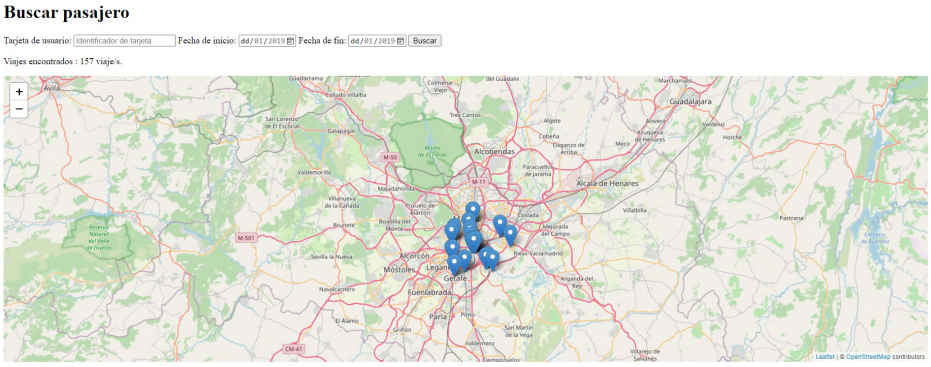


Figura 7.4: Resultado de la consulta de historial de usuarios.

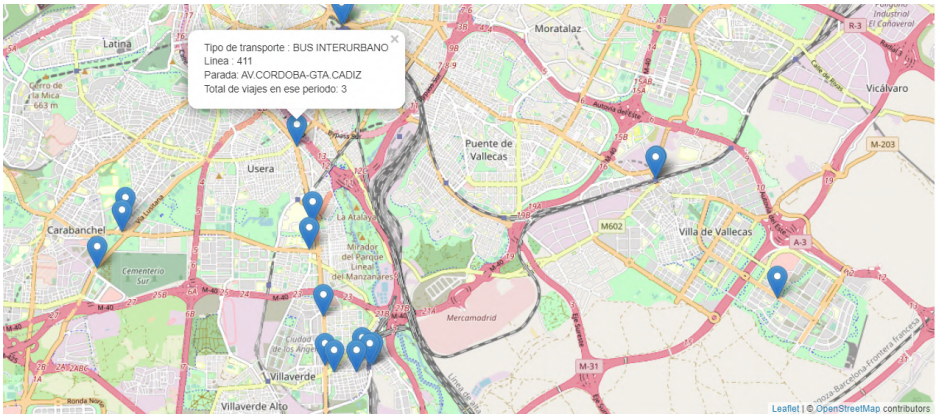


Figura 7.5: Información específica sobre cada parada del historial.

7.3 Página de Visualización Interactiva

Al igual que en la página anterior, en primer lugar se muestra una breve descripción de la finalidad de la página. A continuación, se muestra un *dashboard* o panel de control, generado con la herramienta PowerBI [51] que permite visualizar los siguientes elementos gráficos:

- Afluencia de pasajeros en dos categorías - día laborable y no laborable. Se trata de un gráfico de líneas que permite comparar la cantidad de pasajeros en los días laborables y no laborables a lo largo del mes.
- Contador de total de pasajeros: Muestra la cantidad total de pasajeros registrados.

- Contador de total de viajes: Muestra el número total de viajes realizados.
- Comparativa de total de viajes y pasajeros según el día de la semana. Es un diagrama de columnas que permite comparar el total de viajes y pasajeros en diferentes días de la semana.
- Comparación del total de pasajeros según su categoría. Muestra un diagrama de líneas el cual permite comparar el total de pasajeros a lo largo del mes, desglosado según diferentes categorías dentro de las cuales destacan 3ª edad , Infantil o Joven, entre otras.
- Top 5 paradas más transitadas. Consiste en un diagrama de columnas que muestra las cinco paradas con más afluencia de pasajeros.
- Tipo de descuentos más aplicados en los viajes. Se trata de un diagrama circular que muestra la proporción de viajes según el tipo de descuento aplicado en ellos.
- Mapa de calor. Representa visualmente el municipio de Madrid en un mapa y muestra la intensidad de los viajes que se producen en diferentes zonas. Este gráfico se muestra en una página específica con motivo de mejorar la visualización del mapa.

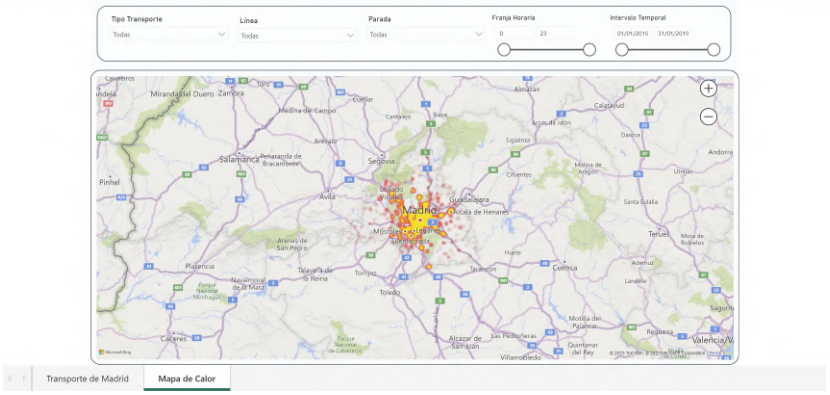
A su vez, cada uno de estos gráficos pueden ser personalizados mediante el uso de diversos filtros, que permiten ajustar los datos visualizados según diferentes criterios. Al aplicar cada uno de estos filtros, se modifica la información mostrada en todos los visuales mencionados previamente.

- **Tipo de transporte.** Permite seleccionar el tipo de transporte deseado. Las opciones posibles son Bus Interurbano, Bus EMT, Cercanías, Metro y Ligero.
- **Línea.** Permite seleccionar una línea de transporte específica.
- **Parada.** Posibilita filtrar los datos según el nombre de la parada.
- **Franja horaria.** Brinda la posibilidad de establecer un rango de horas específico, abarcando desde las 0 a las 23 horas.
- **Intervalo temporal.** Posibilita la selección de un intervalo preciso de tiempo dentro del mes.

Tanto la disposición general de la página, como las distintas pantallas del *dashboard*, se pueden apreciar en las figuras 7.6 y 7.7 respectivamente.

Figura 7.6: Página de visualización interactiva

(a) Primera pantalla del panel de control



(b) Segunda pantalla del panel de control

Figura 7.7: Pantalla del dashboard interactivo

7.4 Página de exploración de la topología de la red

Tal como se hizo en las dos páginas anteriores, se incluye una breve descripción de los objetivos de la página. Por otra parte, se muestra un mapa centrado en el municipio de Madrid sobre el que se le sugiere al usuario "hacer click" en algún punto geográfico del mismo, como se aprecia en la figura 7.8. Una vez el usuario pulsa en cualquier lugar del mapa se establece un marcador en ese punto geográfico sobre el cual se abre un *pop-up* que muestra sus coordenadas. A su vez, se muestran las paradas de cualquier tipo de transporte que se encuentre a menos de 200 metros del punto seleccionado. Estas paradas se muestran a través de marcadores de un color distinto del color del marcador del punto seleccionado por el usuario [52]. Esta consulta se puede apreciar en la figura 7.9.

Por último, si el usuario pulsa cualquiera de los marcadores respectivos a las paradas cercanas se realizarán dos acciones, que se observan en la imagen 7.10:

- **Mostrar información específica de la parada a través de un *pop-up*.** Se enseña el tipo de transporte, el número de línea y el nombre de la parada.
- **Trazado de la ruta.** Se dibuja la ruta de la línea, a la que pertenece la parada pulsada, sobre el mapa del municipio de Madrid.

Figura 7.8: Página de exploración geográfica

7.5 Página de administración

A pesar de que la página de administración viene definida por Django, es necesario habilitarla para su uso, además de añadir ciertas características específicas dependiendo de los

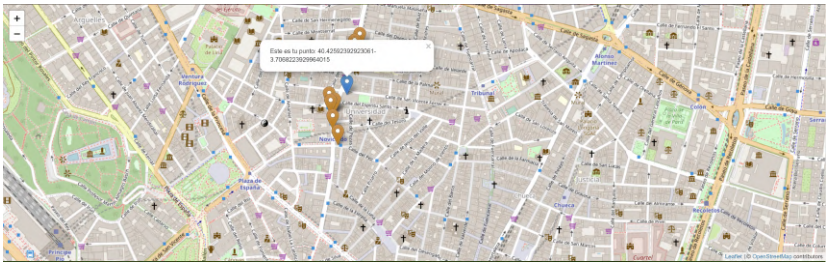


Figura 7.9: Selección de punto y consulta de paradas cercana

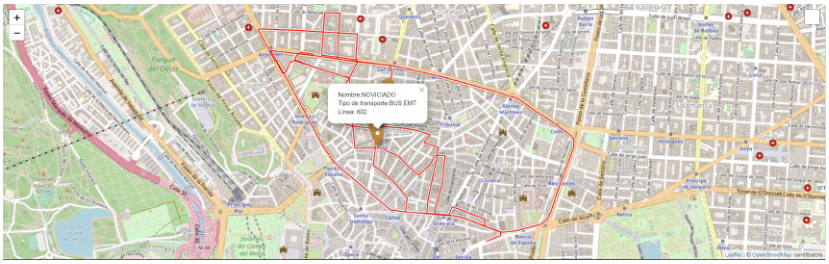


Figura 7.10: Obtención de información de la parada y consulta de ruta de la línea

datos que se quieran gestionar. En nuestro caso, estas características se basaron en propagar la base de datos de viajes a la plataforma de administración, con el objetivo de favorecer su gestión. Además de esa funcionalidad, este panel de administración proporciona las siguiente características:

- Portal de autenticación
- Añadir y eliminar usuarios
- Añadir y eliminar grupos de usuarios
- Asignar roles y configuraciones específicas
- Administración y edición de las bases de datos propagadas
- Visualización de acciones recientes llevadas a cabo en la plataforma

En las figuras B.1a y B.1b se pueden apreciar dos pantallas relativas al panel de administración de Django.

Capítulo 8

Conclusiones y trabajo futuro

En este capítulo se realiza un resumen de ciertos aspectos finales del proyecto como el grado de cumplimiento de los objetivos, las funcionalidades más importantes y la utilidad que aportan en el mundo real y los conocimientos adquiridos durante la realización del proyecto. Además, se mencionan las posibles ampliaciones a realizar de cara a mejorar el funcionamiento de la aplicación web.

8.1 Conclusiones

En primer lugar, se puede concluir que todos los objetivos del proyecto han sido conseguidos de manera exitosa.

Por otra parte, la característica o funcionalidad más importante desarrollada en este proyecto es la posibilidad de que los trabajadores y directivos del Consorcio de Transporte de Madrid puedan realizar análisis y adquirir información de calidad, de manera intuitiva y eficaz, que les permita tomar decisiones exitosas orientadas a la gestión del transporte público del municipio madrileño. Esto se consigue, en gran medida, gracias a la página de visualización interactiva y a la de exploración geográfica.

La primera de ellas permite conocer los patrones de uso del sistema de transporte público por parte de los ciudadanos de la capital española de una manera muy fácil. Una vez obtenida esta información, se puede utilizar la segunda página para, entre otras posibles finalidades reales, conocer si aquellas zonas geográficas con mayor afluencia de pasajeros tienen una buena red de paradas y líneas a su alcance.

Gracias al servicio que propone la aplicación web, se consigue la optimización del sistema público de transporte, lo cual redunda en la calidad de vida de los ciudadanos así como en la mejora de las condiciones del aire del municipio de Madrid.

Un aspecto muy relevante, íntimamente relacionado con la implementación y desarrollo del proyecto, es el conocimiento adquirido.

Durante la realización del proyecto se ha adquirido una mayor destreza en el tratamiento y análisis de grandes volúmenes de datos, ya que, a pesar de que en el grado se han estudiado técnicas para la manipulación de los mismos, nunca se había tratado con un conjunto de datos tan grande.

Por una parte, gracias a este proyecto se pudo mostrar el conocimiento del alumno relacionado con el análisis exploratorio de datos, adquirido en asignaturas como *Modelización Estadística de Datos de Alta Dimensión (MEDAD)*.

Por otro lado, este proyecto permitió el aprendizaje en profundidad del uso del software de visualización PowerBI, el cual se utilizó, a modo de primer contacto, en la asignatura del grado llamada *Bases de Datos Analíticas (BDA)*.

También, este proyecto permitió ampliar y demostrar el dominio en la integración y modelización de bases de datos, haciendo hincapié, más concretamente, en aquellas con información espacio-temporal. Los conocimientos en este campo fueron adquiridos a través de tres asignaturas del grado: *Introducción ás Base de Datos (IBD)*, *Modelaxe de Bases de Datos (MBD)* y *Representación e Xestión de Datos Espazo-Temporais (RXDET)*.

Por último, es destacable el gran aprendizaje logrado en el desarrollo de páginas web, en este caso particular, empleando el framework Django. Este hecho ha supuesto un gran reto personal, debido a que es un aspecto nuevo no estudiado ni tratado en el grado.

8.2 Trabajo Futuro

En esta sección se describen una serie de posibles futuras mejoras y ampliaciones, que se podrían realizar sobre la aplicación, las cuales no estaban encuadradas como objetivos del proyecto. Alguna de ellas son:

- **Proceso de automatización.** Sería de gran interés la implementación de un mecanismo que permita la recopilación, limpieza y carga de los datos de viajes de manera automática a la base de datos de la aplicación.

  Actualmente con los scripts desarrollados por el alumno, se consigue realizar este proceso de manera semi-automática, por lo cual lograr una automatización completa no se presupone un proceso extremadamente costoso.

  Lograr la automatización permitiría tener datos de uso casi a tiempo real, lo cual sería muy útil a la hora de la toma de decisiones. Además, facilitaría la función de gestión de la aplicación, ya que se evitaría recurrir de manera periódica a los administradores para realizar dichas operaciones.

- **Estadísticas personales.** Una opción para aumentar la satisfacción con la aplicación por parte de los usuarios del metro sería incluir una sección de estadísticas persona-

lizadas en la página de historial de usuarios. Esta sección podría mostrar información como el medio de transporte más utilizado, los días de la semana con mayor o menor cantidad de viajes o los trayectos más comunes del usuario. Esto permitiría gozar a la aplicación de un mayor grado de personalización, lo cual puede ser bien aceptado por el colectivo de pasajeros del transporte público.

- **Internacionalización de aplicación.** Sería un buen objetivo permitir que la información se muestre en múltiples lenguajes a mayores del español. Dado que Madrid es un municipio multicultural y con una gran cantidad de habitantes de nacionalidades muy diferentes, podría ser una buena opción permitir visualizar las funcionalidades de la aplicación en sus lenguas de origen.

Apéndices

Apéndice A

Interfaces de usuario adicionales

En este capítulo se incluyen 2 interfaces de usuario que no se incluyeron en la 4.5 debido a la carencia de espacio disponible .

Figura A.1: Mockup de la página de búsqueda de historial de pasajeros

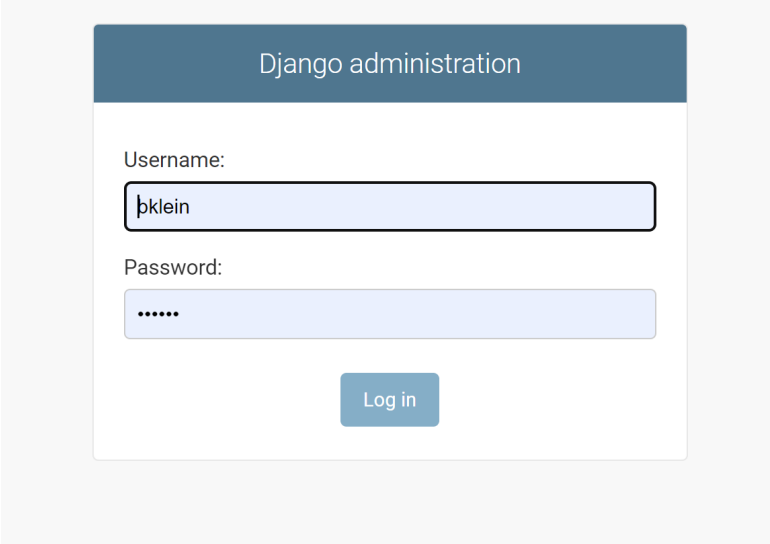
Figura A.2: Mockup de la página de información sobre la topología de la red

Figura A.3: Mockup de la consulta de paradas cercanas

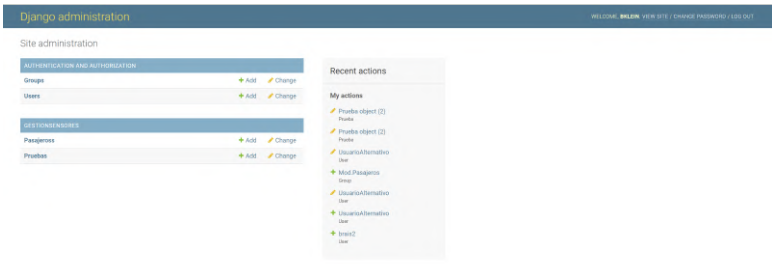
Apéndice B

Imágenes adicionales

En este anexo se mostrarán algunas imágen adicionales que no se ha podido incluir en el contenido por falta de espacio.



(a) Panel de autenticación



(b) Panel de administración

Figura B.1: Página de administración de la aplicación web.

Apéndice C

Transformaciones en PowerBI

En este anexo se mostrarán algunas operaciones de tratamiento de datos realizadas en el software de PowerBI que no se incluyeron en el contenido por falta de espacio.

C.1 Obtención de línea y parada

Para obtener la línea y parada relativos a cada 'click' es necesario hacer una segmentación del campo *Dpaypoint*.

Un ejemplo del formato del campo *Dpaypoint* es 02_L4_P10, siendo el 4 el número de línea y 10 el número de parada.

El proceso de segmentación se dividió en dos partes:

- **Obtención de la línea.** Para llevar a cabo este paso se creó una nueva columna llamada *Línea* a partir de los valores del campo entre la letra 'L' y la '_' :

```
1    = Table.AddColumn(#"Tipo cambiado2", "LINEA", each
     Text.BetweenDelimiters([DPAYPOINT], "L", "_"))
```

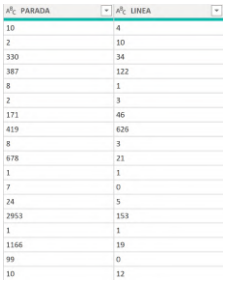
- **Obtención de la parada.** Para llevar a cabo este paso se creó una nueva columna llamada *Parada* a partir de la letra 'P':

```
1    = Table.AddColumn(#"Valor reemplazado", "PARADA", each
     Text.AfterDelimiter([DPAYPOINT], "P"))
```

Gracias este proceso se obtuvo el resultado mostrado en la figura C.1.

C.2 Obtención las columnas "dia de la semana" y "hora"

Para lograr este objetivo se hizo uso de la interfaz visual de la herramienta PowerBI siguiendo los pasos que se describen a continuación:



PARADA	LINEA
10	4
2	10
330	34
387	122
8	1
2	3
171	46
419	626
8	3
678	21
1	1
7	0
24	5
2953	153
1	1
1166	19
99	0
10	12

Figura C.1: Columnas 'línea' y 'parada' en PowerBI.

1. Se abrió el editor de Power Query.
2. Se seleccionó la columna *Fecha*.
3. Se dividió la columna Fecha en dos columnas diferentes: una para la fecha y otra para la hora. Este paso se realizó a través del siguiente código:

```
1    = Table.SplitColumn(#"Columnas quitadas", "FECHA",
     Splitter.SplitTextByDelimiter(" ", QuoteStyle.Csv), {"Fecha",
     "Hora"})
```

4. Se pulsó sobre la opción agregar columna mostrada en el tablero superior de la página de Power Query.
5. Dependiendo del campo que se quería obtener se hizo lo mostrado en la figura C.2, para obtener el día de la semana, y en la figura C.3, para obtener la hora.

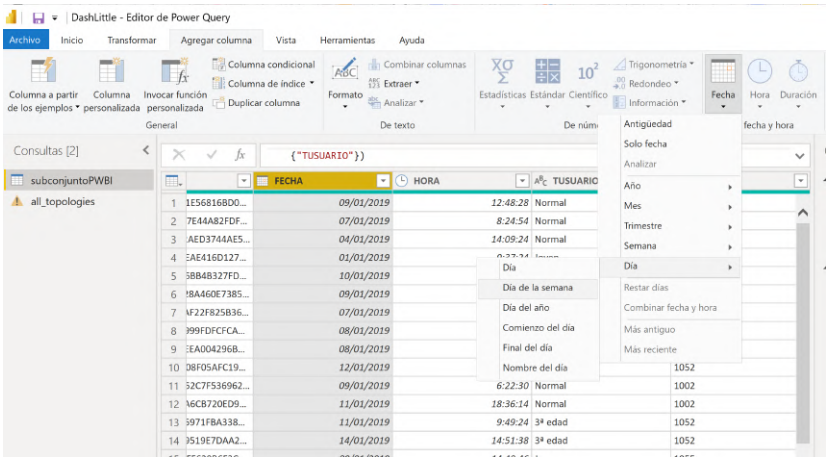


Figura C.2: Obtención de la columna 'día de la semana' en PowerBI

Tras la realización de estos pasos, las columnas resultantes se muestran en la figuras C.4 y C.5.

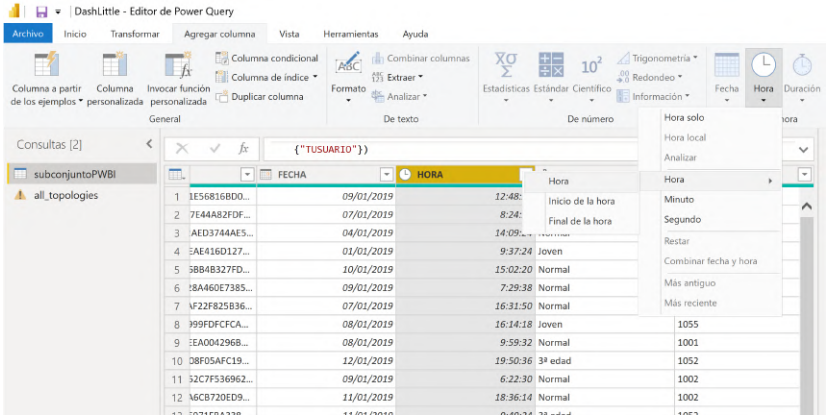


Figura C.3: Obtención de la columna 'hora' en PowerBI

DIASEMANA
miércoles
lunes
viernes
martes
jueves
miércoles
lunes
martes
martes
sábado
miércoles
viernes
viernes
lunes
miércoles

Figura C.4: Columna *DIASEMANA* en PowerBI

Hora.1
12
8
14
9
15
7
16
16
9
19
6
18
9
14
14
7

Figura C.5: Columna *Hora.1* en PowerBI

En la columna relativa a la hora se estableció el nombre *Hora.1* debido a que ya existía una columna llamada *Hora* respectiva a la división realizada en el paso 3 de esta sección. A su vez, se estableció el tipo de datos 'número entero' para el campo anteriormente mencionado.

C.3 Obtención del campo " Laborable"

Con el fin de lograr un campo que permitiese filtrar los datos según si el día es lectivo o no, se creó el campo "LABORABLE". La creación de este campo se logró estableciendo el valor 'laborable' a aquellos días de la semana entre los lunes y los viernes, mientras que se estableció el valor 'no laborable' a los días correspondientes a los fines de semana (sábado y domingo). Este proceso se realizó a través del siguiente código:

```
1    = Table.AddColumn(#"Columnas con nombre cambiado",
     "Personalizado", each (if [DIASEMANA] = "lunes" or [DIASEMANA] =
     "martes" or [DIASEMANA] = "miércoles" or [DIASEMANA] = "jueves"
     or [DIASEMANA] = "viernes" then "Laborable" else "No_Laborable"))
```

El resultado se observa en la figura C.6.

LABORABLE
Laborable
Laborable
Laborable
Laborable
Laborable
Laborable
Laborable
Laborable
Laborable
No_Laborable
Laborable
Laborable
Laborable
Laborable
Laborable
Laborable
Laborable
No_Laborable

Figura C.6: Columna 'laborable' en PowerBI

Apéndice D

Glosario de acrónimos

API *Application Programming Interfaces.*
CSS *Cascading Style Sheets.*
CSV *Comma-Separated Value.*
DAX *Data Analysis Expressions.*
GTFS *General Transit Feed Specification.*
HTML *HyperText Markup Language.*
HTTP *HyperText Transfer Protocol.*
HU *Historias de Usuario.*
IA *Inteligencia Artificial.*
IoT *Intenet of Things.*
JS *JavaScript.*
JSON *JavaScript Object Notation.*
PDCA *Plan-Do-Check-Act.*
REST *Representational State Transfer.*
SGBD *Sistema de Gestión de Bases de Datos.*
SIG *Sistema de Información Geográfico.*
SQL *Structured Query Language.*
TFG *Trabajo Fin de Grado.*
TQM Total Quality Management.
UDC *Universidade Da Coruña.*
URL *Uniform Resource Locator.*

Bibliografía

[1] Ángel, Miguel; Ruiz, Delgado; Angel, Miguel;, "LA HISTORIA DEL BILLETE DE TRANSPORTE PÚBLICO EN MADRID." [En línea]. Disponible en: https://www.madrid.org/bvirtual/BVCM006209.pdf

[2] "Smart City Control Room Dashboards: Big Data Infrastructure, from data to decision support." [En línea]. Disponible en: https://www.snap4city.org/drupal/node/525

[3] Geodan, "Amsterdam Smart City Dashboard | Geodan, s. f." [En línea]. Disponible en: https://www.geodan.com/knowledge-and-innovation/managing-urban-processes-intelligently-with-the-amsterdam-smart-city-dashboard/

[4] Python.org., "Página web de Python," 2023. [En línea]. Disponible en: https://www.python.org/

[5] Ripley, B. D., "The R project in statistical computing." 2001. [En línea]. Disponible en: https://doi.org/10.11120/msor.2001.01010023

[6] "RStudio desktop," 2022. [En línea]. Disponible en: https://posit.co/download/rstudio-desktop/

[7] "Página web de PostgreSQL." [En línea]. Disponible en: https://www.postgresql.org/

[8] Vladimir Agafonkin, "Página web de PostGIS," 2023. [En línea]. Disponible en: http://postgis.net/

[9] Strobl, Christian;, "Página web de Leaflet," 2008. [En línea]. Disponible en: https://leafletjs.com/

[10] "Página web de Django." [En línea]. Disponible en: https://www.djangoproject.com/

[11] "Página web de GeoDjango." [En línea]. Disponible en: https://docs.djangoproject.com/en/4.2/ref/contrib/gis/

[12] Microsoft.com, "Página web de PowerBI." [En línea]. Disponible en: https://powerbi.microsoft.com/

[13] Pydata.org., "Página web de Pandas." [En línea]. Disponible en: https://pandas.pydata.org/

[14] Matplotlib.org., "Página web de Matplotlib." [En línea]. Disponible en: https://matplotlib.org/

[15] Pydata.org., "Página web de Seaborn." [En línea]. Disponible en: https://seaborn.pydata.org/

[16] "Página web de Scipy," 2010. [En línea]. Disponible en: https://scipy.org/

[17] "¿Qué es el Ciclo Planificar-Hacer-Revisar-Actuar?" [En línea]. Disponible en: https://kanbanize.com/es/gestion-lean/mejora-continua/que-es-el-ciclo-pdca

[18] "TQM – Gestión de la calidad total – PDCA Home." [En línea]. Disponible en: https://www.pdcahome.com/tqm/

[19] "Modelo Incremental." [En línea]. Disponible en: http://isw-udistrital.blogspot.com/2012/09/ingenieria-de-software-i.html

[20] "Visual Studio Code - Code Editing. Redefined," 2021. [En línea]. Disponible en: https://code.visualstudio.com/

[21] "GitHub: Let's build from here." [En línea]. Disponible en: https://github.com/

[22] "Overleaf, Online LaTeX Editor." [En línea]. Disponible en: https://www.overleaf.com/

[23] "draw.io - free flowchart maker and diagrams online." [En línea]. Disponible en: https://app.diagrams.net/

[24] Marvel, "Marvel - The design platform for digital products. Get started for free." [En línea]. Disponible en: https://marvelapp.com/

[25] pildorasinformaticas, "Curso Django. Introducción e Instalación.Vídeo 1," 7 2019. [En línea]. Disponible en: https://www.youtube.com/watch?v=7XO1AzwkPPE

[26] I. Pérez Martín, "Tema 6: Django," *ignacio perez martin*, 4 2018. [En línea]. Disponible en: https://perezmartin.es/tema-6-django/

[27] Codercoder, "resolvedPylance error - Reddit." [En línea]. Disponible en: https://www.reddit.com/r/django/comments/noezmy/how_do_you_solve_import_django_filtersviews_could/

[28] "Django no encuentra el módulo de GDAL." [En línea]. Disponible en: https://es.stackoverflow.com/questions/282536/django-no-encuentra-el-m%c3%b3dulo-de-gdal

[29] "API · Leaflet Routing Machine." [En línea]. Disponible en: https://www.liedman.net/leaflet-routing-machine/api/

[30] "GTFS Red de Metro." [En línea]. Disponible en: https://datos.crtm.es/datasets/gtfs-red-de-metro/about

[31] "pandas.read_csv — pandas 2.0.2 documentation." [En línea]. Disponible en: https://pandas.pydata.org/pandasdocs/stable/reference/api/pandas.read_csv.html

[32] P. Pandey, "There is more to 'pandas.read_csv()' than meets the eye," 2022. [En línea]. Disponible en: https://towardsdatascience.com/there-is-more-to-pandas-read-csv-than-meets-the-eye-8654cb2b3a03

[33] "pandas.Series.dt.day_name — pandas 2.0.2 documentation." [En línea]. Disponible en: https://pandas.pydata.org/docs/reference/api/pandas.Series.dt.day_name.html

[34] "pandas.Series.dt.hour — pandas 2.0.2 documentation." [En línea]. Disponible en: https://pandas.pydata.org/docs/reference/api/pandas.Series.dt.hour.html

[35] "pandas.Series.dt.day — pandas 2.0.2 documentation." [En línea]. Disponible en: https://pandas.pydata.org/docs/reference/api/pandas.Series.dt.day.html

[36] M. Film, "rDNS," 11 2018. [En línea]. Disponible en: https://in.pinterest.com/pin/816277501188477497/

[37] J. L. G. Grandes, "¿Por qué debo conocer Django y GeoDjango si hago desarrollos web GIS?" *MappingGIS*, 2023. [En línea]. Disponible en: https://mappinggis.com/2023/03/por-que-debo-conocer-django-y-geodjango-si-hago-desarrollos-web-gis/

[38] B. TechLearn, "How to import excel/text CSV file into Postgres database using Query," 2022. [En línea]. Disponible en: https://www.youtube.com/watch?v=TKjzWfKwF74

[39] T. Bytes, "SOLUCIONAR ERROR POSTGRESQL PERMISO DENEGADO 2022 [TACTICAL BYTE]," 8 2022. [En línea]. Disponible en: https://www.youtube.com/watch?v=DgcdADudRVc

[40] "Which Model Field to use in Django to store longitude and latitude values?" [En línea]. Disponible en: https://stackoverflow.com/questions/30706799/which-model-field-to-use-in-django-to-store-longitude-and-latitude-values

[41] "[u"" value has an invalid format. It must be in YYYY-MM-DD HH:MM[:ss[.uuuuuu]][TZ] format."]." [En línea]. Disponible en: https://stackoverflow.com/questions/36406109/u-value-has-an-invalid-format-it-must-be-in-yyyy-mm-dd-hhmmss-uuuuuu

[42] "pandas.to_datetime — pandas 2.0.2 documentation." [En línea]. Disponible en: https://pandas.pydata.org/docs/reference/api/pandas.to_datetime.html

[43] Minewiskan, "DAX function reference - DAX," 2022. [En línea]. Disponible en: https://learn.microsoft.com/en-us/dax/dax-function-reference

[44] "Django Queries." [En línea]. Disponible en: https://docs.djangoproject.com/en/4.2/topics/db/queries/

[45] "jQuery AJAX get() and post() Methods." [En línea]. Disponible en: https://www.w3schools.com/jquery/jquery_ajax_get_post.asp

[46] "Django." [En línea]. Disponible en: https://docs.djangoproject.com/en/4.2/ref/request-response/

[47] "HTML Form Action: POST and GET (With Examples)." [En línea]. Disponible en: https://www.programiz.com/html/form-action

[48] "Using GeoJSON with Leaflet - Leaflet - a JavaScript library for interactive maps." [En línea]. Disponible en: https://leafletjs.com/examples/geojson/

[49] "How to sort a LatLng pointList in leaflet." [En línea]. Disponible en: https://stackoverflow.com/questions/31254503/how-to-sort-a-latlng-pointlist-in-leaflet

[50] "select two markers draw line between them in leaflet." [En línea]. Disponible en: https://gis.stackexchange.com/questions/53394/select-two-markers-draw-line-between-them-in-leaflet

[51] W. Calcagno, "Insertar un informe Power BI en una página Web," 2019. [En línea]. Disponible en: https://www.youtube.com/watch?v=-CvfdJnhQ80

[52] Pointhi, "GitHub - pointhi/leaflet-color-markers: color variations of the standard leaflet marker." [En línea]. Disponible en: https://github.com/pointhi/leaflet-color-markers