Universitat Oberta de Catalunya

Bahía-Bus

Aplicación web para la gestión de autobuses urbanos

Autor: Cristian González Vélez

Tutor: Jordi Ustrell Garrigos

Profesor: Ferrán Adell Español

Grado Multimedia

Ingeniería Web

3 de enero de 2022

Créditos/Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Bahía-Bus. Aplicación para la gestión autobuses.*
Nombre del autor:	*Cristian González Vélez*
Nombre del colaborador/a docente :	*Jordi Ustrell Garrigos*
Nombre del PRA:	*Ferrán Adell Español*
Fecha de entrega (mm/aaaa):	*01/2022*
Titulación o programa:	*Grado en Multimedia*
Área del Trabajo Final:	*Ingeniería Web*
Idioma del trabajo:	*Español*
Palabras clave	*Autobuses, Movilidad, Tiempo Real, Aplicación web*

Resumen del Trabajo:

El fomento de la movilidad a través del servicio de autobús urbano e interurbano es una manera de apostar por una política de sostenibilidad, bajo coste y solidaridad. Por ello, es preciso contar con herramientas digitales encaminadas a favorecer la accesibilidad de este medio de transporte. Sin embargo, el viajero medio, recurre, principalmente, a medios convencionales impresos en formato analógico, o al contacto vía telefónico con la compañía con el fin de obtener las predicciones de llegada o salida de los autobuses o de sus itinerarios; una opción que sigue siendo válida en pleno 2022, pero que supone una compleja carrera de obstáculos que redunda en una peor gestión del tiempo, y una oportunidad perdida para acercar a la ciudadanía al transporte urbano. Por tanto, se ha propuesto el desarrollo de una aplicación web que informe al usuario en tiempo real sobre la red de autobuses de una localidad concreta de la provincia de Cádiz. Si bien, el proyecto que se describe en este documento no es nuevo, ya que existen algunas aplicaciones similares en otros puntos de España, en lugares como Sanlúcar de Barrameda, ciudad objeto de estudio de este proyecto, con 69.500 habitantes y una red de transporte urbano que cuenta con seis líneas de autobús, este tipo de servicio brilla por su ausencia. Para llevarlo a cabo, el proyecto se apoya en tecnologías web como HTML5, CSS y JavaScript para *backend* y *frontend*, y MongoDB para la persistencia de datos. Con esta aplicación web, se pretende agilizar la forma en la que los usuarios se relacionan con este medio y mantenerlos informados en todo momento.

Abstract:

The promotion of mobility through urban and interurban bus service is a way to bet on a policy of sustainability, low cost and solidarity. Therefore, it is necessary to have digital tools aimed at promoting the accessibility of this means of transport. However, the average traveller resorts mainly either to conventional printed media in analog format, or telephone to contact with the company in order to obtain the predictions of arrival and departure of buses or their itineraries; an option that is still valid in 2022, but it is a complex obstacle course that results in a worse time management, and a missed opportunity to bring citizens closer to urban transport. Therefore, it has been proposed the development of a web application that informs the user in real time about the bus network of a specific town in the province of Cadiz. Although the project described in this document is not new, as there are some similar applications in other parts of Spain, in places like Sanlúcar de Barrameda, the city under study in this project, with 69,500 inhabitants and an urban transport network with six bus lines, this type of service is conspicuous by its absence. To carry it out, the project relies on web technologies such as HTML5, CSS and JavaScript for backend and frontend, and MongoDB for data persistence. With this web application, the aim is to streamline the way in which users interact with this medium and keep them informed at all times.

Dedicatoria

Este trabajo es la culminación de tres duros años de esfuerzo en los que se han producido muchos cambios en mi vida. Por ello, dedico este TFG a todas las personas que me rodean y que me han animado a finalizar este proyecto: padre, madre, hermanos, suegro, suegra y, en especial, mi mujer Ángela y mi hija Alaia.

Agradecimientos

Quiero agradecer a la UOC por estos tres años de enriquecimiento personal.

Notaciones y Convenciones

- **Título principal de sección**: Arial, 20, **Título 1**
- **Título de subsección**: Arial,13, **Título subsección**
- **Título secundario de subsección**: Arial, 13, **Título secundario**
- **Contenido**: Arial, 10, Contenido.
- **Código**: `Consolas, 10, Ejemplo:`

```
const { Schema, model } = require('mongoose');
```

Índice

Figuras y tablas

Índice de figuras

Índice de tablas

Índice de ejemplos de código fuente

1. Introducción

1.1. Introducción

En el PMUS[1] de 2020 de la ciudad de Sanlúcar de Barrameda, se les preguntó a los residentes cuestiones relacionadas con la movilidad en la ciudad. Los sanluqueños respondieron en el 70% de las ocasiones que su medio de transporte predilecto es el automóvil; en segundo lugar, aparece un 13% de ciudadanos que prefiere desplazarse a pie; y en los dos últimos lugares, se encuentran, por un lado, aquellos que utilizan el transporte público urbano, con un 9%, y, por otro, los que emplean la bicicleta, con un 8%.

Figura 1: Autobús de Tranvía.

La política de sostenibilidad de Sanlúcar de Barrameda debe centrarse en reducir el número de medios de transporte con motor de explosión procedentes de vehículos privados, ya que es el principal responsable del crecimiento de emisiones de gases invernaderos en la ciudad, además de otros gases contaminantes que deterioran la calidad del aire como el NO2, PM10 y PM2,5[2], y se debe fomentar el uso del transporte público para reducir el tráfico. Por otro lado, el ruido, los atascos y los accidentes de tráfico son otro problema añadido al exceso de circulación de estos vehículos en la ciudad. (Ayuntamiento de Sanlúcar de Sanlúcar de Barrameda, 2020)

Para revertir esta situación, debe establecerse una política que favorezca el transporte urbano (además del viaje a pie, vehículo eléctrico o bicicleta) y que anime a los sanluqueños a utilizarlo con más frecuencia. El número de viajeros diarios de autobús es de aproximadamente unos 2000, según datos

[1] Plan de Movilidad Urbana Sostenible, es un sistema organizativo de la red viaria promovido por el Ayuntamiento de Sanlúcar con el objetivo de optimizar el coste energético y medioambiental de los desplazamientos en ciudad.

[2] PM2,5 y PM10 son partículas en suspensión de 2,5 y 10 micras o inferior respectivamente.

anteriores a la COVID19. Para mejorar el servicio e incrementar estas cifras, hay que conocer la opinión de la ciudadanía que se recoge en el Plan de Movilidad Urbana Sostenible ya mencionado con anterioridad. En él, los usuarios demandan, entre otros aspectos, aplicar las nuevas tecnologías en el transporte público. Una solución digital dotaría a los viajeros de un mayor control informativo en tiempo real acerca del estado actual de los autobuses urbanos como: la ubicación de la parada más cercana, la del autobús u otras incidencias como posibles retrasos, etc. con el fin de convertirlo en un sistema de transporte eficiente.

El presente documento describe el proceso de desarrollo, así como su funcionamiento, de la solución web Bahía-Bus para resolver todos estos aspectos.

1.2. Descripción

En este documento se describe el desarrollo y puesta en marcha del proyecto web Bahía-Bus en formato escritorio y móvil que gestiona la información del transporte público urbano de la ciudad de Sanlúcar de Barrameda, cuyo servicio se compone de seis líneas de autobuses que cubren 77,2kms en un total de 238 paradas, en tiempo real.

Con el fin de adecuar la demanda de transporte urbano a las necesidades de los ciudadanos, la empresa que gestiona el transporte público en Sanlúcar de Barrameda puso a disposición de sus usuarios una aplicación móvil que sirve para informar acerca de las paradas y líneas de autobuses de la ciudad. Sin embargo, dicha aplicación carece de algunas funcionalidades básicas que la harían más interesante a los pasajeros. Asimismo, recientemente se instaló un sistema de geoposicionamiento en los autobuses urbanos que sin embargo la aplicación no aprovecha para ofrecer la información en tiempo real de la posición de las autobuses, aunque sí del tiempo que resta a un autobús de una línea concreta para llegar a una parada. En este contexto, la aplicación web soluciona las carencias que la aplicación actual del servicio de transporte urbano de la ciudad ofrece, e integra las nuevas tecnologías en los procesos implicados para conseguir que Sanlúcar de Barrameda se convierta en una ciudad inteligente[3].

Características principales de la aplicación:

- Recepción en tiempo real de la posición geográfica de los autobuses, así como su visualización en tiempo real en un mapa a través de iconos representativos.
- Gestión del geoposicionamiento de los autobuses.
- Gestión de las diferentes líneas del transporte urbano diferenciadas por un código de color.
- Gestión de las paradas de cada línea representadas mediante elementos iconográficos en un mapa virtual.
- Obtención por parte del usuario de la parada más cercana a destino, así como la de origen.
- Obtención del itinerario de paradas más directo hasta un destino a través de un buscador.
- Acceso a usuarios para añadir líneas a una lista de favoritos, además de la recepción de notificaciones o alertas.

La tecnología seleccionada para construir la aplicación se basa en JavaScript, y se lleva a cabo mediante el empleo del *solution stack*[4] MERN, que comprende cuatro tecnologías: MongoDB, Express, React y NodeJS. Este *stack* se basa en una arquitectura de capas que ayuda a separar cada una de

[3] Es un concepto que se refiere a un desarrollo urbano basado en la sostenibilidad y que es capaz de responder de manera eficaz a las necesidades de los ciudadanos, empresas e instituciones tanto económicamente como desde el punto de vista ambiental o social.

[4] En informática, es un conjunto de subsistemas o componentes de software necesarios para crear una plataforma completa, de manera que no sea necesario ningún software adicional para soportar las aplicaciones.

las responsabilidades implicadas en el desarrollo, y permite un mayor control para el mantenimiento y escalabilidad de la aplicación:

- Para la persistencia de datos, se ha seleccionado la base de datos MongoDB, de tipo documental[5]. El uso de MongoDB como base de datos garantiza un rápido acceso a datos. Sin embargo, esto puede aumentar la complejidad de la base de datos, ya que se corre el riesgo de no dotarla de la suficiente integridad o de duplicar datos.
- NodeJS es el servidor web en el que correrá nuestra aplicación a nivel de API Rest y seguridad.
- Express es un *framework* de desarrollo sobre NodeJS y que, como consecuencia, corre del lado del servidor, *backend*.
- React es una biblioteca de JavaScript para el desarrollo de aplicaciones interactivas del lado del cliente, o *frontend.*

[5] Al contrario de las bases de datos relacionales, que se organizan en tablas, claves foráneas, relaciones y hacen de la normalización su *modus operandi*, las bases de datos documentales son mucho más flexibles. Los datos se almacenan en documentos y colecciones de datos y no siguen unas normas tan estrictas como en el caso de las bases de datos de tipo relacional.

1.3. Objetivos generales

A continuación se enumeran los objetivos principales y secundarios por orden de importancia que persigue el proyecto Bahía-Bus.

1.3.1. Objetivos principales

Objetivos de la aplicación:

- Gestionar toda la red de transportes urbanos a través de un mapa virtual.
- Visualizar todas las líneas y paradas a través de este mapa virtual.
- Lograr un servicio personalizado mediante la gestión de favoritos.
- Conseguir una interfaz atractiva.
- Lograr que alcance al mayor número de usuarios posible.
- Lograr un servicio eficiente de la aplicación.
- Visualizar el posicionamiento de los autobuses en tiempo real.

Objetivos para el usuario:

- Lograr una buena experiencia de usuario.
- Alejarlo de las prácticas convencionales empleadas para obtener información actualizada del servicio de autobuses.
- Conseguir mantenerle informado acerca de paradas, líneas y autobuses en tiempo real.

Objetivos personales:

- Poner en práctica todos los conocimientos alcanzados durante la carrera.
- Aprender a manejar la tecnología MERN, y, en particular, JavaScript.

1.3.2. Objetivos secundarios

Objetivos adicionales:

- Promover el uso de un servicio de autobús de calidad.
- Modificar la cultura de la movilidad en la vía pública.
- Reducir el uso del vehículo privado con el fin de:
    - Reducir el impacto medioambiental que supone el tráfico en la ciudad (acústico, aire, atascos, etc.)
    - Reducir el peligro del tráfico en la ciudad.
- Lograr un uso más eficiente de la movilidad en la ciudad.

1.4. Metodología y proceso de trabajo

Es un proyecto que parte de cero, aunque arrastra consigo ideas ya contempladas en otras soluciones del mercado. Si bien es cierto que ya existen aplicaciones similares en otros lugares de España, la única aplicación dedicada a informar al usuario del transporte urbano en Sanlúcar de Barrameda, y que pertenece a la empresa[6] que gestiona dichos autobuses, carece de algunas características importantes que la harían más interesante al ciudadano.

La principal fuente de información e investigación es el PMUS, ya que a través de este documento se puede consultar toda la política de movilidad que impera en la ciudad, así como el diseño de la red urbana de transporte, la eficacia y el impacto social, económico y medioambiental que se obtiene con la aplicación del Plan, así como de la opinión de los usuarios a través de encuestas.

Como se trataba de un proyecto que había que finalizar en poco más de tres meses, no representaba riesgo para un negocio real, su desarrollo no requería de un análisis exhaustivo de requisitos, la incertidumbre era baja, y, por otro lado, el objetivo de la aplicación que se presentaba era claro pero la solución, *a priori,* era poco conocida o indeterminada, se propuso el empleo de una metodología de carácter ágil[7] basada en iteraciones incrementales, y cuyos requisitos y soluciones variarían a lo largo del desarrollo, lo que daría como resultado una solución flexible que aceptaría cambios en los requisitos aunque tuviesen lugar en etapas muy avanzadas del proyecto.

Por otro lado, la tecnología empleada MERN es la primera vez que iba a ser abordada por el autor del proyecto, por lo que su proceso de aprendizaje se ha llevado a cabo de forma paralela al desarrollo del propio trabajo. Además, como la solución divide cada área de responsabilidad en una capa diferente, el proceso se dividió en dos partes fundamentales: *backend* y *frontend*.

- Durante el desarrollo del *backend* de la aplicación, se trabajaron tres tecnologías diferentes: NodeJS, Express y MongoDB, que facilitan la puesta en marcha de una API Rest de acceso privado tanto para usuarios como para los dispositivos GPS que deban conectarse en remoto a la aplicación através de *websockets*.
- Por su parte, el *frontend* se construyó en HTML5, CSS y JavaScript con la ayuda de ReactJS, y es el principal encargado de consumir la API Rest y de imprimir los datos en pantalla.

[6] Grupo Avanza.

[7] El Manifiesto Ágil sigue cuatro principios fundamentales: 1. Individuos e interacciones sobre procesos y herramientas. 2. *Software* funcionando sobre documentación extensiva. 3. Colaboración con el cliente sobre negociación contractual. 4. Respuesta antes el cambio sobre seguir un plan.

- El desarrollo de la versión móvil y web se acometió de forma paralela, ya que el *framework* de diseño de estilos CSS para *frontend* de Materialize[8] facilitó mucho la tarea.

En este sentido, tanto *backend* como *frontend* se desarrollaron en paralelo mediante iteraciones. Al final de cada una, se obtuvo un producto funcional al que se le añadieron características en función de su evolución. Para organizar cada una de las iteraciones, se hizo uso del *software* de Trello[9], a través del cual se aplicó una metodología próxima a la filosofía KANBAN[10].

Por último, durante el desarrollo, se contó con un gestor de versiones, Git, y un repositorio en github.com.

[8] Materialicecss es un *framework* para diseño con hojas de estilo basado en Material Design y creado por Google para aplicaciones móviles y web.

[9] Software para la gestión de proyectos (www.trello.com)

[10] KANBAN, letrero en japonés, surgida en Toyota Production System (TPS), es una metodología de trabajo ágil basada en la división de tareas separadas por estados a lo largo de una pizarra. En la metodología KANBAN, cada iteración se divide en tres etapas como mínimo: *por hacer*, *en curso* y *finalizada*, y en cada una de ellas, se añade un proceso escrito en una tarjeta.

1.5. Planificación

		Dedicación		
		Duración	Inicio	Final
Tareas	**PEC1**	**5 días**	**15/09/21**	**19/09/21**
	Propuesta de TFG	5	15/09/21	19/09/21
	PEC2	**14 días**	**20/09/21**	**03/10/21**
	Redacción Memoria	6 días	20/09/21	25/09/21
	Estudio *frontend*	4 días	26/09/21	29/09/21
	Estudio *backend*	4 días	30/09/21	03/10/21
	PEC3	**28 días**	**04/10/21**	**31/10/21**
	Redacción Memoria	18 días	04/10/21	21/10/21
	Requisitos	4 días	08/10/21	11/10/21
	Diseño BD	5 días	12/10/21	16/10/21
	Progr. *backend*	10 días	17/10/21	26/10/21
	Progr. *frontend*	15 días	12/10/21	26/10/21
	PEC4	**35 días**	**01/11/21**	**05/12/21**
	Redacción memoria	35 días	01/11/21	05/12/21
	Pruebas	15 días	01/11/21	15/11/21
	PEC5	**29 días**	**06/12/21**	**16/12/21**
	Finalizar memoria	11 días	06/12/21	16/12/21
	Despliegue	3 días	06/12/21	08/12/21
	Pruebas	18 días	17/12/21	03/01/22
	Defensa virtual	**5 días**	**10/01/22**	**14/01/22**

Tabla 1. Planificación de tareas.

1.5.1. Diagrama de Gantt y Diagrama de Tareas

	Nombre	Duracion	Inicio	Terminado
1	**PEC1**	**5 days**	**15/09/21 9:00**	**19/09/21 19:00**
2	Propuesta de TFG	5 days	15/09/21 9:00	19/09/21 19:00
3	**PEC2**	**14 days**	**20/09/21 9:00**	**3/10/21 19:00**
4	Redacción de la memoria	6 days	20/09/21 9:00	25/09/21 19:00
5	Estudio del FrontEnd	4 days	26/09/21 9:00	29/09/21 19:00
6	Estudio del Backend	4 days	30/09/21 9:00	3/10/21 19:00
7	**PEC3**	**28 days?**	**4/10/21 9:00**	**31/10/21 19:00**
8	Redacción de la memoria	18 days	4/10/21 9:00	21/10/21 19:00
9	Requisitos	4 days	8/10/21 9:00	11/10/21 19:00
10	**Backend**	**15 days?**	**12/10/21 9:00**	**26/10/21 19:00**
11	Diseño de la base de d...	5 days?	12/10/21 9:00	16/10/21 19:00
12	Programación	10 days	17/10/21 9:00	26/10/21 19:00
13	**Frontend**	**20 days?**	**12/10/21 9:00**	**31/10/21 19:00**
14	Programación	15 days?	12/10/21 9:00	26/10/21 19:00
15	Diseño Visual	5 days	27/10/21 9:00	31/10/21 19:00
16	**PEC4**	**35 days?**	**1/11/21 9:00**	**5/12/21 19:00**
17	Redacción de la memoria	35 days?	1/11/21 9:00	5/12/21 19:00
18	Pruebas	15 days	1/11/21 9:00	15/11/21 19:00
19	**PEC5**	**29 days?**	**6/12/21 9:00**	**3/01/22 19:00**
20	Finalizar memoria	11 days?	6/12/21 9:00	16/12/21 19:00
21	Despliegue del proyecto	3 days	6/12/21 9:00	8/12/21 19:00
22	Pruebas	18 days	17/12/21 9:00	3/01/22 19:00
23	Defensa Virtual	5 days	10/01/22 9:00	14/01/22 19:00

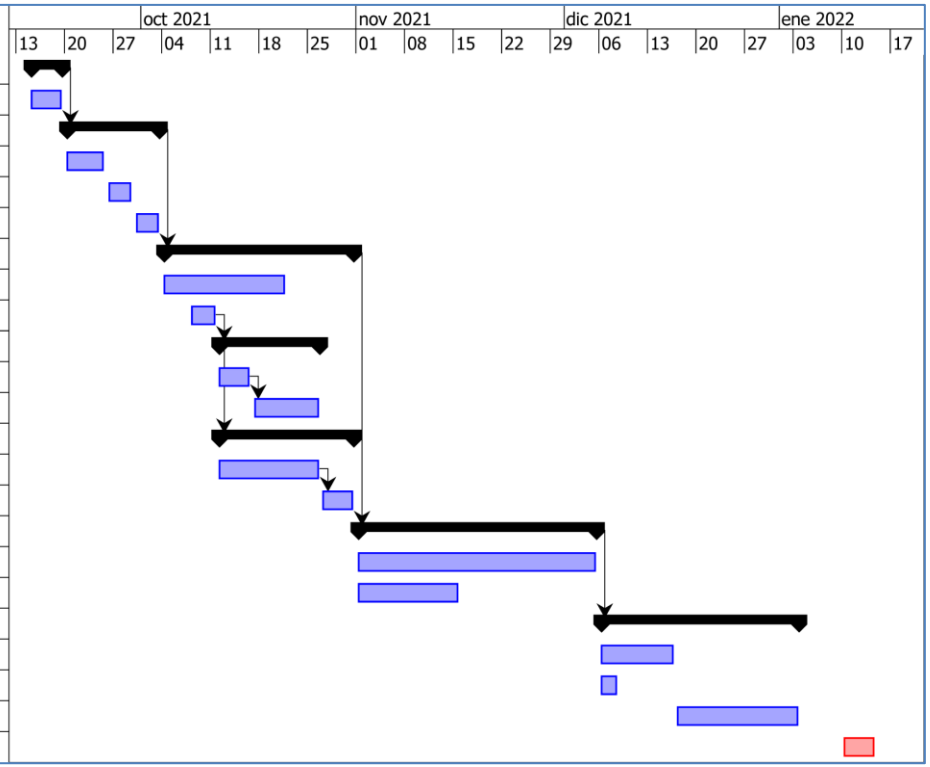


Figura 2. Diagrama de Gantt.

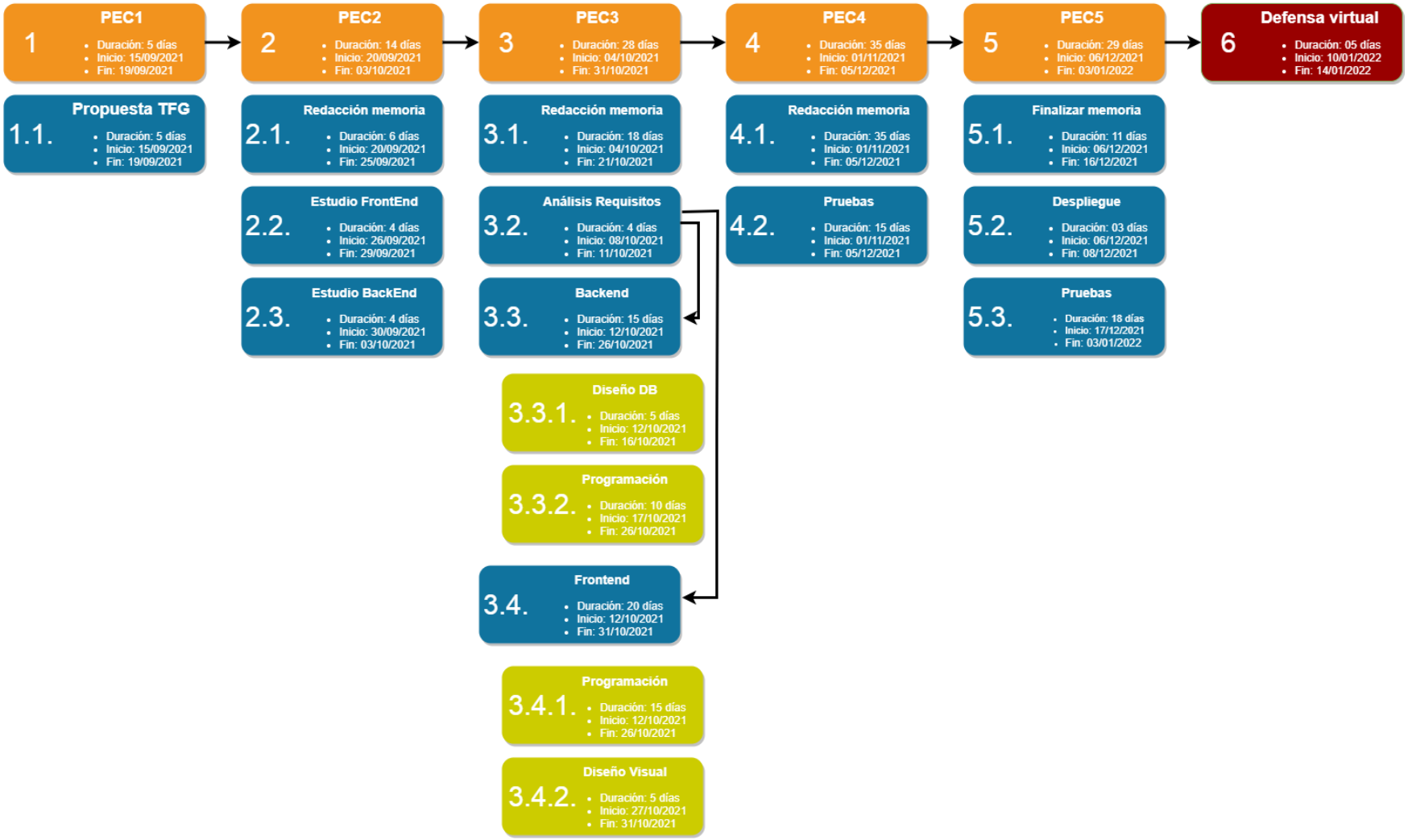


Figura 3. Diagrama de tareas.

1.6. Presupuesto

	Horas	€/hora	Subtotal
Fase de Análisis			
Recopilación de requisitos	32	35€	1.120€
Desarrollo			
Programación *frontend*	15 días	35€	525€
Programación *backend*	10 días	35€	350€
Diseño de base de datos	5 días	35€	175€
Diseño visual			
Diseño gráfico	1 días	20€	20€
Diseño de contenidos	1 días	15€	15€
Maquetación	3 días	30€	90€
Pruebas			
Tests unitarios	33 días	35€	1.155€
Producción			
Despliegue	3	35€	105€
Total			**3.555€**

Tabla 2. Presupuesto para recursos humanos.

	Precio
Hosting	
Plan Free and Hobby	Gratis
API de terceros	
MapboxGL	Gratis primeras 50.000 impresiones o solicitudes al mes
Base de datos	
MongoDB Atlas	Gratis, hasta 512 MB de almacenamiento y RAM compartida
Total	**gratis**

Tabla 3. Presupuesto de recursos técnicos durante el despliegue.

1.7. Estructura del resto del documento

A continuación se resumirá brevemente cada uno de los contenidos que aparecen en el resto de apartados de este documento:

- **Análisis**: Se estudia la situación actual del mercado en relación a la existencia de aplicaciones similares en el repositorio de Google Play así como posibles competidores. Asimismo, se analiza el público objetivo de esta aplicación y el perfil de usuario de la misma. Finalmente, a través de un análisis DAFO, se valoran las debilidades y fortalezas de Bahía-Bus.

- **Propuesta**: A partir del análisis anterior, se definen los objetivos de la aplicación.

- **Diseño**: Se estudia la arquitectura de la información, de su navegación, de la integración del diseño con la aplicación y de la usabilidad. Se explica el lenguaje utilizado, su funcionamiento y su motivación. Por otro lado, se describen las partes más interesantes del proyecto relacionadas con el *backend* y el *frontend,* así como las bibliotecas de las que dependen. En el *frontend* se describen los componentes definidos. También se tratan dos casos de uso críticos en la aplicación.

- **Implementación**: Se detallan los requisitos de instalación, así como las instrucciones de instalación de la aplicación.

- **Demostración**: Se explican las instrucciones de uso de la aplicación, y se muestran los diferentes prototipos desarrollados durante la etapa de elaboración.

- **Conclusiones**: Se desarrollan las últimas conclusiones del trabajo y su perspectiva de futuro en cuanto a nuevas funcionalidades para la aplicación, mejoras de rendimiento o estrategias para abaratar sus costes.

2. Análisis de mercado

2.1. Público objetivo y perfiles de usuario

El público potencial con el que cuenta la aplicación se divide en dos grupos. Por un lado, el de los usuarios, que es un grupo muy diverso, ya que el servicio urbano de autobuses reúne una gran variedad de tipologías de viajeros clasificados según los centros de generación de desplazamiento y de atracción de la movilidad en la ciudad. En un primer lugar, se encuentran los viajeros de movilidad obligada, que se relacionan con el desplazamiento de trabajadores y estudiantes condicionados por el horario de entrada y salida de sus centros de destino. En este grupo, existen tanto personas con edad de trabajar (18 años en adelante) como aquellas que hacen uso del servicio por motivos educativos (transporte escolar en la mayoría de casos). En segundo lugar, existe un tipo de viajero de movilidad no obligada por algunos de los siguientes motivos: administrativos, comerciales, culturales, deportivos, zonas verdes y centros de culto o interés turístico. Son viajeros cuyas edades oscilan desde los doce años en adelante. En definitiva, los únicos requisitos que debe cumplir este grupo son los de poseer una experiencia mínima con las nuevas tecnologías y la necesidad de desplazarse en ciudad.

Por otro lado, en la parte administrativa de la aplicación, el verdadero cliente, se encuentra la empresa de transporte urbano de autobuses de Sanlúcar de Barrameda, que desea añadir un servicio informativo y automático a través de una solución digital. El perfil administrador de la aplicación debe ser más técnico que el del usuario ordinario, ya que requiere algunos conocimientos mínimos para tareas como: la gestión del mapa (en cuanto al establecimiento de rutas, líneas, paradas, autobuses, usuarios, etc.), gestión de usuarios (modificaciones o bajas), etc. Por lo tanto, la edad del administrador de la aplicación debe ser de 18 en adelante.

2.2. Competencia/Antecedentes

La empresa que gestiona los autobuses de Sanlúcar de Barrameda, Grupo Avanza, ya cuenta en la actualidad con una aplicación móvil a la que definen en Google Play Store[11] como: “aplicación para la consulta de horas de paso”. No existe en la actualidad otra aplicación específica para la gestión de horarios y líneas urbanas en esta ciudad, ya que estas suelen ser aplicaciones diseñadas a medida por la compañía que se hace cargo del transporte urbano de la ciudad interesada.

A continuación, se analizan varias aplicaciones móviles y web que existen en el mercado. Primero, se menciona la que opera en Sanlúcar de Barrameda, a continuación, se analizan aplicaciones de otros entornos geográficos y, por último, se estudia una de ámbito más generalista.

[11] La aplicación, BusApp Sanlúcar, se puede encontrar en Play Store en el siguiente enlace: https://play.google.com/store/apps/details?id=com.busmatick.carlos.busappsanlucar (Recuperado el 11 de octubre de 2021)

2.2.1. Aplicación actual: BusApp Sanlúcar

La aplicación del Grupo Avanza BusApp Sanlúcar solo está disponible para *smartphones* con sistema operativo Android y reúne las siguientes funcionalidades:

- Líneas y paradas: se muestran las diferentes líneas urbanas que existen. En ellas se refleja el número de línea, el nombre (definido por origen y destino), sus paradas, la hora estimada de llegada de cada autobús que cubre dicha parada, y el acceso a la ubicación de esta mediante icono gráfico en Google Maps. También otorga la posibilidad de compartir parada a través de aplicaciones o redes sociales, y de navegar a través de la calle donde se ubica mediante Google Street View. Además, da la posibilidad de descargar un archivo PDF del mapa de la línea con todas sus paradas. Por último, existe la posibilidad de añadir la parada a un apartado de favoritos que se denomina "Paso por parada". En la vista de parada se visualiza el tiempo estimado (basado en tiempo real) de llegada del autobús a una parada.
- Mapa: ubicación de todas las paradas en mapa. Se trata de un mapa interactivo basado en Google Maps desde el que se pueden observar las paradas de todas las líneas de autobús de la ciudad. Al clicar encima de una de ellas, se observa un *bocadillo* emergente con el número de líneas que la cubren.
- Paso por parada: paradas almacenadas en favoritos, acceso directo a ellas y a todas las funcionalidades ya comentadas con anterioridad.

2.2.2. Aplicaciones similares en otros ámbitos geográficos

Bus Cádiz.

Bus Cádiz[12] es una aplicación que, "permite obtener toda la información de los autobuses urbanos de la ciudad de Cádiz". Está disponible tanto en sistemas operativos Android como IOS. Según detalla el autor en Google Play Store, "gracias a los nuevos servicios de geolocalización instalados en los autobuses de la ciudad, te permite saber cuánto tiempo tardará el autobús en llegar a tu parada". Sin embargo, esto no es cierto porque la aplicación extrae la información de la web de la empresa Tranvía de Cádiz a San Fernando y Carraca S.A[13]., que basa el tiempo de llegada de los autobuses en horarios preestablecidos.

[12] Bus Cádiz se puede obtener en Play Store desde el siguiente enlace: https://play.google.com/store/apps/details?id=es.danielramirez.autobusesdecadiz (Recuperado el 11 de octubre de 2021)

[13] Se puede acceder desde el siguiente enlace: https://tranviadecadizasanfernandoycarraca.es/ (Recuperado el 11 de octubre de 2021).

Bus Cádiz tiene un diseño muy visual y cuenta con características parecidas a la aplicación BusApp Sanlúcar:

- Integración con Google Maps.
    - Visualización en mapa de puntos de venta de tickets con iconos gráficos.
    - Visualización de paradas con iconos gráficos.
    - Visualización de recorridos de líneas de autobús diferenciadas por colores.
- Tiempo de llegada a parada de un autobús de línea urbana.
- Consulta de paradas.
- Apartado de favoritos.
- Apartado de tiempo meteorológico.

Autobuses Urbanos El Puerto de Santa María.

La aplicación del servicio de transporte Colectivo Urbano de El Puerto de Santa María[14] añade, además de las características mencionadas con anterioridad para la gestión de líneas, paradas y horarios por parte del usuario y, lo más llamativo: localización en tiempo real de los autobuses. Solo está disponible para dispositivos con sistema operativo Android.

En vistas generales, la aplicación cuenta con:

- Integración con Google Maps:
    - Recorrido de cada línea con código de color.
    - Ubicación de paradas con iconos gráficos.
    - Ubicación en tiempo real de los autobuses en una línea concreta en mapa y mediante iconos gráficos.
- Horarios.

Por otro lado, el servicio cuenta con una página web[15] en la que se puede extraer la misma información, aunque da la posibilidad de aplicar filtros de líneas y horarios. En cualquier caso, la aplicación móvil parece más actualizada en la fecha en la que se ha escrito este documento, ya que la web dispone de un diseño poco navegable y no adaptativo, por lo que la mayoría de usuarios optará por la otra versión.

[14] Se puede descargar en Google Play Store desde: https://play.google.com/store/apps/details?id=es.autobusesdelpuerto.app&hl=es&gl=US (Recuperado el 11 de octubre de 2021).

[15] La web se puede consultar en http://elpuertodesantamaria.interbus.es:9098/ (Recuperado el 11 de octubre de 2021)

BilboBus.

Está disponible tanto para IOS como para aquellos *smartphones* que basan el sistema operativo en Android. Se puede instalar de manera gratuita desde Apple Store o Google Play Store respectivamente, aunque también es accesible desde la web[16]. Se trata de una aplicación muy completa, pues ofrece más funcionalidades que las comentadas en las aplicaciones anteriores:

- Integración con Google Maps.
    - Visualización de paradas mediante iconos gráficos.
    - Visualización de recorridos de líneas de autobús diferenciadas por colores.
    - Visualización de la ubicación en tiempo real de los autobuses mediante icono gráfico.
- Tiempo de llegada a parada de un autobús de línea urbana.
- Consulta de paradas. Detecta las paradas más cercanas.
- Apartado de favoritos para líneas o paradas.
- Avisos configurables según línea.
- Última hora. Un acceso directo al Twitter de BilboBus para conocer las novedades e incidencias de última hora.
- Noticias. Un apartado para anunciar todos los eventos próximos relacionados con el servicio de autobús.
- Tarifas. Un apartado informativo para conocer los precios de las diferentes modalidades de billetes.
- Ciudadanía. Que incluye:
    - Objetos perdidos. Acceso directo al teléfono de la compañía que gestiona los objetos perdidos en los autobuses.
    - Buzón de sugerencias, que incluye un formulario.
    - Cambiar idioma. Ofrece: español, euskera e inglés.
    - Normativa. Instrucciones para la regulación de los derechos y obligaciones de los viajeros.

2.2.3. Un caso de aplicación más generalista: Moovitapp

Moovit es una solución de Movilidad como servicio (MaaS[17]). Moovit, propiedad de Intel desde 2020, es, sin duda, la más potente y completa aplica-

[16] Se puede acceder a su versión *responsive* desde el siguiente enlace: https://www.bilbao.eus/bilbobusapp/lines.html (Recuperado el 11 de octubre de 2021)

[17] MaaS define un alejamiento de la modalidad de transporte personal en vehículo privado hacia soluciones de movilidad que se usan como servicio, y promueve la combinación de diferentes tipos de medios de transporte, público o privado según la necesidad de viaje de los usuarios. (https://es.wikipedia.org/wiki/Movilidad_como_un_servicio , Recuperado el 12 de octubre de 2021)

ción sobre movilidad de transporte público. Toda la información que gestiona Moovit ha sido recopilada por los usuarios a través de *crowdsourcing*[18]. Moovit ofrece una gran cantidad de servicios como:

- Aplicación de marca, para integrar la marca de una empresa en la aplicación.
- Análisis de movilidad urbana.
- Pago de billetes.
- Bajo demanda.
- Tiempo real.
- API de transporte público.
- Gestor de datos estáticos.

Moovitapp está disponible tanto en formato web como móvil[19] y ofrece una interfaz gráfica basada en MapboxGL y OpenStreetMap desde la que se pueden planificar rutas señalando origen y destino deseado. Además, pone el servicio a disposición de cualquier usuario a nivel planetario.

2.2.4. Tabla comparativa entre aplicaciones

	BusApp	**Bus Cádiz**	**AUEPSM**	**BilboBús**	**MoovitApp**
Para Sanlúcar	Sí	No	No	No	Sí, pero no es exclusiva de la localidad.
Integración con API de mapas	Sí	Sí	Sí	Sí	Sí
Información en tiempo real	Sí	No. Las estimaciones de llegada las extrae de la página de Tranvía.	Se conoce la ubicación de los autobuses. El tiempo de espera no lo da aunque parece que alguna vez lo dio.	Se conoce la ubicación de los autobuses pero hay que refrescar. También las horas de llegada.	Horas de llegada en tiempo real.

[18] Es una forma de trabajo colaborativo y abierto llevado a cabo por una comunidad grande de usuarios.

[19] Se puede acceder desde https://moovitapp.com/ (Recuperado el 12 de octubre de 2021)

Consultar horarios	Existe pero actualmente no funciona.	Solo los horarios más cercanos a la hora actual.	Sí	Sí	Sí
Pago de billetes	No	Ofrece ubicación de establecimientos.	No	No. Ofrece los precios.	Sí
Diferenciación de líneas por código de color sobre mapa	No	Sí	Sí	Sí	Sí
Buscador de rutas	No	No	No	No	Sí
Variedad de rutas con medios de transporte	Solo autobús	Solo autobús	Solo autobús	Solo autobús	Sí
Paradas cercanas	No	Sí	No	Sí	Sí
Favoritos	Sí	Sí	Sí	Sí	Sí
Cambio de idioma	No	No	No	Sí	Sí
Apartado de avisos	No	No	No	Sí	Sí
Versión web	No	No	Sí	Sí	Sí

Tabla 4. Comparación entre aplicaciones dedicadas a la movilidad.

2.3. Análisis DAFO

Fortalezas:

- Tecnología web flexible, escalable y adaptable a los cambios.
- No sujeta a ningún sistema operativo en particular.
- Al tratarse de una solución SaaS (*Software As A Service*)[20], no necesita ser instalado. Funciona desde el navegador.

Debilidades:

- El programador empezó a conocer la tecnología empleada a medida que desarrollaba la aplicación, por lo que se corre el riesgo de obtener un producto poco eficiente o muy básico.
- Dependencia de un hardware ajeno: los GPS que incorporan los autobuses, que podrían no funcionar correctamente.
- Dependencia de una API externa que provee servicios de búsqueda de itinerarios e impresión de mapas en pantalla.

Amenazas:

- Adaptación de otras aplicaciones similares al caso particular de Sanlúcar de Barrameda.
- Solo cubre un nicho de mercado que, además, está vinculado al sector público.

Oportunidades:

- Aunque se trata del único cliente posible para explotar esta aplicación en ámbitos urbanos, la Administración Pública, a través de la empresa adjudicada para llevar a cabo la gestión de los autobuses, es sinónimo de garantía.
- Pocos competidores locales. Solo existe una aplicación orientada a la movilidad de la ciudad y está administrada por la empresa que gestiona el transporte urbano.
- Posibilidad de expandir su ámbito a otros modelos de transporte: tren, taxis, etc. O de extenderlo a servicios de autobuses interurbanos, donde la administración pública no es tan rígida.

[20] Más información acerca de SaaS en https://es.wikipedia.org/wiki/Software_como_servicio (recuperado el 2 de enero de 2022)

3. Propuesta

Bahía-Bus pretende adecuar el transporte urbano a las necesidades de los ciudadanos sanluqueños. El Ayuntamiento de Sanlúcar de Barrameda gestiona la movilidad a través de la empresa denominada Grupo Avanza, que pone a disposición de los ciudadanos la aplicación BusApp SanLúcar, solo disponible en Android, y que se puede descargar desde Google Play Store. A través de BusApp Sanlúcar, los usuarios acceden a las líneas de autobuses, y pueden navegar entre sus paradas mediante la API de Google Maps, así como comprobar la hora de llegada de los autobuses en tiempo real. Sin embargo, esta aplicación carece de funcionalidades básicas como:

- Buscador de destinos. Es decir, un buscador que le dé la posibilidad al viajero de encontrar el autobús más adecuado para llegar a la parada de destino.
- Buscador de la parada más cercana. Se puede navegar a través del mapa para visualizar dónde se ubica una parada concreta, pero no calcula cuál es la más cercana con respecto a la ubicación actual del usuario.
- Líneas con rutas diferenciadas por colores sobre el mapa. La aplicación informa al usuario sobre las paradas que tienen las líneas, pero no facilita, de una manera visual, el recorrido que realiza un autobús en una línea concreta, por lo que pierde valor en cuanto a usabilidad.
- No visualiza los autobuses en el mapa en tiempo real, aunque sí muestra la hora estimada a la que llega el autobús. Recientemente se instaló un sistema de geoposicionamiento en los autobuses urbanos que sin embargo la aplicación no aprovecha para ofrecer información en tiempo real de la posición de las autobuses.
- No recibe incidencias. No informa al usuario sobre incidencias en tiempo real como: cancelaciones de paradas, retrasos por diversos motivos como obras en la vía, etc.
- No dispone de un apartado en el que se informe sobre el precio del autobús.
- La aplicación no es multiplataforma. Solo existe una versión para *smartphones* basados en Android.

En este contexto, la aplicación web soluciona alguna de estas carencias, e integra las nuevas tecnologías en los procesos implicados para conseguir que Sanlúcar de Barrameda se convierta en una ciudad inteligente.

3.1. Definición de objetivos/especificaciones del producto

A continuación se describen los objetivos principales de Bahía-Bus según los distintos perfiles de usuario:

3.1.1. Objetivos desde el punto de vista del usuario de la aplicación

La aplicación debe satisfacer una serie de requisitos para cada uno de los siguientes puntos:

1. Búsqueda
    a. Encontrar el itinerario para alcanzar un destino especificado.
2. Paradas
    a. Visualizar todas las paradas de una línea.
    b. Buscar parada de destino y visualizar la línea que necesita tomar para llegar a ella.
    c. Buscar la parada más cercana con respecto a la ubicación del usuario en el mapa.
3. Líneas
    a. Visualizar líneas con códigos de color en mapa.
    b. Añadir línea a favoritos.
4. Autobuses
    a. Visualización de la ubicación de los autobuses en tiempo real sobre mapa virtual.
    b. Incidencias en tiempo real de un autobús.
5. Registro.
    a. Dar de alta al usuario.
    b. Iniciar sesiones seguras de usuario.
    c. Modificar el perfil.
6. Versión web y móvil.

3.1.2. Objetivos desde el punto de vista del administrador de la aplicación

Mostrar un área privada para gestión administrativa del sitio desde la que se llevarán a cabo las siguientes funciones:

1. Gestión de las líneas de autobús.
    a. Altas, bajas y modificaciones.
    b. Añadir el color que corresponda a la línea.
    c. Añadir periodos de operatividad.
2. Añadir autobuses
    a. Altas, bajas y modificaciones.
    b. Asignar un autobús a una línea.
3. Añadir rutas.
    a. Altas, bajas y modificaciones.
    b. Asignar ruta a líneas.
4. Gestión de paradas de autobuses y la vinculación existente con cada una de las líneas de autobús.
    a. Altas, bajas y modificaciones.
    b. Asignar su ubicación.
    c. Asignar paradas y líneas con las que tiene conexión cada una de ellas.
5. Gestión de usuarios
    a. Altas, bajas y modificaciones.
    b. Sesiones de usuario seguras.
6. Notificaciones.
    a. Añadir notificaciones de línea o autobús.

4. Diseño

4.1. Arquitectura general de la aplicación

La aplicación se divide en *backend* y *frontend*:

4.1.1. *Backend*

El *backend,* por un lado, es responsable de poner una API a disposición de la aplicación. El *backend* se divide en controladores que atienden las peticiones de la aplicación a través de enrutadores (que harían la labor de un *Page Controller*[21]) y las deriva a los controladores para atender el caso de uso correspondiente. Los modelos se encargarán de llevar a cabo la lógica del negocio en el *backend* y de solicitar los datos a la capa de persistencia: la base de datos. En este sentido, la API se maneja desde la URL a través de diferentes *endpoints*[22] que acceden a los servicios que se solicitan.

Otra de las funciones que cumple el *backend* es la de abrir un canal de comunicación orientado a conexión entre los clientes y este mediante *websockets* para llevar a cabo las funciones en tiempo real de la aplicación. Los clientes establecen un puente de conexión con el *backend* para actualizar su información, y, seguidamente, el *backend* se encarga de repartir la información entre todos los clientes que permanezcan a la escucha en dicho canal. Esta funcionalidad la aprovechan los autobuses cuando van a actualizar su posición o a registrar incidencias durante el servicio.

El *backend,* también se encarga de gestionar las sesiones de usuario y las validaciones de *tokens*[23] (con los que el sistema identifica la validez de la sesión de usuario). Algunos de los *endpoints* de la API están protegidos en función de dos criterios:

- De sesión: si el usuario está *logueado* o no en la aplicación,
- De rol de usuario: en función de la categoría (administrador, usuario o autobús) que ostente la cuenta de usuario que solicita los datos.

Por último, el *backend* también gestiona la conexión con la base de datos a través de los modelos definidos en el dominio. Los modelos son instancias de los documentos almacenados en las colecciones que fueron previamente definidas al crear la base de datos.

[21] Mediante el patrón *Page Controller* «hay un controlador de entrada para cada página lógica del sitio web. Ese controlador puede ser la página en sí, como suele ser en entornos de página de servidor, o puede ser un objeto separado que corresponde a esa página» (Fowler, 2011).

[22] Un *endpoint* describe una solicitud HTTP de cliente de tipo: POST, GET, UPDATE o DELETE, que gestiona un controlador y que invoca la función o método correspondiente con la lógica necesaria para generar una respuesta.

[23] Según Wikipedia: “En los sistemas informáticos, un *token* de acceso contiene las credenciales de seguridad para una sesión de inicio de sesión e identifica al usuario, los grupos de usuarios, los privilegios del usuario y, en algunos casos, una aplicación en particular.” https://en.wikipedia.org/wiki/Access_token (Recuperado el 30 de octubre de 2021)

4.1.2. *Frontend*

El *frontend* se encarga de mostrar en pantalla el resultado de las solicitudes de datos demandadas al *backend*. Al igual que el *backend*, también gestiona cierta parte de la lógica del negocio de estos datos, ya que funciona como la vista del sistema y se muestra al usuario en función de criterios como: sesión de usuario o *endpoints* solicitados. Al tratarse de una SPA (Single Page Application), el *frontend* también ofrece servicios de enrutamiento con el fin de no tener que recargar la página para mostrar las vistas actualizadas.

El *frontend* está enfocado a componentes reutilizables, y todo el estado de esta capa está centralizado y gestionado desde Redux[24]. El *frontend* también se encarga de gestionar el comportamiento de todos los componentes que aparecen en pantalla y de darles estilos.

Se divide en dos apartados diferentes según el nivel de usuario: Administrador y Usuario convencional. El área de administración controla las funciones básicas CRUD[25] de autobuses, líneas, paradas, notificaciones, rutas y usuarios. El área de usuario es el área de servicios que ofrece la aplicación.

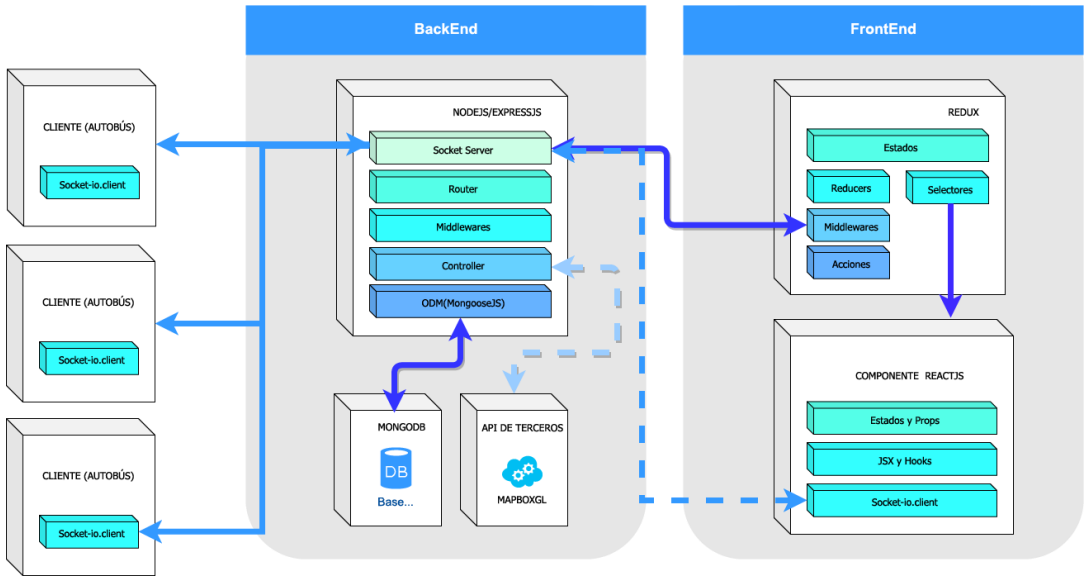


Figura 4. Esquema general de la aplicación.

[24] Redux es una biblioteca de JavaScript encargada de desacoplar el estado de la aplicación y los componentes. Se basa en tres principios: 1. Una única fuente de datos: el *Store* 2. Los estados almacenados son solo de lectura (inmutables) y se modifican a través de acciones 3. Los cambios se realizan mediante *reducers*, que son funciones puras, es decir, no pueden tomar datos externos o invocar otras funciones. 4. Solo funciones síncronas. Para utilizar asincronía se necesitan los *middlewares*. Se puede consultar en https://redux.js.org (Recuperado el 29 de noviembre de 2021).

[25] CRUD son iniciales que en jerga informática resume las labores: *Create* (crear), *Read* (leer), *Update* (actualizar), *Delete* (borrar).

4.1.3. La base de datos

La base de datos de Bahía-Bus no es relacional. Este tipo de base de datos está alejada del concepto tradicional que poseen las bases de datos relacionales en cuanto a normalizaciones y reglas estrictas de integridad. Una de las características principales de esta base de datos es que el formato con el que trabaja es JSON, o BJSON (Binary JSON). Para llevar a cabo la gestión de la base de datos, se ha utilizado Mongoose, una biblioteca de JavaScript que actúa como ODM (*Object Document Mapping*) que exige la creación de *schemas* en la base de datos para manejar las entidades de una manera más predecible y segura, y que define la estructura y los tipos de los documentos que se almacenan en las colecciones. Mongoose, además, también valida el formato de los campos que las componen.

A continuación, se muestra un diagrama de clases UML que describe de una manera aproximada cómo se relacionan las colecciones, según el *schema* de cada una de ellas, en la base de datos:

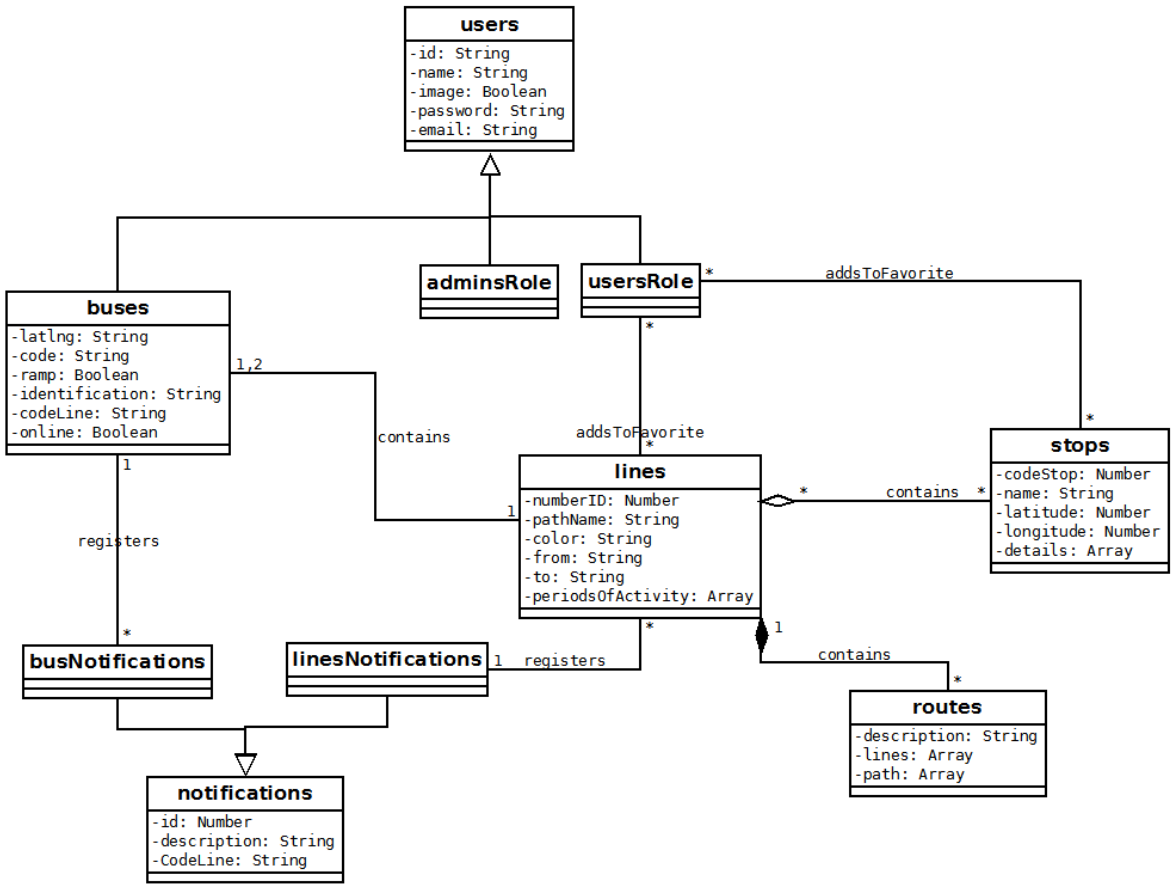


Figura 5. Diagrama de clases UML de la base de datos no relacional.

Este diagrama de clases solo se expone de forma conceptual, ya que el proyecto no utiliza el paradigma orientado a objeto, sino que es completamente funcional y algunas relaciones que se observan en él no se cumplen de la misma manera en la base de datos. Por ejemplo, las herencias no están

soportadas por la base de datos MongoDB. Tanto *notifications* como *users* se pueden resolver con campos con restricciones de tipo *enum* en vez de emplear objetos que heredan propiedades de una clase superior. A continuación, se muestra un ejemplo de cómo Mongoose modela la entidad *User* en la base de datos:

```
const { Schema, model } = require('mongoose');

const UserSchema = Schema({
    name: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true,
        unique: true
    },
    password: {
        type: String,
        required: true
    },
    state: {
        type: Boolean,
        required: true,
        default: true
    },
    rol: {
        type: String,
        required: true,
        enum: ['ADMIN_ROLE', 'USER_ROLE', 'BUS_ROLE'],
        default: 'USER_ROLE'
    },
    favoriteLines: [{
        type: Schema.Types.ObjectId,
        ref: 'Line',
        required: false
    }]
},{timestamps: true});

module.exports = model('User', UserSchema );
```

Código fuente 1. Ejemplo de schema de tipo Users definido por Mongoose.

Como se puede ver en el código anterior, Mongoose se encarga de generar un *schema* que, *a priori*, MongoDB no soporta, y que define un modelo de tipo *User* en el que se describen campos como *name, email, password, state* y *rol*. Este último es de tipo *String* y solo admite valores especificados en el

campo *enum*. La opción *timestamps* genera automáticamente dos campos en el documento: *created_at* y *updated_at*, que almacenan la fecha y hora de la creación o actualización de los datos.

De este modo, un documento queda almacenado así en la base de datos:

```
{
    "_id": {
        "$oid": "615edae286ba6cfe4821cd84"
    },
    "name": "Cristian",
    "email": "ejemplo@ejemplo.com",
    "password": "$2345435oijiop2a$10$R96hsKe31UnwA/……",
    "state": true,
    "rol": "ADMIN_ROLE",
    "createdAt": {
        "$date": "2021-10-07T11:32:50.668Z"
    },
    "updatedAt": {
        "$date": "2021-10-07T11:32:50.668Z"
    },
    "__v": 0
}
```

Código fuente 2. Ejemplo de cómo queda almacenado un documento de tipo *User* en la base de datos.

La estructura que se genera es la de un JSON compuesto por multitud de objetos y campos de tipos definidos como *number, String, Boolean, date,* etc.

De igual forma, MongoDB dispone de dos maneras de almacenar información procedente de interrelaciones binarias o n-arias (muchos a muchos). Una de ellas consiste en incrustar directamente el objeto con el que se relaciona dentro de la colección. La otra utiliza una referencia, que no es más que un código único que apunta al documento. Ejemplo de cómo se relaciona *Lines* con *Stops* a través de una referencia:

```
const mongoose = require('mongoose');

const lineSchema = new mongoose.Schema({
    //_id: mongoose.Schema.Types.ObjectId,

    codeLine: {
        type: String,
        required: true,
        unique: true,
        index: true
    },
```

```
    from:{
        type:String,
        required:true,
        unique:false,
        index:false,
    },
    to:{
        type:String,
        required:true,
        unique:false,
        index:false,
    },

    colorStart:{
        type: String,
        required: false,
        index: false,
    },
    colorEnd:{
        type: String,
        required: false,
        index: false,
    },

    periodOfActivity:[{
        initDate: {
            type: String
        },
        endDate: {
            type: String
        },
        daysOfWeek:{
            type: [String],
            enum: ['monday', 'tuesday', 'wednesday', 'thursday', 'friday',
'saturday', 'sunday']
        },
        initHour: {
            type: String,
            default: '7:30'
        },
        endHour: {
            type: String,
            default: '23:30'
        }
    }],
    firstStop:{
```

```
        type: mongoose.Schema.Types.ObjectId,
        ref: 'stop',
        required: false
    },
    lastStop:{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'stop',
        required: false
    },
    stops: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'stop',
        required: false
    }],
    buses:[{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Bus',
        required: false
    }],
    route: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Route',
        required: false,
        default: null
    }

});

exports.Line = mongoose.model('Line', lineSchema);
```

Código fuente 3. Ejemplo de cómo se construye una referencia en MongoDB.

La propiedad *stops* hace referencia a un *array* de objetos de tipo *Stop*.

4.1.4. Colecciones que almacena la base de datos.

En resumidas cuentas, la información se almacena en seis colecciones de datos[26] definidas en Mongoose con los siguientes nombres:

- **Users**: colección de usuarios que se registran en la aplicación. *Users*, a su vez, almacena los *array* de *lines* y *stops* mediante referencias.

[26] En MongoDB, una colección es el equivalente a una tabla de una base de datos relacional. En las tablas, lo que se almacena se denomina registro, mientras que en una colección, lo que se almacena son documentos, que se componen de claves y valores, y no tienen por qué seguir un mismo patrón estructural todos los documentos de una misma colección.

- **Buses**: colección de autobuses con GPS que existe en la flota y que registran notificaciones a modo de incidencias. Los autobuses acceden a la aplicación vía *websocket* y HTTP mediante la API Los autobuses guardan una referencia con la colección línea (una sola línea) en un momento dado, y registran notificaciones (muchas).
- **Lines**: colección de líneas de autobuses que existen actualmente en la localidad. También emiten incidencias que se registran en notificaciones. Además, contiene una lista de rutas y paradas y se definen con un color. También registra el periodo de actividad mediante un objeto que almacena los intervalos de tiempo.
- **Notifications**: colección de notificaciones. Pueden proceder de una línea o de un autobús. Las notificaciones de línea se lanzan desde el panel de administración.
- **Stops**: colección de paradas. Las paradas pueden tener correspondencia con más de una línea. Almacena su posición geográfica mediante coordenadas, así como las conexiones con otras paradas y líneas.
- **Routes**: colección que almacena *arrays* de las coordenadas que trazan las rutas de cada una de las líneas de autobús. Uno de sus campos es un *array* a todas las referencias de *lines* que posee la ruta especificada por el documento.

4.2. Arquitectura de la información y diagramas de navegación

La aplicación cuenta con una estructura simple de pocos niveles de profundidad. A continuación se describe el árbol de navegación que aparece en la figura número 6 del documento.

***Landing*.** La aplicación comienza en una página de aterrizaje que facilita el acceso al registro o *login* de usuario.

Login* | *Register. No es una página aparte del *landing*, sino que es un subapartado. Este apartado cuenta con un formulario para darse de alta en la aplicación o iniciar sesión.

Admin. No es una página concreta, sino parte de la URI de administración de recursos concretos como /admin/users o /admin/lines. Los recursos gestionados por admin son:

- ***Users***. En la página *Users* se gestionan los usuarios con controles CRUD.
- ***Buses***: En la página *buses* se gestionan las altas y bajas de autobuses con controles CRUD.
- ***Stops***. En la página *Stops* se gestionan las paradas con controles CRUD.
- ***Routes***. En la página *Routes* se gestionan las rutas de cada línea con controles CRUD.
- ***Lines***. En la página *Lines* se gestionan las líneas con controles CRUD.

- ***Notifications***: En la página *Notifications* se gestionan las notificaciones que se puede mandar a cada línea o autobús. También integra el resto de las funciones CRUD (menos actualizar, ya que he considerado que no tiene sentido actualizar una notificación).

User Dashboard. La página principal consta de accesos directos al formulario de búsqueda de rutas y a las líneas disponibles. El *dashboard* cuenta con las siguientes vistas:

- ***SearchResults***: En la página *SearchResults* se visualizan y consultan las rutas mediante un formulario de búsqueda (el mismo que aparece en la página principal del *dashboard*).
- ***Lines***: En la página *Lines* se visualizan todas las líneas disponibles en el servicio de autobuses. También aparece un mapa interactivo con las rutas a color de cada línea.
- ***Favorites*:** En la página *Favorites* se visualiza la lista de líneas favoritas de un usuario, así como las notificaciones.
- ***View Line***: En la página *View Line* se carga la vista correspondiente a una línea concreta. Además del mapa interactivo con la línea concreta que se consulta, cuenta con un listado interactivo desde el que se puede consultar la ruta hasta la parada seleccionada (la ruta se calcula según la posición del usuario que la solicita). También dispone de un listado de instrucciones de navegabilidad para llegar al lugar solicitado. En esta vista también se informan de todas las notificaciones de línea y bus relacionado con la línea y se visualiza el icono gráfico del autobús en tiempo real a lo largo de la ruta trazada en el mapa.
- ***Stops***: En la página *Stops* se carga la vista correspondiente al listado de todas las paradas. En esta se da la posibilidad de filtrar paradas mediante campos de texto, consultar el detalle de cada parada y cómo llegar.
- ***Notifications***: En la página *Notifications* se carga la vista correspondiente a todas las incidencias registradas por cualquier autobús o línea.
- ***User***: En la página *User*, se carga la vista del perfil de usuario. Desde esta vista se puede modificar la contraseña de usuario.

Además de las páginas mencionadas para el usuario corriente, el usuario con rol de tipo autobús también dispone de un apartado denominado **Demo Buses** a efectos de demostración y que no se ha reflejado en la figura 6 ya que se da acceso a funcionalidades que se llevarían a cabo desde un hipotético cliente HTTP/WebSocket del propio autobús y que no está contemplado en este proyecto. Desde este apartado, el autobús puede mandar notificaciones y activar el *script* de simulación de los autobuses en tiempo real.

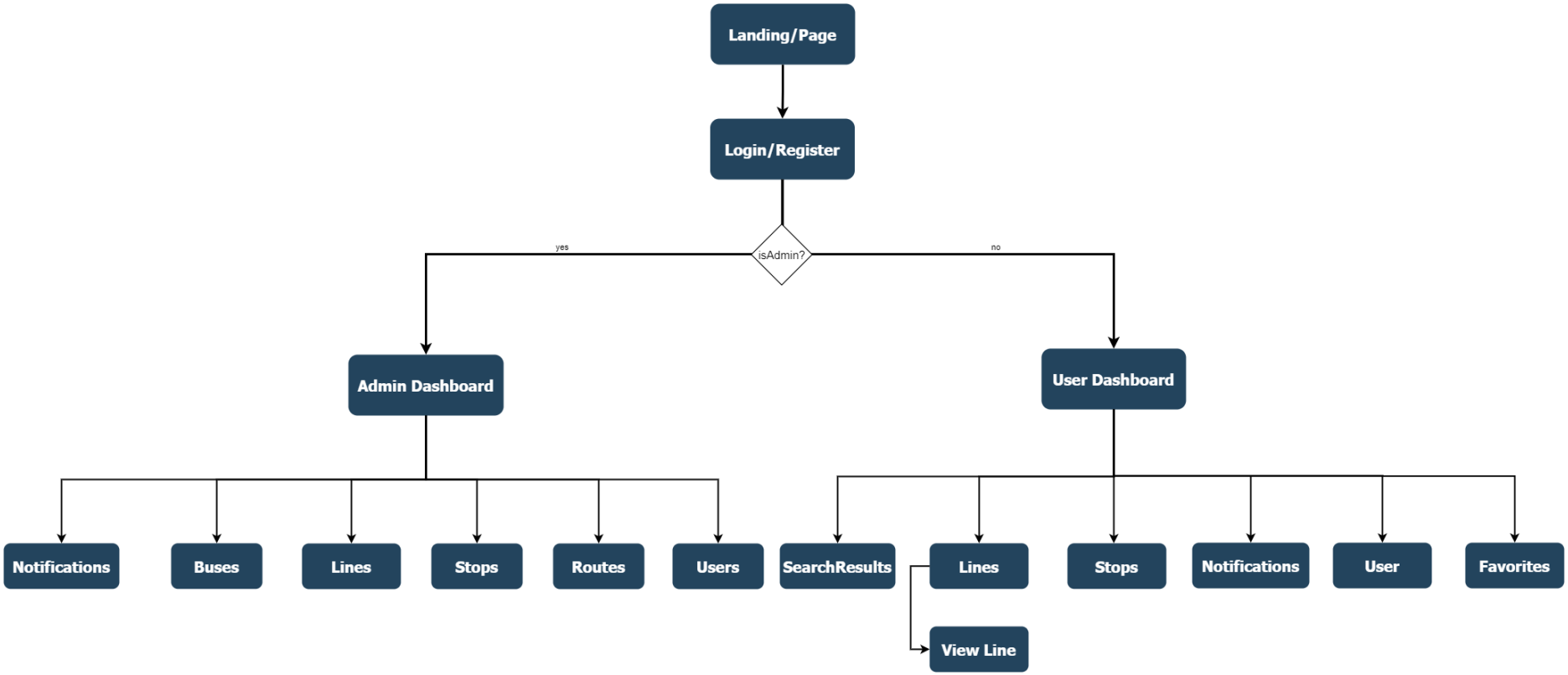


Figura 6. Diagrama de navegación de la aplicación.

4.2.1. Casos de uso más críticos

Identificador de caso de uso: Consultar ubicación de autobús

Actor principal: Usuario de autobús.

Nivel: General

Ámbito: Sistema

Precondición:

- El usuario debe estar *logueado* en la aplicación.

Garantías en caso de éxito: el sistema le mostrará al usuario la ubicación en tiempo real del autobús de una línea concreta.

Escenario principal de éxito:

1. El usuario accede al enlace de la línea concreta (página *View Line*) de los autobuses que desea monitorizar en tiempo real.
2. El sistema carga el mapa interactivo con la ruta correspondiente a la línea y la ubicación de los autobuses en tiempo real.
3. El caso de uso termina.

Escenario alternativo:

2a. El sistema no encuentra autobuses porque no están en horario de servicio.

2a1. El sistema muestra en pantalla el mapa interactivo con la ruta correspondiente a la línea y, mediante un mensaje de texto en pantalla, notifica al usuario de que no hay autobuses emitiendo su posición.

2a2. El caso de uso termina.

2b. El sistema no encuentra autobuses porque el GPS de los autobuses no funciona adecuadamente.

2b1. El sistema muestra en pantalla el mapa interactivo con la ruta correspondiente a la línea y, mediante un mensaje de texto en pantalla, notifica al usuario de que no hay autobuses emitiendo su posición.

2b2. El caso de uso termina.

Identificador de caso de uso: Buscar un destino

Actor principal: Usuario de autobús.

Nivel: General

Ámbito: Sistema

Precondición:

- El usuario debe estar *logueado* en la aplicación.

Garantías en caso de éxito: el sistema le mostrará al usuario la ruta para alcanzar el destino solicitado.

Escenario principal de éxito:

1. El usuario, mediante formulario de búsqueda situado en el *Dashboard* o *ResultSearch* introduce el origen y el destino seleccionando algunas de las paradas que le ofrece el sistema.
2. El sistema encuentra el itinerario más acertado para alcanzar la parada de destino.
3. El sistema carga en *SearchResults* los resultados obtenidos con la ruta establecida para la parada de origen y destino.
4. El caso de uso finaliza.

Escenario alternativo:

1a. El usuario ignora las paradas que le ofrece el sistema e introduce manualmente el origen y destino que desea alcanzar que no coinciden con ningún nombre de parada.

1a1. El sistema obtiene las coordenadas de las paradas de origen y destino más cercanas.

1a2. El caso continúa en el paso 3.

3a. El sistema no encuentra resultados.

3a1. El sistema muestra por pantalla que no existen resultados posibles para los datos de origen y destino utilizados.

3a2. El caso de uso finaliza.

3b. El sistema detecta que el origen o el destino de la búsqueda se encuentra fuera de la localidad de Sanlúcar de Barrameda.

3b1. El sistema le indica al usuario, mediante mensaje de texto, que el origen o el destino se encuentra en una localización diferente a la de Sanlúcar de Barrameda.

3b2. El caso de uso termina.

4.2.2. Peticiones que gestiona la API

A continuación se detalla una lista de *endpoints* que gestiona la API del servidor.

Nombre	**Autenticación de usuario**
URI	/api/auth
Métodos	POST: autentica o no al usuario
Respuesta con status 200	ok: *true* uid: <uid de usuario> name: <*name* de usuario> rol: <rol de usuario> token: <token generado por JWT en el servidor>
Otros status	400 Bad Request. Usuario no coincide o *password* incorrecta. | Validación de campos no superada. 500 Error en el servidor.

Nombre	**Creación de usuario**

URI	/api/auth/new
Métodos	POST: registra un nuevo usuario en la base de datos
Respuesta con *status* 201 (*created*)	ok: *true* uid: <uid de usuario> name: <*name* de usuario> rol: <rol de usuario> token: <token generado por JWT en el servidor>
Otros status	400 Bad Request. Usuario ya existe | Validación de campos no superada. 500 Error en el servidor.

Nombre	**Renovación de token**
URI	/api/auth/renew
Métodos	GET: devuelve un token de usuario
Respuesta con status 201 (*created*)	ok: *true* uid: <uid de usuario> name: <*name* de usuario> rol: <rol de usuario> token: <token generado por JWT en el servidor>
Otros status	401 Unauthorized. Token no válido | Usuario no existe | Usuario borrado | No hay token. 500 Error en el servidor.

Nombre	**Obtener favoritos**
URI	/api/favorites
Métodos	GET: regresa un listado de líneas favoritas.
Respuesta con status 200 (ok)	ok: *true* data: <lista_de_favoritos>
Otros status	401 Unauthorized. Token no válido | Usuario no existe | Usuario borrado | No hay token. 404 No existen favoritos 500 Error en el servidor.

Nombre	**Quitar favoritos**
URI	/api/favorites/:codeLine
Métodos	DELETE: Borra una línea de favoritos favoritos.
Respuesta con status 200 (ok)	ok: *true* data: <lista_de_favoritos>

Otros status	401 Unauthorized. Token no válido | Usuario no existe | Usuario borrado | No hay token. 400 No existen favoritos 409 Error en el servidor.

Nombre	**Agregar a favoritos**
URI	/api/favorites/
Métodos	POST: Agrega una línea a favoritos.
Respuesta con 201 (created)	ok: *true* data: <lista_de_favoritos>
Otros status	401 Unauthorized. Token no válido | Usuario no existe | Usuario borrado | No hay token. 404 No existen favoritos 409 Error en el servidor.

Nombre	**Parada más cercana**
URI	/api/stopinplace/:place
Métodos	GET: Devuelve las paradas más cercanas a un lugar dado.
Respuesta con status 200 (ok)	El servidor devuelve: ok: *true* data: <lista_de_paradas>
Otros status	401 Unauthorized. Token no válido | Usuario no existe | Usuario borrado | No hay token. 400 Los lugares propuestos no están en Sanlúcar de Barrameda' 404 La dirección no existe | Escribe bien la dirección 500 Error en el servidor.

Nombre	**Comprobar si usuario está en Sanlúcar de Barrameda**
URI	/api/coordsinsanlucar/:long/:lat
Métodos	GET: Devuelve *true* o *false*.
Respuesta con status 200 (ok)	El servidor devuelve: ok: *true*
Otros status	401 Unauthorized. Token no válido | Usuario no existe | Usuario borrado | No hay token. 404 La dirección no existe 400 Su ubicación no corresponde con Sanlúcar de Barrameda' | La dirección no existe 500 Error en el servidor.

Nombre	**Líneas**
URI	/api/lines/
Método	POST: Crea una línea
Respuesta con status 201 (created)	ok: *true* data: La línea con el código <codeLine>
Otros status	401 Unauthorized. Token no válido | Usuario no existe | Usuario borrado | No hay token. | Ya existe la línea | Usuario no es admin. 409 Error en el servidor | Hable con el administrador

Nombre	**Obtener todas las líneas**
URI	/api/lines/
Métodos	GET: Devuelve todas las líneas
Respuesta con status 200 (ok)	ok: *true* data: <lista_de_líneas>
Otros status	401 Unauthorized. Token no válido | Usuario no existe | Usuario borrado | No hay token. | Ya existe la línea 404 No existen líneas 500 Error en el servidor | Hable con el administrador

Nombre	**Actualizar una línea**
URI	/api/lines/:id
Métodos	PUT: Actualiza una línea
Respuesta con status 200 (ok)	ok: *true* data: La línea con el código <codeLine>
Otros status	401 Unauthorized. Token no válido | Usuario no existe | Usuario borrado | No hay token. | Ya existe la línea | Usuario no es admin. 400 La línea/líneas no existe/n 409 Error en el servidor | Hable con el administrador

Nombre	**Borrar una línea**
URI	/api/lines/:id
Métodos	DELETE: Destruye una línea

Respuesta con status 200 (ok)	ok: *true*
Otros status	401 Unauthorized. Token no válido | Usuario no existe | Usuario borrado | No hay token. | Ya existe la línea | Usuario no es admin. 400 La línea/líneas no existe/n 409 Error en el servidor | Hable con el administrador

Nombre	**Obtener detalle de línea**
URI	/api/lines/:codeLine
Métodos	GET: obtiene una línea en concreto especificada por <codeLine>
Respuesta con status 200 (ok)	ok: *true* *data:* la línea especificada con <codeLine>
Otros status	401 Unauthorized. Token no válido | Usuario no existe | Usuario borrado | No hay token. | Ya existe la línea 404 La línea/líneas no existe/n 500 Error en el servidor | Hable con el administrador

Nombre	**Obtener todas las Notificaciones**
URI	/api/notifications/
Métodos	GET: Obtiene todas las notificaciones
Respuesta con status 200 (ok)	ok: *true* data: <lista_de_notificaciones>
Otros status	401 Unauthorized: Token no válido | Usuario no existe | Usuario borrado | No hay token. 404 No existen notificaciones 500 Hable con el administrador

Nombre	**Obtener Notificaciones según código de línea**
URI	/api/notifications/:code
Métodos	GET: Obtiene las notificaciones especificadas por el parámetro *code*
Respuesta con status 200 (ok)	ok: *true* data: <lista_de_notificaciones>
Otros status	401 Unauthorized: Token no válido | Usuario no existe | Usuario borrado | No hay token. 404 No existen notificaciones para esa línea 500 Error en el servidor | Hable con el administrador

Nombre	**Crear Notificación**
URI	/api/notifications/
Métodos	POST: Crea una notificación
Respuesta con status 201 (created)	ok: *true* data: <lista_de_notificaciones>
Otros status	401 Unauthorized: Token no válido | Usuario no existe | Usuario borrado | No hay token. | Usuario no es admin. | Usuario no es bus. 409 Error de inserción

Nombre	**Borrar Notificación**
URI	/api/notifications/:id
Métodos	DELETE: borra una notificación
Respuesta con status 201 (created)	ok: *true* data: <lista_de_notificaciones>
Otros status	401 Unauthorized: Token no válido | Usuario no existe | Usuario borrado | No hay token. | Usuario no es admin. 400 No existe esa línea | 409 Error de borrado

Nombre	**Perfil**
URI	/api/profile/
Métodos	POST: Cambia la password del usuario
Respuesta con status 200 (ok)	ok: *true* data: ‘’
Otros status	401 Unauthorized: Token no válido | Usuario no existe | Usuario borrado | No hay token. 400 El usuario no existe. 500 Error en el servidor | Hable con el administrador

Nombre	**Obtener rutas**
URI	/api/routes
Métodos	GET: obtiene todas las rutas de las líneas.
Respuesta con status 200 (ok)	ok: *true* data: <lista_de_rutas>

Otros status	401 Unauthorized:. Token no válido | Usuario no existe | Usuario borrado | No hay token. 404: Error, no existen rutas. 500 Error en el servidor | Hable con el administrador

Nombre	**Crear ruta**
URI	/api/routes
Métodos	POST: crea una ruta.
Respuestas con status 201(created)	ok: *true* data: <lista_de_rutas>
Otros status	401 Unauthorized:. Token no válido | Usuario no existe | Usuario borrado | No hay token. | Usuario no es admin. 409: Error durante la creación | Hable con el administrador

Nombre	**Borrar ruta**
URI	/api/routes/:id
Métodos	DELETE: borra una ruta.
Respuesta con status 200 (ok)	ok: *true*
Otros status	401 Unauthorized:. Token no válido | Usuario no existe | Usuario borrado | No hay token. | Usuario no es admin. 400: Error, no existe esa ruta. 500 Error en el servidor | Hable con el administrador

Nombre	**Actualizar ruta**
URI	/api/routes/:id
Métodos	PUT: actualiza una ruta.
Respuesta: Status 200 (ok)	ok: *true* data: <lista_de_rutas>
Otros status	401 Unauthorized: Token no válido | Usuario no existe | Usuario borrado | No hay token. | Usuario no es admin. 400: Error, no existe esa ruta. 409: Error durante la actualización | Hable con el administrador

Nombre	**Búsqueda de itinerario en autobús**
URI	/api/search/:place1/:place2
Métodos	GET: Obtiene un itinerario
Respuesta con status 200 (ok)	ok: *true* data: <paradas_ordenadas>
Otros status	401 Unauthorized: Token no válido | Usuario no existe | Usuario borrado | No hay token. 404: Error, no se puede encontrar el itinerario. 500 Error en el servidor | Hable con el administrador

Nombre	**Obtener paradas**
URI	/api/stops/
Métodos	GET: Lista paradas
Respuesta con status 200 (ok)	ok: *true* data: <lista_de_paradas>
Otros status	401 Unauthorized: Token no válido | Usuario no existe | Usuario borrado | No hay token. 500 Error en el servidor | Hable con el administrador

Nombre	**Crea una parada**
URI	/api/stops/
Métodos	POST: Crea una parada
Respuesta 201 (created)	ok: *true* data: <lista_de_paradas>
Otros status	401 Unauthorized: Token no válido | Usuario no existe | Usuario borrado | No hay token. | Usuario no es admin 409 Error al crear parada | Hable con el administrador

Nombre	**Actualizar parada**
URI	/api/stops/:id
Métodos	PUT: actualiza una parada.
Respuesta: Status 200 (ok)	ok: *true* data: <parada_actualizada>
Otros status	401 Unauthorized:. Token no válido | Usuario no existe | Usuario borrado | No hay token | Usuario no es admin. 400: Error, no existe esa parada. 409: Error durante la actualización | Hable con el administrador

Nombre	**Borrar parada**
URI	/api/stops/:id
Métodos	DELETE: borra una parada.
Respuesta: **Status 200 (ok)**	ok: *true*
Otros status	401 Unauthorized:. Token no válido | Usuario no existe | Usuario borrado | No hay token. | Usuario no es admin. 400: Error, no existe esa parada. 409: Error durante borrado | Hable con el administrador

Nombre	**Obtener usuarios**
URI	/api/users/
Métodos	GET: Lista usuarios
Respuesta con status 200 (ok)	ok: *true* data: <lista_de_usuarios>
Otros status	401 Unauthorized: Token no válido | Usuario no existe | Usuario borrado | No hay token. | Usuario no es admin 500 Error en el servidor | Hable con el administrador

Nombre	**Actualizar usuario**
URI	/api/users/:id
Métodos	PUT: actualiza un usuario.
Respuesta: **Status 200 (ok)**	ok: *true* data: <usuario_actualizado>
Otros status	401 Unauthorized: Token no válido | Usuario no existe | Usuario borrado | No hay token. | Usuario no es admin 400: Error, no existe ese usuario. 409: Error durante la actualización | Hable con el administrador

Nombre	**Borrar usuario**
URI	/api/user/:id
Métodos	DELETE: borra un usuario.

Respuesta: **Status 200 (ok)**	ok: *true*
Otros status	401 Unauthorized:. Token no válido | Usuario no existe | Usuario borrado | No hay token. | Usuario no es admin 400: Error, no existe ese usuario. 409: Error durante borrado | Hable con el administrador

Nombre	**Instrucciones a pie hasta destino.**
URI	/api/directions/:long1/:lat1/:long2/:lat2
Métodos	GET: Devuelve las instrucciones para llegar a un lugar saliendo desde las coordenadas geográficas indicadas por origen (long1, lat1) y destino (long2, lat2)
Respuesta con status 200 (ok)	ok: *true* data: <lista_de_instrucciones_a_pie>
Otros status	401 Unauthorized:. Token no válido | Usuario no existe | Usuario borrado | No hay token. 404: Error de ruta no encontrada en la API de MapBox-GL. 500 Error en el servidor | Hable con el administrador

Nombre	**Obtener autobuses**
URI	/api/buses/
Métodos	GET: Lista autobuses
Respuesta con status 200 (ok)	ok: *true* data: <lista_de_buses>
Otros status	401 Unauthorized:. Token no válido | Usuario no existe | Usuario borrado | No hay token. 500 Error en el servidor | Hable con el administrador

Nombre	**Crea un autobús**
URI	/api/buses/
Métodos	POST: Crea un autobús
Respuesta 201 (created)	ok: *true* data: <lista_de_autobuses>
Otros status	401 Unauthorized: Token no válido | Usuario no existe | Usuario borrado | No hay token. | Usuario no es admin 409 Error al crear autobús | Hable con el administrador

Nombre	**Actualizar autobús**
URI	/api/buses/:id
Métodos	PUT: actualiza un autobús.
Respuesta: **Status 200 (ok)**	ok: *true* data: <autobús_actualizado>
Otros status	401 Unauthorized: Token no válido | Usuario no existe | Usuario borrado | No hay token | Usuario no es admin. 400: Error, no existe ese autobús. 409: Error durante la actualización | Hable con el administrador

Nombre	**Borrar autobús**
URI	/api/buses/:id
Métodos	DELETE: borra un autobús.
Respuesta: **Status 200 (ok)**	ok: *true*
Otros status	401 Unauthorized: Token no válido | Usuario no existe | Usuario borrado | No hay token. | Usuario no es admin. 400: Error, no existe ese autobús. 409: Error durante borrado | Hable con el administrador

4.3. Diseño gráfico e interfaces

4.3.1. Estilos

Nombre de la aplicación y símbolo gráfico.

Aunque la zona geográfica para la que se ha diseñado este proyecto no contiene ninguna bahía, ya que en principio solo se aplica a la Costa Noroeste de la provincia de Cádiz, el sustantivo Bahía hace alusión a la zona metropolitana de la Bahía de Cádiz, lugar de nacimiento del autor del proyecto. Tanto la palabra Bus como el resto de elementos gráficos del símbolo gráfico hacen alusión a una herramienta capaz de generar resultados de itinerarios de búsqueda en autobús. El arco que rodea al logotipo recuerda tanto a un camino asfaltado como a una bahía.

Figura 7. Símbolo gráfico Bahía-Bus.

Paleta de colores y tipografía

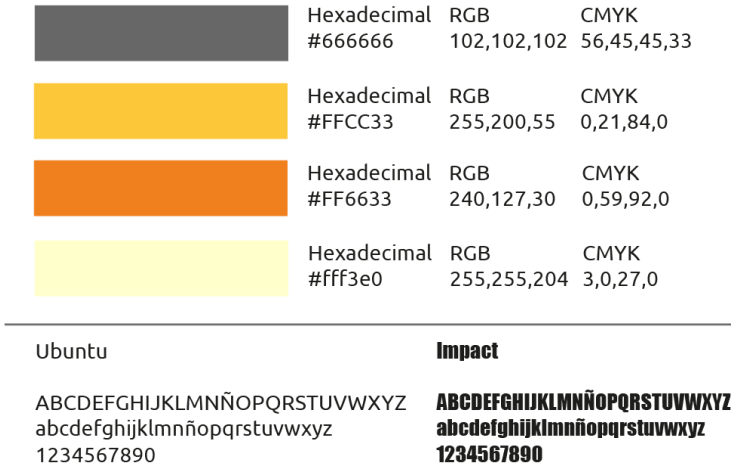


Figura 8. Paleta de colores y tipografía.

La paleta seleccionada está compuesta en su mayoría por colores energéticos anaranjados que se combinan con gris oscuro. Estos colores, en conjunto, forman lo que Itten[27] denomina como contraste de cualidad. Esta paleta se utiliza tanto en el logo como en la aplicación y la divide en dos zonas: una con tonos anaranjados y grises, el *header* y el *nav* horizontal superior; y otra con blanco de fondo, que ocupa el resto del espacio y que contrasta con el gris oscuro del texto.

Ninguna de las tipografías seleccionadas para el logotipo, Ubuntu o Impact, tienen *serifa*[28], ya que no se busca un alto contraste estético u ornamental entre ambas, sino una combinación correcta definida por el grosor de sus líneas y la estética moderna de Ubuntu, con terminales más puntiagudos que Impact.

4.3.2. Usabilidad /UX

La aplicación se desarrolla teniendo en cuenta las bases de la heurística promulgada por Jakob Nielsen, en los 10 principios de usabilidad publicados en el libro titulado *Ingeniería de la usabilidad* (Nielsen, 1993):

- **Visibilidad del estado del sistema**: La aplicación muestra continuamente barras de progreso o códigos de estado con los que informar al usuario del estado del sistema. Un estado 404 informa al usuario de la no encontrabilidad de una página. Un estado 200, aunque es transparente, muestra al usuario un resultado satisfactorio. Igualmente, las migas de pan de la aplicación le indican al usuario dónde está en todo momento.
- **Relación entre el sistema y el mundo real**: La aplicación emplea metáforas visuales como iconos gráficos con las que representar el mundo real. Estas metáforas están acompañadas de textos explicativos que hacen más amena la interacción con el sistema.
- **Libertad y control del usuario**: El usuario dispone de toda la información necesaria para manejar la aplicación, ya que su interfaz es sencilla e intuitiva y es poco probable que ocurran efectos imprevistos. Además, el usuario tiene la posibilidad de hacer y deshacer cualquiera de las opciones llevadas a cabo en la aplicación.
- **Consistencia y estándares**: La aplicación utiliza los mismos recursos gráficos y estructurales durante toda la experiencia de navegación con el objeto de no provocar una experiencia poco predecible y que el aprendizaje sea mínimo. Además, se ajusta a los estándares HTML5 de la W3C[29].

[27] Según Wikipedia: "Johannes Itten (Wachseldorn, Suiza, 11 de noviembre de 1888 – 25 de marzo de 1967) fue un pintor, diseñador, profesor y escritor suizo. Formó parte de la escuela Bauhaus y fue profesor de la escuela HfG en Ulm (Alemania)." (https://es.wikipedia.org/wiki/Johannes_Itten , recuperado el 21 de octubre de 2021).

[28] Una fuente tipográfica no *serifada* es aquella que no presenta ningún tipo de adorno estético, o gracia. Son también denominadas fuentes de palo seco.

[29] World Wide Web Consortium. El Consorcio World Wide Web (W3C) es una comunidad internacional que desarrolla estándares abiertos para asegurar el crecimiento a largo plazo de la Web. (https://www.w3.org/ - recuperado el 21 de octubre de 2021).

- **Prevención de errores:** La aplicación incorpora mecanismos de control y prevención de errores (como, por ejemplo, evitar introducir caracteres de texto donde deben ir números), así como vías alternativas para que los efectos producidos por la aparición de aquellos sean mínimos.
- **Reconocer mejor que recordar:** La aplicación recuerda búsquedas ya efectuadas con anterioridad, por lo que no tiene que estar recordándolas continuamente. De igual forma, tiene la posibilidad de incluir en una lista de favoritos las paradas y las líneas que más le convengan con el fin de no tener que recordar cada una de ellas.
- **Flexibilidad y eficiencia de uso:** La aplicación dispone de atajos, entendidos como accesos directos, a la información más relevante, como es el caso de las paradas y líneas favoritas, lo que mejora la eficiencia en el manejo de la misma a usuarios más expertos o experimentados.
- **Estética y diseño minimalista:** La aplicación pone a disposición del usuario la información mediante elementos estéticos poco recargados (minimalistas) que focalizan toda la atención en los aspectos más relevantes de aquella.
- **Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de errores**: La aplicación está diseñada para que la mayoría de errores no tengan que ser gestionados de manera directa por los usuarios. Sin embargo, habrá veces que se encontrarán una URL inexistente, o un problema derivado del servidor que inevitablemente les impedirá utilizar la aplicación del modo esperado. Estos errores son notificados de manera clara al usuario, de modo que sepan qué hacer en el momento en el que ocurran, como esperar a que el sistema se recupere del error, acceder a un enlace alternativo o introducir la información correcta de un formulario.
- **Ayuda y documentación:** La aplicación ofrece ayudas en forma de *tips* para completar ciertos campos de texto, pero no incluye documentación adicional en forma de manual, ya que intenta ser lo más intuitiva posible.

4.4. Lenguajes de programación y APIs utilizados

El sistema, en su conjunto, se basa en un paradigma de programación funcional. El paradigma funcional es aquel que se destina a evitar cambios de estado y de datos mutables, y procura que todas las funciones devuelvan exactamente el mismo valor dado un mismo argumento. Esto se hace para conseguir que el resultado de la ejecución del código sea lo más predecible posible, y que las funciones sean reutilizables. En este sentido, JavaScript es un lenguaje de programación muy versátil, ya que se puede emplear utilizando muchos de los paradigmas de programación existentes en diferentes contextos, y por ello se ha convertido en uno de los más demandados en el mercado de la web.

El *stack* utilizado para el desarrollo de este proyecto es el de MERN. La ventaja es que todas las tecnologías implicadas en él utilizan JavaScript como lenguaje. Históricamente, este lenguaje fue concebido para ejecutarse directamente en el navegador y dotar a la web de dinamismo. Netscape

Communications, empresa creadora de uno de los primeros navegadores web de la historia, Netscape Navigator, lanzó su primera versión en diciembre de 1995. A pesar del parecido de su nombre con Java, JavaScript es un lenguaje independiente. En 1997 fue propuesto como estándar de la ECMA (European Computer Manufacturers' Association), y fue adoptado con el nombre de ECMAScript.

JavaScript está soportado por la gran mayoría de navegadores y destaca por su versatilidad. Como se describe en JavaScript.info (Kantor), JavaScript de lado del cliente permite:

- Añadir o modificar contenido a la web.
- Reaccionar a eventos de ratón y teclado.
- Enviar solicitudes a través de la red a servidores remotos, descargar archivos y subirlos.
- Gestionar Cookies, enviar mensajes al usuario.
- Memorizar datos en el *local storage*.

Con el tiempo, se ha comprobado la eficacia de JavaScript no solo en el lado del navegador, sino también en el del servidor, y a día de hoy compite con otros lenguajes ya consolidados como PHP, ASP.NET o JSP.

4.4.1. NPM: Node Package Management

Otra de las ventajas de utilizar JavaScript durante todo el plan de desarrollo, ha sido la de gestionar los paquetes de dependencias de terceros mediante NPM (Node Package Management), una plataforma que permite instalar o desinstalar *software* de dependencias a través de la consola de comandos. Para disponer de NPM, primero hay que tener instalado NodeJS[30]. Una manera rápida de instalar los paquetes básicos para crear una aplicación basada en React consiste en invocar el comando: *npx create-react-app <nombre_de_directorio>.* Para instalar paquetes de la manera ordinaria, hay que ejecutar *npm i <nombre_de_paquete>.*

4.4.2. Características de ReactJS

En el lado del cliente, se utiliza la biblioteca de ReactJS, creada por la empresa Facebook Inc y con licencia *open source* (MIT). React es considerado por la comunidad como un *framework* de JavaScript, pero esto no es del todo cierto. Un *framework* ofrece parte de la implementación del diseño que hay que desarrollar, así como una arquitectura y unos patrones ya establecidos que hay que respetar escrupulosamente, y, lo más importante: en un *framework* el desarrollador no posee el control del flujo del *software* con el que trabaja, sino que se basa en un patrón denominado *inversion of control* e informa al programador de aquellos recursos que necesita. React, sin embargo, es una biblioteca que se encarga de ayudar al desarrollador a implementar la capa de la vista única y exclusivamente con

[30] Primero se descarga desde la página de NodeJS para el sistema operativo que haga falta: https://nodejs.org/es/download/ (Recuperado el 22 de octubre de 2021).

toda la libertad que el lenguaje JavaScript ofrece a los programadores. Lo más parecido a un *framework* en este proyecto es Redux, que, como ya se comentó anteriormente, centraliza toda la gestión concerniente a los cambios de estado de la aplicación a través de *reducers*.

A continuación, se ofrece una lista de características fundamentales de React:

1. React se sirve de una extensión de JavaScript que se denomina JSX[31], que no es más que JavaScript + XML, y que recuerda a la sintaxis de HTML con ligeras modificaciones.

```
import './App.css';
import Mensaje from './Mensaje.js';

const App = () => {
  const mensaje = "Hola qué tal a todos"

  return (
    <div className="App">
      <Mensaje color='red' message='mensaje 1'/>
      <Mensaje color='blue' message='mensaje 2'/>
      <Mensaje color='yellow' message='mensaje 3'/>
      {mensaje + ' evaluación JSX'}
    </div>
  );
}

export default App;
```

Código fuente 4. Ejemplo de código en JSX.

2. React además se sirve de ReactDOM, es decir, un DOM virtual[32], una representación en memoria del DOM real reescrita en JavaScript. El DOM es muy lento a la hora de trabajar con él para actualizarlo directamente. Para mejorar su rendimiento, el DOM Virtual se carga en memoria, y React lo compara con el DOM real para llevar a cabo las modificaciones oportunas sin tener que renderizar todo el DOM de nuevo.

3. React ayuda al programador a crear sus propios componentes reutilizables, es decir, trozos de código independientes con personalidad propia en forma de función que devuelven HTML, lo que es una ventaja de cara al futuro en cuanto a mantenibilidad y escalabilidad de la aplicación. Estos componentes se pueden anidar y se les puede emitir mensajes a través de *props* desde el padre hacia el hijo.

[31] Se puede encontrar más información en: https://es.reactjs.org/docs/introducing-jsx.html (Recuperado el 22 de octubre de 2021).

[32] Se puede encontrar más información en: https://es.reactjs.org/docs/faq-internals.html (Recuperado el 22 de octubre de 2021)

```
const App = () => {
  const mensaje = "Hola qué tal a todos"

  return (
    <div className="App">
      <Mensaje color='red' message='mensaje 1'/>
      <Mensaje color='blue' message='mensaje 2'/>
      <Mensaje color='yellow' message='mensaje 3'/>
      {mensaje + ' evaluación JSX'}
    </div>
 );
}

const Mensaje = (props) => {
    const {color, message} = props;
    return <h1 style={{ color: color }}>Hola, el mensaje es: { message } </h1>
}
```

Código fuente 5. Ejemplo de cómo pasa atributos un padre a un componente hijo mediante *props*.

4. A partir de la versión 16.8 de React, se introdujeron los *hooks*, una nueva manera de trabajar los estados sin utilizar clases ni tener en cuenta el ciclo de vida de los componentes o la complejidad asociada al manejo de la lógica de estado. El *hook* se define de la siguiente manera: «[…] es una función especial que permite "conectarse" a características de React. Por ejemplo, useState es un Hook que te permite añadir el estado de React a un componente de función.» (React). Los dos más utilizados son: *useState* y *useEffect*. Además, React permite crear hooks personalizados. En este proyecto se utilizan varios como useForm, useMapBox, useSocket, etc.

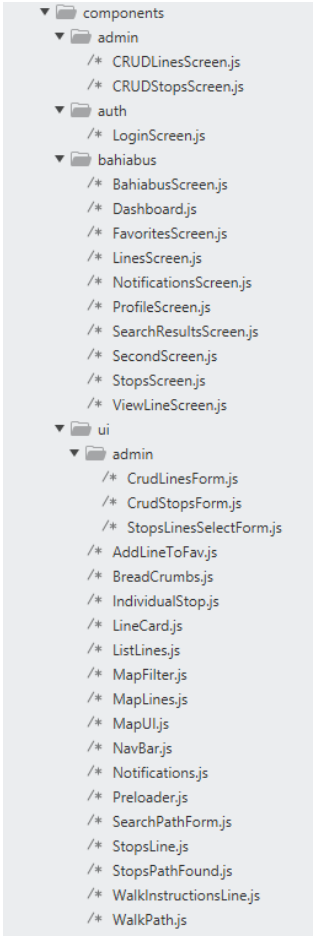
Dependencias que utiliza React en este proyecto:

- axios: biblioteca para realizar conexiones HTTP. Es una fachada con la que se realizan solicitudes a través del objeto XMLHttpRequests.
- mapbox-gl: biblioteca para el manejo interactivo de mapas en la web.
- materialize-css: *framework* para el desarrollo de estilos en CSS.
- redux y react-redux: biblioteca para el control de estados de la aplicación a nivel global.
- react-router-dom: biblioteca que gestiona las rutas URL de la aplicación.
- redux-thunk: biblioteca que le otorga soporte para llamadas asíncronas a acciones gestionadas por redux a través de *middlewares*.

- uuid: biblioteca para la creación de objetos con números de identificación únicos (regulado en la RFC4122[33]).
- Socket.io: permite generar comunicaciones a través de *websocket* desde el cliente.

4.4.3. Componentes programados en React.

El *frontend* se divide en dos tipos de componentes: unos de nivel general, y otros de nivel más particular referidos a la UI propiamente dicha. Los componentes de nivel general son aquellos que aglutinan componentes de nivel más específico, es decir, una página. Cada página es un componente que incluye el sufijo *Screen* en su denominación. Así, existen componentes principales para el apartado de administración, para la autenticación y para la aplicación en sí (directorio bahiabus). El resto de componentes son reutilizables en la mayoría de casos.



Componentes del FrontEnd

Los componentes de UI más repetidos en todas las páginas son: **<NavBar /> y <BreadCrumbs />.**

El componente, o página, *Dashboard* se invoca mediante la etiqueta **<Dashboard />** y contiene componentes de nivel detalle del tipo:

- Formulario de búsqueda: con dos campos de texto, un botón de *submit* y un fondo con una imagen. Se invoca mediante la etiqueta **<SearchPathForm />**

En SearchResults **<SearchResultsScreen />**:

- Se reutiliza el componente de formulario de búsqueda **<SearchPathForm />.**
- Mapa: el componente que genera la API de MapboxGL y que dibuja, además del mapa, iconos de paradas y autobuses, y *polylines* de los itinerarios. Se invoca mediante el *custom hook* useMapbox, que abstrae toda la lógica del mapa, y que es invocado desde dos ubicaciones diferentes: **<MapUI/>**, que muestra los resultados de búsqueda de itinerarios y que incluye un filtro de líneas denominado **<MapFilter />**.

- El componente de instrucciones **<WalkPath />** que muestra al usuario la ruta a pie hasta parada origen, aunque en este caso, también puede mostrar el destino solicitado, también invoca otro componente denominado **<WalkInstructionsLine />**, *contenedor contenido*

[33] La RFC4122 se puede consultar en la siguiente URL: https://datatracker.ietf.org/doc/html/rfc4122 (Recuperado el 30 de octubre de 2021)

vinculado al anterior, y que muestra línea a línea el resultado de la llamada a la API de Mapbox-GL.

- **<StopsPathFound />** muestra el itinerario encontrado tras realizar una búsqueda. La API devuelve el resultado en formato JSON y el componente se dedica a mapearlo en pantalla en forma de lista.

Líneas, componente **<LinesScreen/>**, contiene un listado de líneas de tipo *cards*[34] con cada línea y enlace a parada. Se invoca mediante la etiqueta **<ListLines /> y <LineCard/>** Además, contiene**:**

- El componente de instrucciones **<WalkPath />**, que muestra al usuario la ruta a pie hasta la parada solicitada por el usuario.
- Para mostrar el mapa, esta vista invoca **<MapFilter />,** que muestra los autobuses en tiempo real.

Listado de paradas, **<StopsScreen/>,** es un componente que lista de una manera gráfica las paradas de autobuses que invoca al componente **<StopsLine />** que, a su vez, invoca a **<IndividualStop />** y que renderiza el resultado de todas las paradas en pantalla.

- Utiliza el componente **<MapLine />,** que reutiliza el *custom hook* denominado useMapbox, e imprime en pantalla las rutas de las líneas. Además, recibe eventos del *socket* desde el servidor para actualizar la posición de los autobuses en tiempo real.
- También reutiliza **<WalkPath/>**.

En ViewLine, **<ViewLineScreen />,** se visualiza la vista de una línea en concreto:

- Reutiliza el componente **<MapLine />** para mostrar todas las paradas en el mapa, así como **<WalkPath/>**.
- Notificaciones: listado de avisos emitidos por las líneas (el administrador de la aplicación en este caso) o un autobús a través de su sistema de navegación. Se invoca mediante la etiqueta **<Notifications />**
- **<AddLineToFav/>** componente que muestra un icono de favoritos para añadir una línea al perfil de favoritos del usuario.

En Notifications **<NotificationsScreen />** Se obtienen las notificaciones de línea y autobús. Utiliza el componente **<Notifications/>** que renderiza los resultados procedentes de la API del servidor.

En Favorites **<FavoritesScreen />**:

- Se reutiliza **<ListLines />**

[34] MaterializeCSS la describe de la siguiente manera: “Las tarjetas son un medio para mostrar contenido compuesto por diferentes tipos de objetos. También son adecuados para presentar objetos similares cuyo tamaño o acciones admitidas pueden variar considerablemente, como fotos con leyendas de longitud variable.”

- Se reutiliza **<Notifications />**

En Users **<ProfileScreen />**, se muestra un formulario para cambiar la password del usuario.

Las páginas del panel de administración utilizan un esquema muy parecido y basado en tablas y formularios. Sus componentes principales incluyen en su nombre la siguiente nomenclatura: **<CRUDNombreComponenteScreen />**, que, como indican las siglas, posibilita las tareas de Creación, Lectura, Actualización y Borrado de registros de las tablas. Estos componentes incluyen otros más específicos para la creación de formularios, y que se encuentran en la ruta **src/components/ui/admin**.

4.4.4. Características de la implementación del servidor.

Cálculo de la ruta más corta.

Cuando un usuario realiza una búsqueda de un itinerario concreto, el servidor es capaz de calcular la ruta más corta mediante grafos computacionales. Un grafo computacional no es más que una estructura de datos que interconecta vértices, o nodos, a través de aristas.

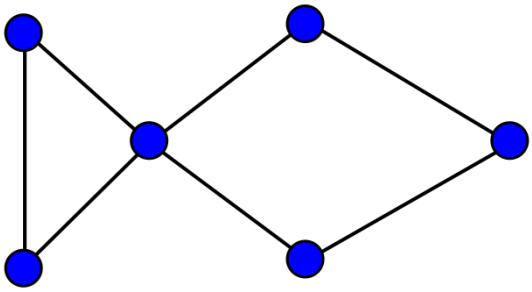


Figura 9. Ejemplo visual de grafo no dirigido.

En Bahía-Bus, cada nodo es una parada. El grafo que se genera en el servidor es de tipo dirigido, eso significa que cada nodo se conecta solo con el nodo, o los nodos, que se indiquen mediante puntas de flecha. Si el grafo no fuese dirigido, los nodos estarían interconectados entre sí de manera bidireccional. Sin embargo, los grafos dirigidos pueden conectarse bidireccionalmente o no. En el siguiente ejemplo, un nodo A se conecta a un nodo B, pero el nodo B no podría alcanzar al nodo A. Sin embargo, A puede conectarse tanto a B como a C.

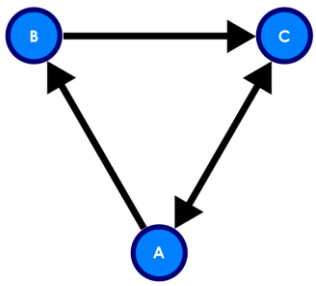


Figura 10. Ejemplo de grafo dirigido.

En la aplicación, cada vértice, o nodo, lo representa una parada. Esta parada, a su vez, conoce sus posibles conexiones (sus vértices adyacentes), además de la línea que las conecta. A través de la implementación en JavaScript de un algoritmo de búsqueda en amplitud, se calcula el recorrido más corto desde una parada a otra sin tener en cuenta los tiempos de espera en caso de transbordo, ya que los horarios y rutas alternativas no están bien documentados en la página del Grupo Avanza y es complicado de implementar para obtener un grafo ponderado en ese sentido.

El algoritmo de búsqueda en amplitud, que en inglés se denomina, por sus siglas, BFS, Breadth First Search, obtiene una matriz de acceso mediante el empleo de la búsqueda de todos los vértices conectados en cada nivel. Se podría imaginar como la búsqueda por niveles en un árbol binario. El algoritmo BFS atraviesa el árbol desde el primer vértice especificado y visita a todos sus vértices adyacentes, capa por capa. Se le llama algoritmo de recorrido en amplitud porque en primer lugar, recorre los vértices a lo ancho, y luego, en profundidad. Una explicación más detallada de cómo funciona el algoritmo, se puede leer en el libro *Learning JavaScript Data Structures and Algorithms by* (Groner, 2014). En concreto, el algoritmo que implementa la aplicación para encontrar la ruta más corta está basado en una respuesta que el usuario Michael Laszlo lanza en *stackoverflow*[35].

La ruta a pie a parada origen y a destino.

La ruta a pie a parada origen y a lugar de destino se generan mediante llamada a la API de Mapbox-GL, por lo que la aplicación solo se dedica a *renderizar* su resultado. Se pueden dar dos situaciones:

- En el caso de que se solicite una búsqueda de un origen que no contenga una parada concreta, la aplicación genera una llamada a la API de Mapbox-GL (Directions) y se obtiene una respuesta de las coordenadas de dicho origen. A continuación, el sistema calcula la parada más cercana mediante la biblioteca Turf.js. Después, el servidor vuelve a mandar otra solicitud a la API de Mapbox-GL para obtener la ruta a pie hasta dicha parada tomando como punto inicial la ubicación del usuario en GPS. El resultado se *renderiza* en pantalla.

- En el caso de que se solicite un origen que no contenga parada, la aplicación envía una petición a la API de Mapbox-GL para obtener las coordenadas de destino. A continuación, calcula la parada más cercana al destino solicitado con la biblioteca Turfs.js. Cuando se obtiene la parada más cercana, se vuelve a solicitar la ruta desde la parada hasta el destino solicitado. El resultado se *renderiza* en pantalla.

[35] El hilo a la respuesta en cuestión se puede visitar en:
https://stackoverflow.com/questions/32527026/shortest-path-in-javascript
(Recuperado el 05 de diciembre de 2021)

Socket de conexión.

En cuanto el usuario inicia la aplicación y se conecta al *login*, se abre un *websocket* que comunica al cliente con el servidor. Los autobuses envían mensajes a través de dicho canal de comunicación. Cuando el servidor recibe uno de estos mensajes, los transforma en ubicaciones que son retransmitidas en modo *broadcast* para que llegue a todos los clientes subscritos al *socket*.

Los autobuses necesitan, además, un *token* de seguridad para conectarse al servidor a través de la API. Estos envían las credenciales con nombre de usuario y *password* a través de una llamada HTTP normal al servidor, que le devuelve un *token* que se almacenará localmente en el cliente, y que, a continuación, se utilizará a través del canal de conexión abierto por el *websocket* para comprobar que el usuario es de tipo autobús y que existe. Una vez validado, el autobús comienza a enviar su ubicación y el servidor ejecuta un controlador para cambiar la propiedad *online* de tipo *booleano* del autobús almacenado en la base de datos a *true*. A su vez, el servidor lo almacenará localmente en un *array* de objetos de tipo marcador, y, por último, lo devolverá a todos los clientes subscritos al *socket*, que los posicionará en un mapa de la biblioteca MapBoxGL con el icono del autobús. Cuando el autobús se desconecta del *socket*, es decir, deja de emitir mensajes a través de él, el servidor cierra la comunicación y establece la propiedad *online* del documento correspondiente a la colección autobuses a *false*.

4.4.5. Características de NodeJS y ExpressJS

NodeJS es una plataforma de desarrollo de aplicaciones del lado del servidor concebida para la creación de aplicaciones de propósito general. Su caso de uso más común es el de realizar la labor de servidor de aplicaciones web, tal y como lo hacen Apache e IIS, y su popularidad va en aumento de acuerdo a los datos arrojados por Node Source[36].

Junto a NodeJS, se utiliza ExpressJS, un *framework* de desarrollo con licencia MIT que provee servicios de enrutamiento y *middlewares* a NodeJS entre otras cosas, y a través del cual se ha generado una API.

Otras dependencias necesarias para el proyecto son:

- Bcryptjs: biblioteca para *hashear* la *password* que se va a almacenar en la base de datos.
- Cors: permite solicitar recursos restringidos en una página web fuera del dominio desde el que se sirvió el primer recurso.
- Dotenv: que carga variables de entorno desde un archivo *.env* a *process.env*.

[36] Estos datos se pueden consultar en la web: https://nodesource.com/node-by-numbers (Recuperado el 22 de octubre de 2021).

- Express-validator: es un conjunto de *middlewares* con funciones de validación y de *sanitizer* (*scripts* que protegen de ataques XSS[37]).
- Jsonwebtoken: implementación del estándar abierto JSON Web Token para la propagación de identidad y privilegios en la aplicación. Sirve, sobre todo, para proteger las sesiones de usuario.
- Moment: para manejo y formateo de fechas.
- Mongoose: de la que ya se ha hablado. Una herramienta para la gestión y modelado de objetos que han sido mapeados desde colecciones y documentos de MongoDB.
- Socket.io: permite generar comunicaciones a través de *websocket* desde el servidor.

4.4.6. APIs de terceros utilizadas

Se utiliza Mapbox como biblioteca para la generación de mapas interactivos. Además, Mapbox se divide en diferentes paquetes de API, cada una con una finalidad y un coste monetario asociado. Además de la inserción de mapas, Mapbox dispone de APIs para la navegación desde un punto a otro, trazar rutas, buscar sitios a través de coordenadas o cadenas de texto, etc. En resumen, Mapbox ofrece los siguientes servicios:

- Maps: integración de aplicaciones con mapas interactivos.
- Navigation: navegación a través de puntos o rutas de manera inteligente con guías paso a paso, información sobre tráfico, modalidad de viaje a pie, bicicleta o coche; avisos por incidentes.
- Atlas: para la creación de mapas interactivos desde cero.
- Search: búsqueda geográfica por coordenadas o nombres de sitios.
- Studio: editor gráfico para diseñar mapas de manera personalizada.
- Vision: es una herramienta para generar experiencias de conducción con realidad aumentada a través de redes neuronales.
- Data: conjuntos de datos adicionales que se agregan a un mapa como: límites administrativos, códigos postales, datos de tráfico, límites de velocidad, etc.

Las principales diferencias con Google Maps API son:

- Aunque ambas plataformas son personalizables, Mapbox es mucho más potente en este aspecto, ya que Google Maps API obliga a utilizar su capa base. Además, Mapbox dispone de un editor de mapas: Mapbox Studio.
- Mapbox tiene un rendimiento superior al de Google Maps API en cuanto a carga de marcadores en mapas.

[37] *Cross Site Scripting*: este tipo de ataque se suele ejecutar a través de campos de entrada de texto o la barra de direcciones del navegador, y permite a un usuario introducir códigos maliciosos escritos en JavaScript principalmente (con etiquetas <script> o <iframe>).

- Ambos servicios son gratuitos hasta cierto límite a partir del cual comienzan a cobrar. El modelo de precios de la API de Google se basa en el uso y solicitud, mientras que la API Mapbox ofrece diferentes opciones de precios para usuarios individuales. Mapbox proporciona hasta 50.000 cargas mensuales de mapa en web gratuitamente. A partir de ahí, los precios varían entre 50.001 y 100.000 cargas, y baja con el aumento en la cantidad de solicitudes o cargas. El precio de Google Maps es más alto y el precio de Dynamic Maps es de 7 dólares por cada 1.000 solicitudes.
- A diferencia de la API de Google, y a fecha en la que se publica este proyecto, Mabpox no exige de entrada la introducción de un número de tarjeta de crédito para empezar a utilizarla, ya que si se sobrepasa el límite gratuito, el servicio se bloquea hasta que se pague por su sobreuso, y entonces sí que habrá que facilitarla.

5. Implementación

5.1. Requisitos de instalación

Recursos necesarios para la instalación:

Software

- NodeJS y NPM. Node Package Management. Debe descargarse desde la URL que ofrece su página web en: **https://nodejs.org/es/download/** (a 5 de diciembre de 2021)
- El *software* ha sido testeado en los siguientes navegadores:
    - Google Chrome Versión 96.0.4664.45 (Build oficial) (64 bits)
    - Mozilla Firefox 94.0.2 (64-bit)
    - Opera Versión:81.0.4196.60

Hardware.

- Para que la aplicación funcione en local como cliente y servidor, los requisitos mínimos son los de un PC de gama baja (año 2021).
- Si cliente y servidor se alojan en internet, al tratarse de una aplicación que es consumida por un número alto de usuarios, los requisitos de *hardware* y *software* son superiores.

5.2. Instrucciones de instalación

5.2.1. Base de datos. Creación de una cuenta en MongoDB Atlas.

La base de datos se instalará directamente en los clústeres de MongoDB Atlas. Por lo tanto, lo primero que habrá que hacer será crearse una cuenta gratuita que nos ofrece 512MB de espacio, suficiente para el propósito de este proyecto. La URL es https://www.mongodb.com/es/cloud/atlas/register (a 5 de diciembre de 2021)

Figura 11. Atlas MongoDB. Paso 1 para restauración de la base de datos.

A continuación, hay que indicar de qué trata la aplicación.

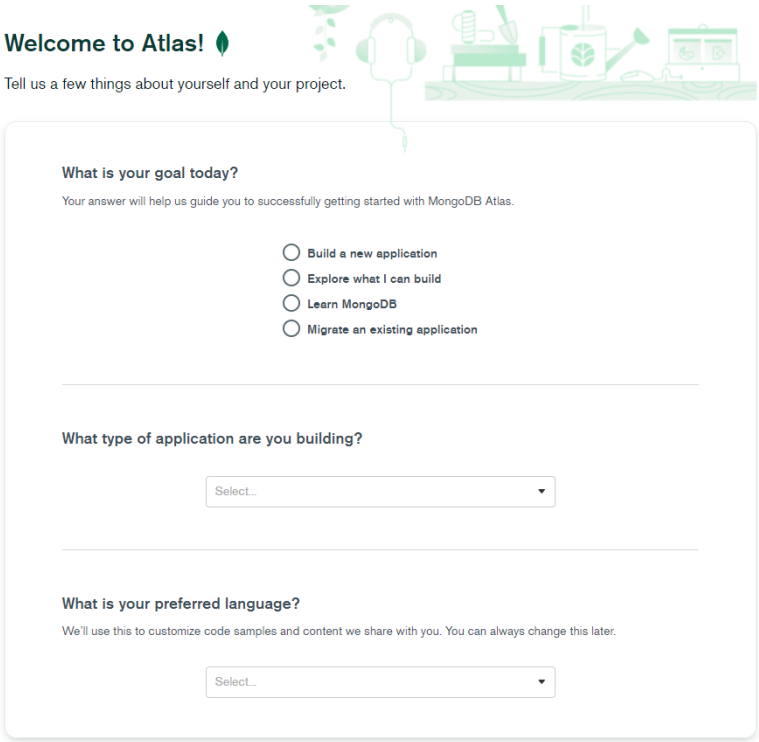


Figura 12. Atlas MongoDB. Paso 2 para restauración de la base de datos.

En el siguiente paso, Atlas MongoDB ofrece los siguientes planes. Habrá que elegir el plan gratuito.

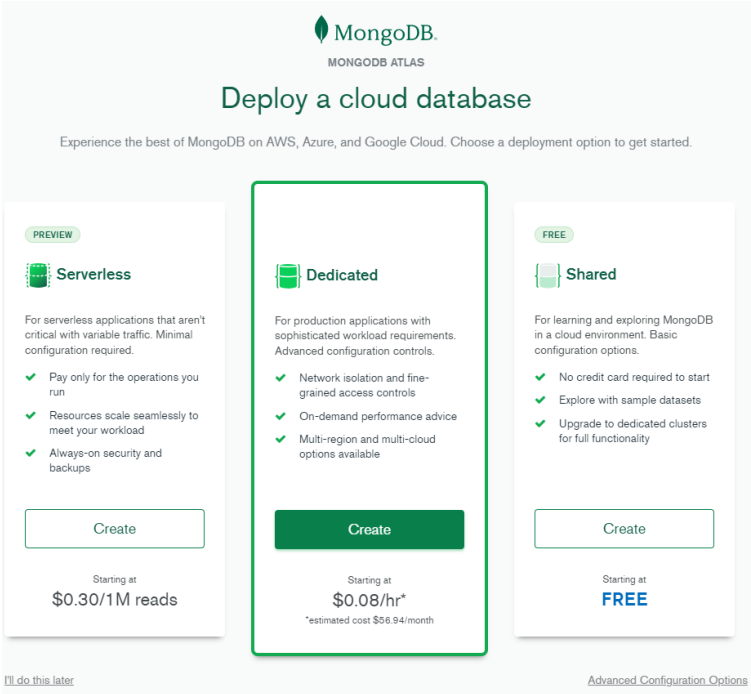


Figura 13. Atlas MongoDB. Paso 3 para restauración de la base de datos.

Ahora se elegirá la opción *shared*. Se selecciona la ubicación más cercana y se pulsará en *Create Cluster*. El clúster se creará entre 3 y 5 minutos.

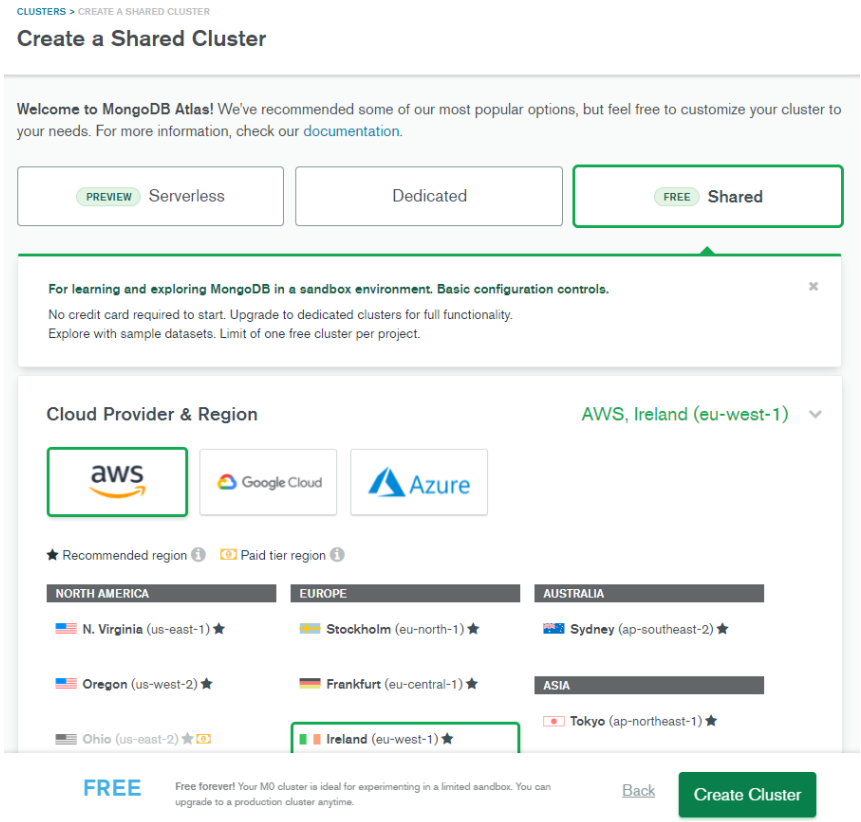


Figura 14. Atlas MongoDB. Paso 4 para restauración de la base de datos.

Después, hay que crear un usuario y una *password* para la base de datos. Habrá que apuntarlos.

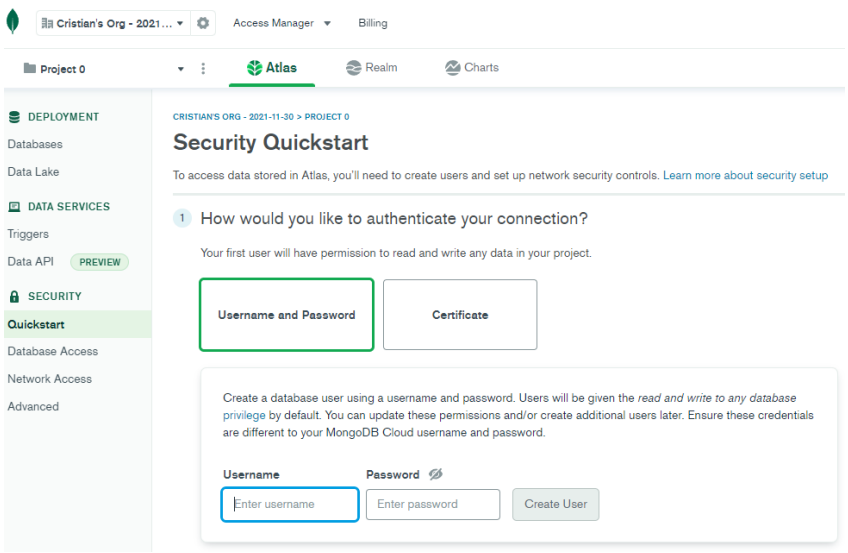


Figura 15. Atlas MongoDB. Paso 5 para restauración de la base de datos.

A continuación, se hace clic en *Databases*, dentro del apartado *DEPLOYMENT* del menú lateral izquierdo, y se pulsa en *Connect*.

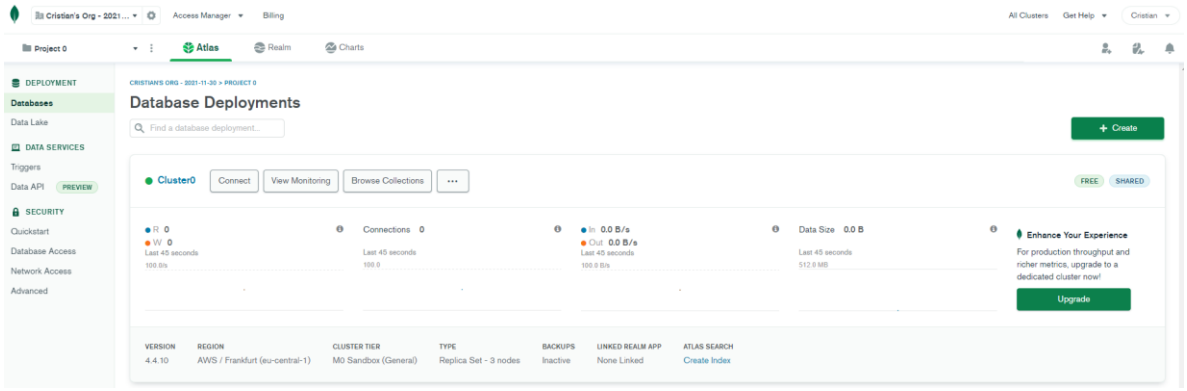


Figura 16. Atlas MongoDB. Paso 6 para restauración de la base de datos.

Se dará autorización a cualquier IP para permitir el acceso a la base de datos creada (*Allow Access from Anywhere*).

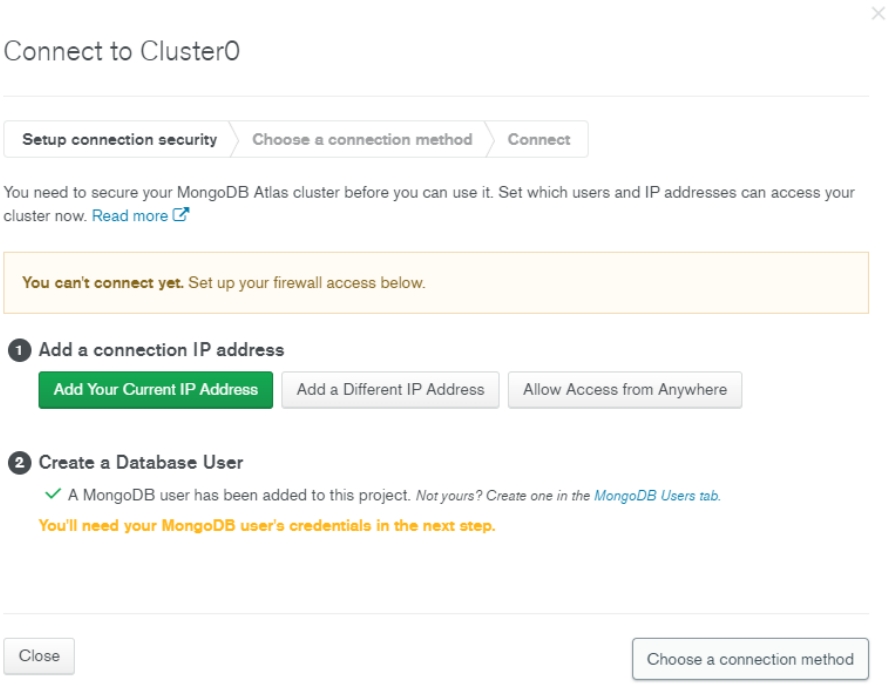


Figura 17. Atlas MongoDB. Paso 7 para restauración de la base de datos.

En este punto, hay que recalcar que, una vez abierto el formulario de dos campos (IP *Address* y *Description*), hay que pulsar en el botón verde titulado *Add IP Address* para que el *firewall* de Atlas MongoDB permita las conexiones, si no, será imposible conectarse.

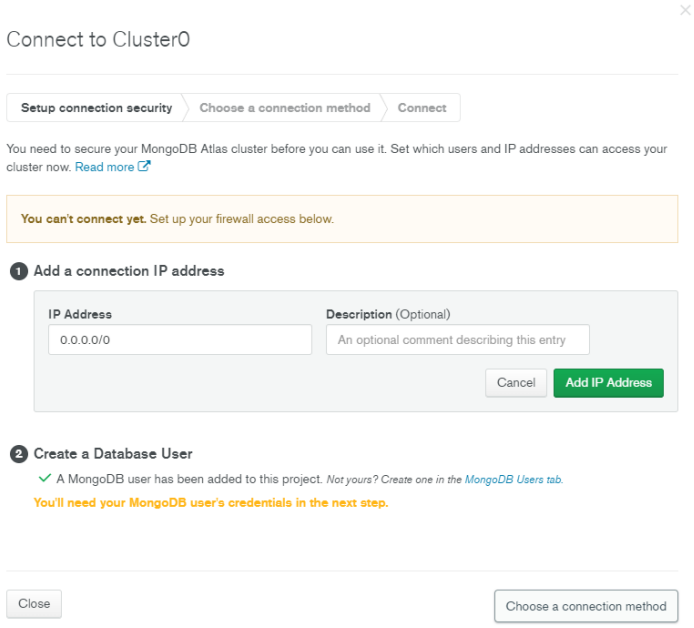


Figura 18. Atlas MongoDB. Paso 8 para restauración de la base de datos.

En este punto, ya podremos seleccionar un método de conexión (*Choose a connection method*)

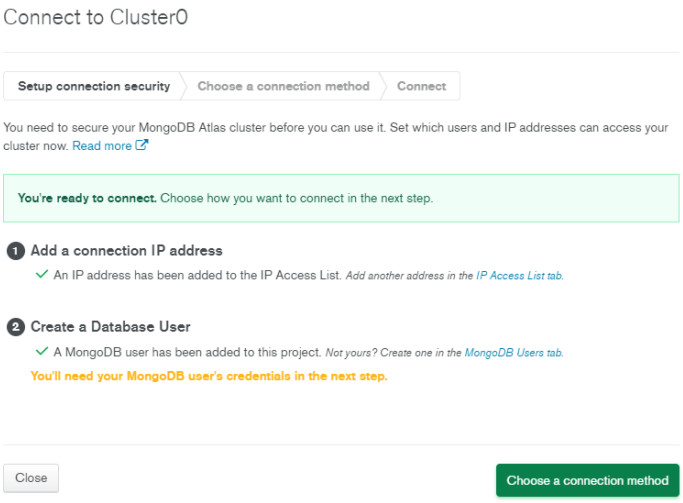


Figura 19. Atlas MongoDB. Paso 9 para restauración de la base de datos.

Se elegirá la tercera opción: *Connect using MongoDB Compass*, de la que solo nos interesa la cadena de conexión. Se recomienda anotarla para restaurar en esta cuenta de Atlas MongoDB la copia de seguridad de la base de datos del proyecto.

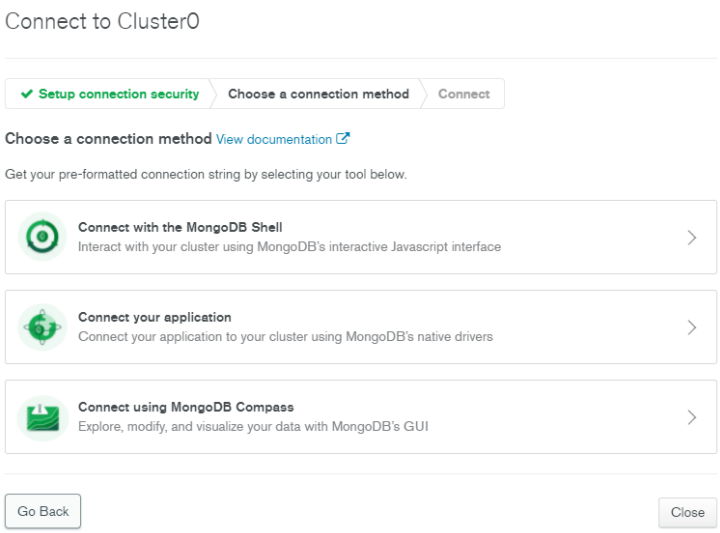


Figura 20. Atlas MongoDB. Paso 10 para restauración de la base de datos.

Esta cadena de conexión se encuentra en el punto 2 de la figura que se ofrece a continuación.

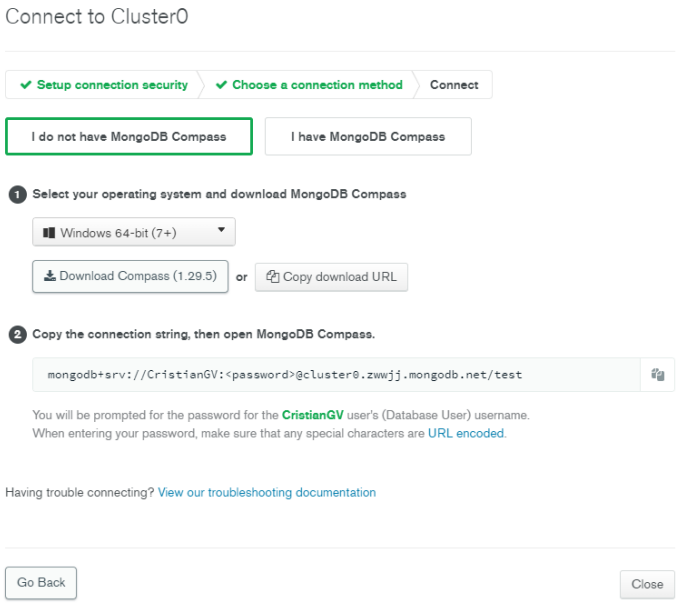


Figura 21. Atlas MongoDB. Paso 11 para restauración de la base de datos.

5.2.2. Base de datos. Restauración de la base de datos busDB.

Para importar toda la base de datos, la manera más rápida será descargarse las MongoDB Database Tools desde <**https://www.mongodb.com/try/download/database-tools**>. Se puede descargar como paquete *msi* para su instalación en Windows, o como *rpm* y *deb* para sistemas Red Hat o basados en Debian. También existe la opción de utilizar los binarios compilados, empaquetados y comprimidos en *tgz* o *zip* para cualquiera de los sistemas operativos que se ofrecen. Por lo tanto, lo recomendable es descargar uno de estos paquetes *zip* con los binarios y, opcionalmente, añadir a la

variable de entorno PATH del sistema operativo la ruta de dichas herramientas con el fin de que sean accesibles en consola de comandos desde cualquier ubicación.

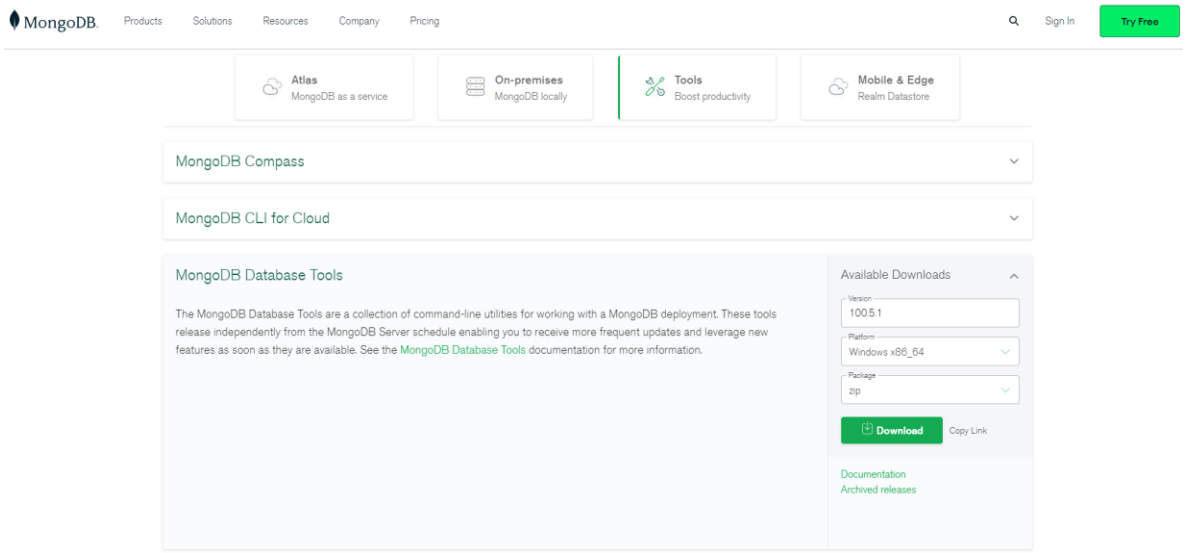


Figura 22. Web de descarga de MongoDB Database Tools.

Una vez descargado el *zip*, se descomprime y se accede al directorio *bin* dentro del mismo paquete ya descomprimido. Para restaurar la copia de seguridad de la base de datos en MongoDB Atlas, con la cuenta ya creada y configurada, se seguirán los siguientes pasos:

- Crear un directorio denominado *dump* dentro de *bin*.
- Descomprimir la base de datos dbBus.zip, y copiar el directorio busDB en *bin/dump*
- Abrir la consola de comandos y situarse en el directorio *bin* de estas herramientas.
- Ejecutar la orden: **./mongorestore.exe --uri CADENA_DE_CONEXIÓN_MONGO** Es importante tener en cuenta que **la cadena de conexión acaba en mongodb.net**, ya que el resto de parámetros, como el nombre de base de datos, no hay que indicarlos. Esta cadena de conexión se obtuvo en el último paso del punto 5.2.1.
- Ejemplo ficticio de cómo restaurar una base de datos desde el directorio *bin* con el comando **mongorestore**:

```
./mongorestore --uri mongodb+srv://<USUARIO>:<PASSWORD>@cluster0.uqnpf.mongodb.net
```

  Hay que omitir los símbolos de mayor y menor y sustituir USUARIO y PASSWORD por sus correspondientes valores. También habrá que sustituir la URL del subdominio de mongodb.net por la que corresponda.
- La herramienta se ejecutará y volcará la base de datos en la cuenta de MongoDB Atlas creada. El resultado deberá ser similar al que se ofrece a continuación:

```
2021-11-30T13:41:01.011+0100    WARNING: On some systems, a password provided directly in a connection
string or using --uri may be visible to system status programs such as `ps` that may be invoked by other
```

```
users. Consider omitting the password to provide it via stdin, or using the --config option to specify a
configuration file with the password.
2021-11-30T13:41:01.906+0100    using default 'dump' directory
2021-11-30T13:41:01.906+0100    preparing collections to restore from
2021-11-30T13:41:01.908+0100    reading metadata for busDB.users from dump\busDB\users.metadata.json
2021-11-30T13:41:01.908+0100    reading metadata for busDB.bstops from dump\busDB\bstops.metadata.json
2021-11-30T13:41:01.908+0100    reading metadata for busDB.buses from dump\busDB\buses.metadata.json
2021-11-30T13:41:01.908+0100    reading metadata for busDB.lines from dump\busDB\lines.metadata.json
2021-11-30T13:41:01.909+0100    reading metadata for busDB.notifications from
dump\busDB\notifications.metadata.json
2021-11-30T13:41:01.909+0100    reading metadata for busDB.routes from dump\busDB\routes.metadata.json
2021-11-30T13:41:01.909+0100    reading metadata for busDB.stops from dump\busDB\stops.metadata.json
2021-11-30T13:41:02.234+0100    restoring busDB.routes from dump\busDB\routes.bson
2021-11-30T13:41:02.305+0100    restoring busDB.stops from dump\busDB\stops.bson
2021-11-30T13:41:02.310+0100    restoring busDB.lines from dump\busDB\lines.bson
2021-11-30T13:41:02.367+0100    finished restoring busDB.lines (5 documents, 0 failures)
2021-11-30T13:41:02.425+0100    restoring busDB.notifications from dump\busDB\notifications.bson
2021-11-30T13:41:02.472+0100    finished restoring busDB.notifications (4 documents, 0 failures)
2021-11-30T13:41:02.503+0100    finished restoring busDB.stops (160 documents, 0 failures)
2021-11-30T13:41:02.543+0100    restoring busDB.users from dump\busDB\users.bson
2021-11-30T13:41:02.549+0100    restoring busDB.bstops from dump\busDB\bstops.bson
2021-11-30T13:41:02.563+0100    restoring busDB.buses from dump\busDB\buses.bson
2021-11-30T13:41:02.590+0100    finished restoring busDB.users (2 documents, 0 failures)
2021-11-30T13:41:02.620+0100    finished restoring busDB.buses (2 documents, 0 failures)
2021-11-30T13:41:02.700+0100    finished restoring busDB.bstops (160 documents, 0 failures)
2021-11-30T13:41:03.530+0100    finished restoring busDB.routes (5 documents, 0 failures)
2021-11-30T13:41:03.530+0100    restoring indexes for collection busDB.users from metadata
2021-11-30T13:41:03.530+0100    index: &idx.IndexDocument{Options:primitive.M{"background":true,
"name":"email_1", "unique":true, "v":2}, Key:primitive.D{primitive.E{Key:"email", Value:1}}, Partial-
FilterExpression:primitive.D(nil)}
2021-11-30T13:41:03.530+0100    restoring indexes for collection busDB.lines from metadata
2021-11-30T13:41:03.530+0100    index: &idx.IndexDocument{Options:primitive.M{"background":true,
"name":"codeLine_1", "unique":true, "v":2}, Key:primitive.D{primitive.E{Key:"codeLine", Value:1}},
PartialFilterExpression:primitive.D(nil)}
2021-11-30T13:41:03.530+0100    restoring indexes for collection busDB.buses from metadata
2021-11-30T13:41:03.530+0100    index: &idx.IndexDocument{Options:primitive.M{"background":true,
"name":"identification_1", "unique":true, "v":2}, Key:primitive.D{primitive.E{Key:"identification",
Value:1}}, PartialFilterExpression:primitive.D(nil)}
2021-11-30T13:41:03.530+0100    no indexes to restore for collection busDB.bstops
2021-11-30T13:41:03.530+0100    no indexes to restore for collection busDB.notifications
2021-11-30T13:41:03.530+0100    no indexes to restore for collection busDB.routes
2021-11-30T13:41:03.530+0100    restoring indexes for collection busDB.stops from metadata
2021-11-30T13:41:03.530+0100    index: &idx.IndexDocument{Options:primitive.M{"background":true,
"name":"CodParada_1", "unique":true, "v":2}, Key:primitive.D{primitive.E{Key:"CodParada", Value:1}},
PartialFilterExpression:primitive.D(nil)}
2021-11-30T13:41:03.723+0100    338 document(s) restored successfully. 0 document(s) failed to restore.
```

5.2.3. Cliente y servidor

La instalación en local de cliente y servidor es un proceso sencillo, ya que el gestor de paquetes de JavaScript (NPM) llevará a cabo gran parte de la tarea. El cliente corre en el puerto 3000, y el servidor, en el 3001. Primero, hay que descomprimir el *zip* que contiene el proyecto con los directorios bus_server y bus_client:

Servidor

- Para instalar el servidor, desde consola de comandos, se accede al directorio bus_server.
- Hay que lanzar el comando **npm install** con el que se instalan todas las dependencias. Suele tardar unos minutos.
- Terminado el proceso de instalación, se accede al archivo *.env* que se encuentra en el directorio raíz del servidor (bus_server) y se modifican los valores:
    - **DBPWD**=<Password_de_Base_de_Datos>
    - **DBUSER**=<Usuario_Base_de_Datos>
    - **DB**=<Nombre_Base_de_Datos>
    - **DBDOMAIN**=<dominio_cluster_ATLAS_MONGODB >

  Estos valores se obtienen desde la cadena de conexión comentada en el apartado anterior.

  Ejemplo:

```
DBPWD=r5HAgCH2sLH35Tf
DBUSER=busApp
DB=busDB
DBDOMAIN=cluster0.mdqel.mongodb.net
```

- Para finalizar, se lanza el servidor desde consola con el comando **npm start**.

```
Server corriendo en puerto: 3001
DB Online
```

Cliente

- Para instalar el cliente, desde consola, se accede al directorio bus_client.
- Hay que lanzar el comando **npm install** con el que se instalan todas las dependencias. Este proceso será más largo que en el caso del servidor, pues contiene más dependencias.
- Terminado el proceso, se lanza **npm start** para arrancar el servidor de la aplicación.
- Automáticamente, se abrirá una página en el navegador del sistema, y se podrá acceder a la aplicación desde la dirección: **http://localhost:3000**

Autobuses en tiempo real

Abrir desde navegador la URL **http://localhost:3000** con la cuenta **bus@bahiabus.com** y *password* **tfg-Multimedia** . Luego, entrar en *Demo Buses*, seleccionar el autobús o autobuses que se quieren simular y clicar en iniciar. Visitar el punto 6.3.2. para más información.

6. Demostración

6.1. Instrucciones de uso

Si se han seguido los pasos indicados en el apartado 5.2. para llevar a cabo la instalación de la aplicación, a la hora de arrancar el cliente desde consola de comandos (con el servidor ya funcionando), el sistema operativo abrirá la primera vista del *frontend* a través del navegador en la URL del *localhost,* en el puerto 3000. Una vez allí, accederemos a la *landing page*[38] de la aplicación.

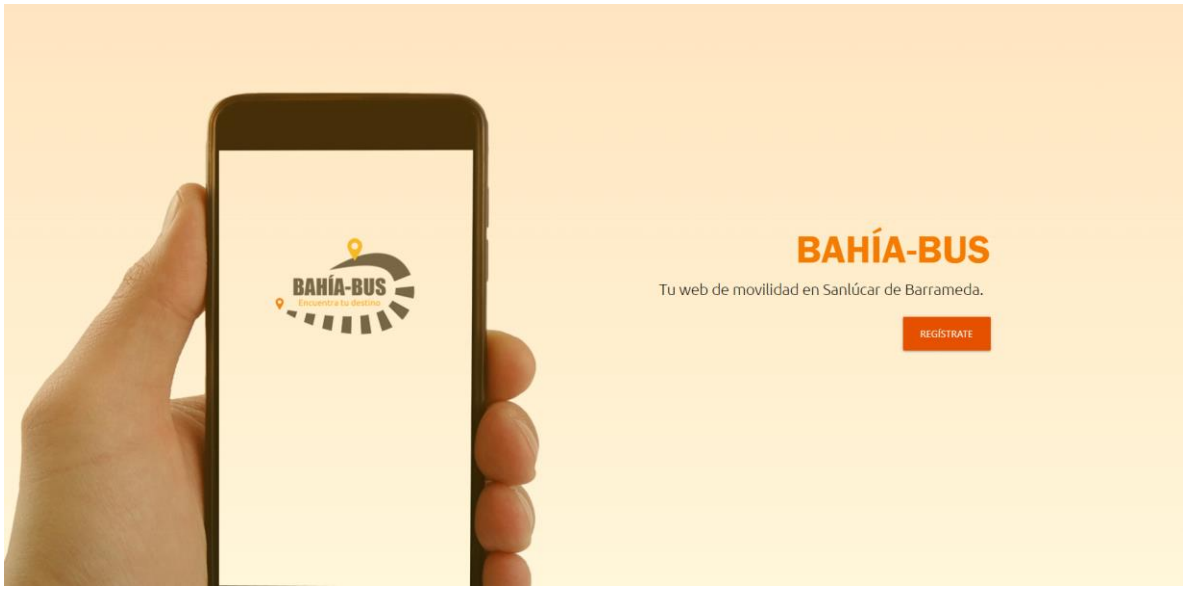


Figura 23. *Landing page* de Bahía-Bus

Si se pulsa el botón de registrarse, la página conducirá al usuario a un formulario de *login* de usuario o registro. La cuenta de *admin* es **admin@bahiabus.com** | *password*: **tfg-Multimedia**

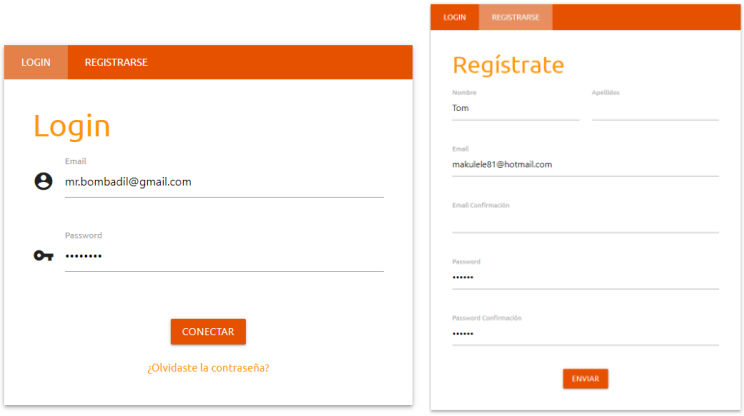


Figura 24. Formulario de login/registro de Bahía-Bus

[38] Una *landing page*, traducido del inglés al español como página de aterrizaje, es aquella a la que se llega a través de un anuncio publicitario. También se denomina de esta manera a una página de inicio en la que se explica de una manera resumida el producto que se pretende utilizar.

Para ver instrucciones más detalladas, consultar el vídeo demostrativo ubicado en la siguiente URL: https://youtu.be/Nw1W7wA8Xss?t=433

También se puede acceder directamente a una demostración de la aplicación desde **https://bahiabustfg.herokuapp.com**

Datos de acceso de los diferentes roles de usuario de ejemplo almacenados en la base de datos: **admin@bahiabus.com, bus@bahiabus.com**, **usuario@bahiabus.com** (todos con **contraseña tfg-Multimedia**)

6.2. Prototipos

A continuación se presentan los *wireframes* trabajados para este proyecto.

6.2.1. Prototipos Lo-Fi

Las páginas se estructuran mediante un *grid* de 12 columnas.

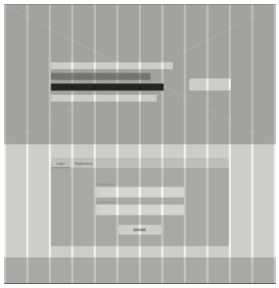


Figura 25. *Grid* de 12 columnas.

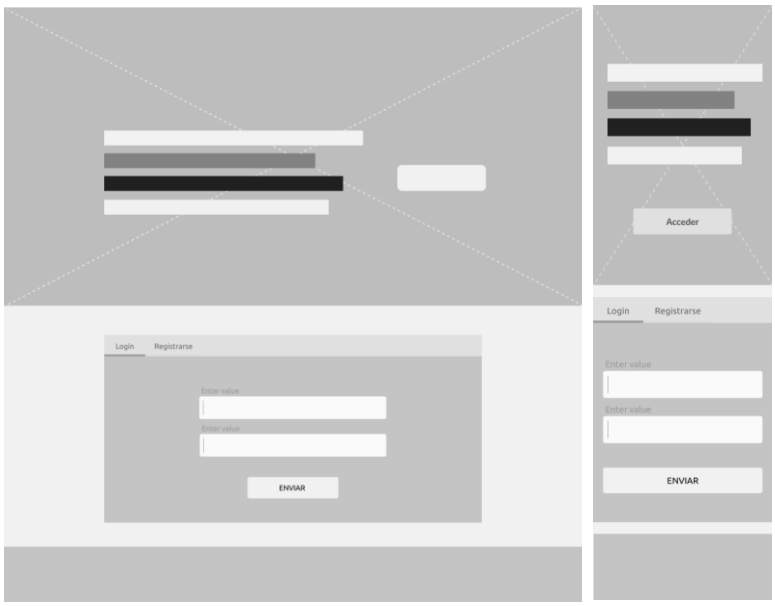


Figura 26. *Index*. Pantalla de presentación y registro. Versión escritorio y móvil.

Se trata de una pantalla de presentación y aterrizaje con una llamada a la acción. Esta página conduce, mediante animación con *scroll,* al formulario de inicio de sesión o registro que hay justo debajo, y que se integra en la misma página de inicio.

6.2.2. Frontend de usuario.

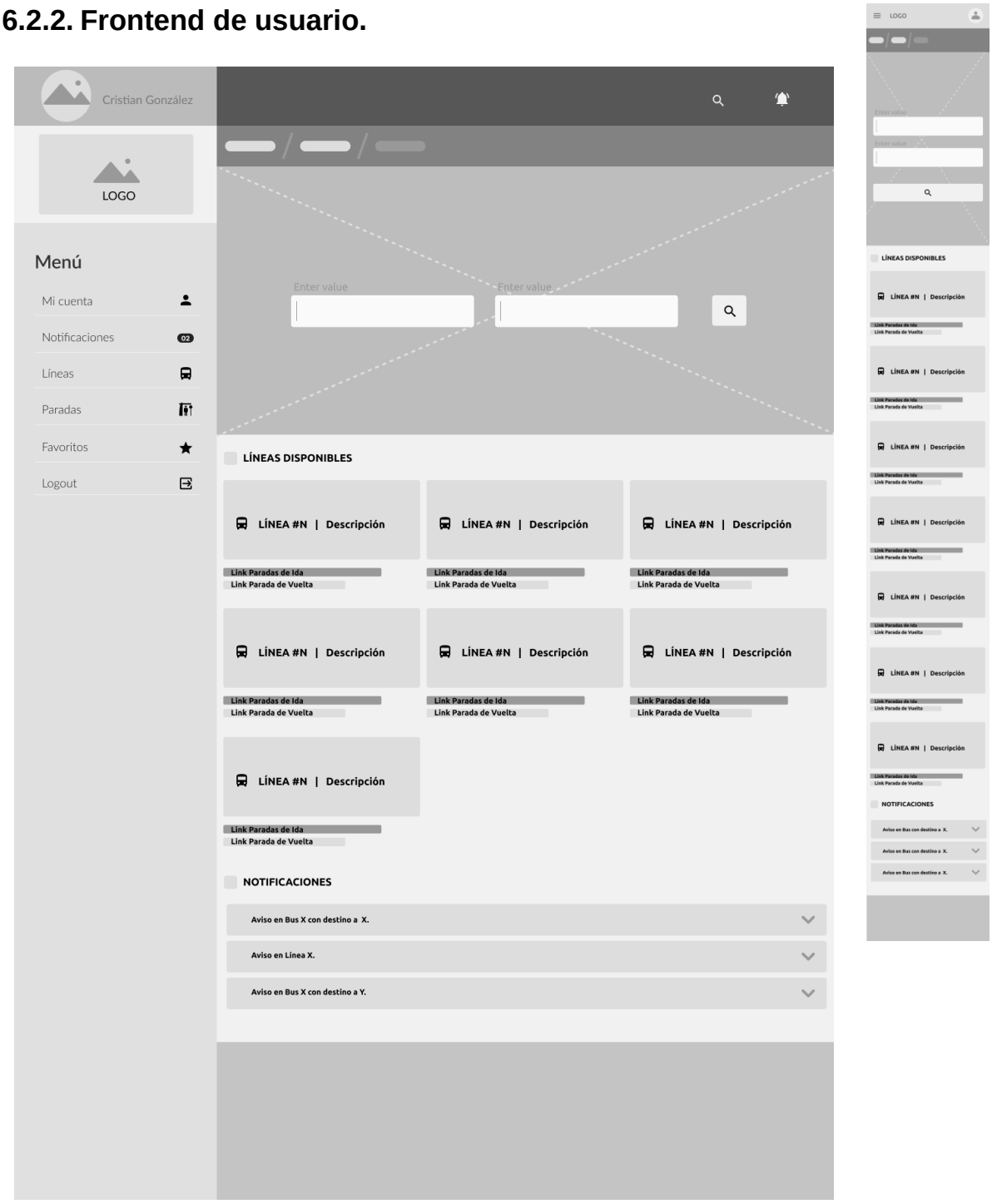


Figura 27. *Dashboard*. Versión escritorio y móvil.

Una vez *logueados*, se encuentra la página de *Dashboard*. Se trata de la pantalla de inicio de la aplicación. Esta página, así como las siguientes, se divide en cuatro partes:

- La cabecera, subdividida en tres: a derecha hay un acceso directo al buscador y otro para notificaciones; a izquierda, aparece el avatar del usuario que ha iniciado sesión junto a su nombre; abajo, dedicada al *breadcrumbs*.

- A la izquierda está el menú de navegación. Cabe señalar que este menú es el mismo que aparece en la versión móvil cuando se hace clic en el icono de hamburguesa. Justo debajo se ubican los enlaces a las diferentes páginas. Encima de él aparece el símbolo gráfico y logotipo de la aplicación.

- A la derecha se sitúa el contenido principal, en el que aparecen diferentes componentes con contenido dinámico de la aplicación. En el caso del *Dasboard,* aparecen un formulario buscador para averiguar la ruta desde una parada o ubicación y una lista de las líneas disponibles de autobús (mediante *cards*).

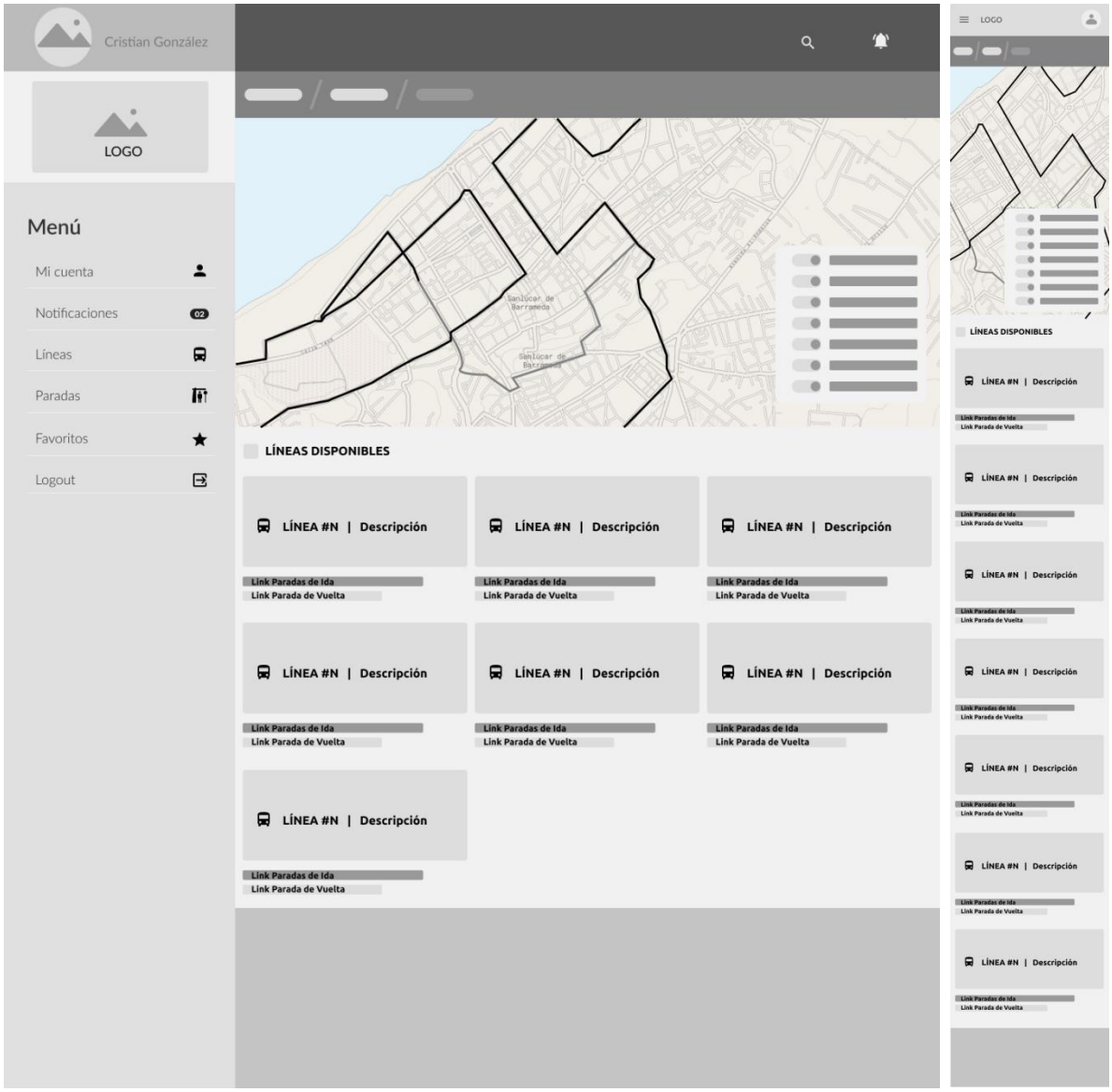


Figura 28. *Lines*. Versión escritorio y móvil.

En la página *lines* aparecen las líneas disponibles con su recorrido dibujado mediante líneas de colores a través de un componente de mapas que dispone de un panel para filtrar cada una de ellas. Debajo del mapa aparece un listado de líneas disponibles con sus enlaces a paradas igual que el que se obtenía en la página *Dashboard*.

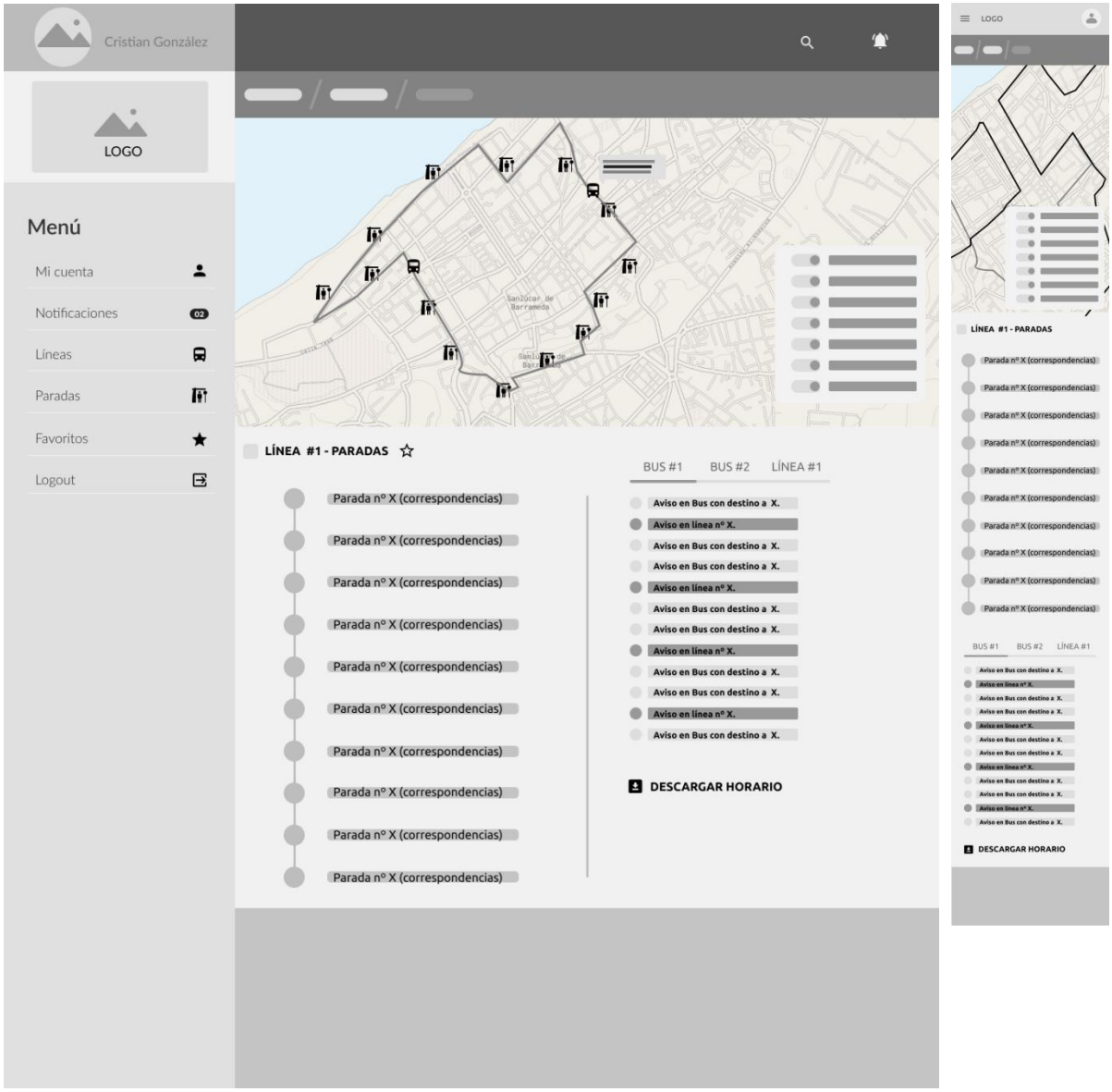


Figura 29. *ViewLine*. Versión escritorio y móvil.

En *ViewLine* se observa el recorrido de la línea, así como sus paradas, a través del componente interactivo de mapa. Reutiliza el mismo panel de filtrado de *Lines* para las paradas. Los iconos de parada son interactivos. En cada clic, se muestra información sobre su ubicación y nombre.

Junto al título de la parada, ya en el apartado de abajo, existe un icono interactivo en forma de estrella que añade a la lista de favoritos la línea en cuestión. En este apartado, además, aparecen en orden y de manera detallada cada una de las paradas con la información de correspondencias con otras líneas. A la derecha de este, se muestran los avisos relacionados con los autobuses de la línea o de la línea en sí. Este componente consiste en un conjunto de solapas, o *tabs,* que reaccionan al clic del usuario para mostrar el contenido. También dispone de un enlace a la descarga del horario de las paradas en formato PDF proporcionado por la web del Grupo Avanza.

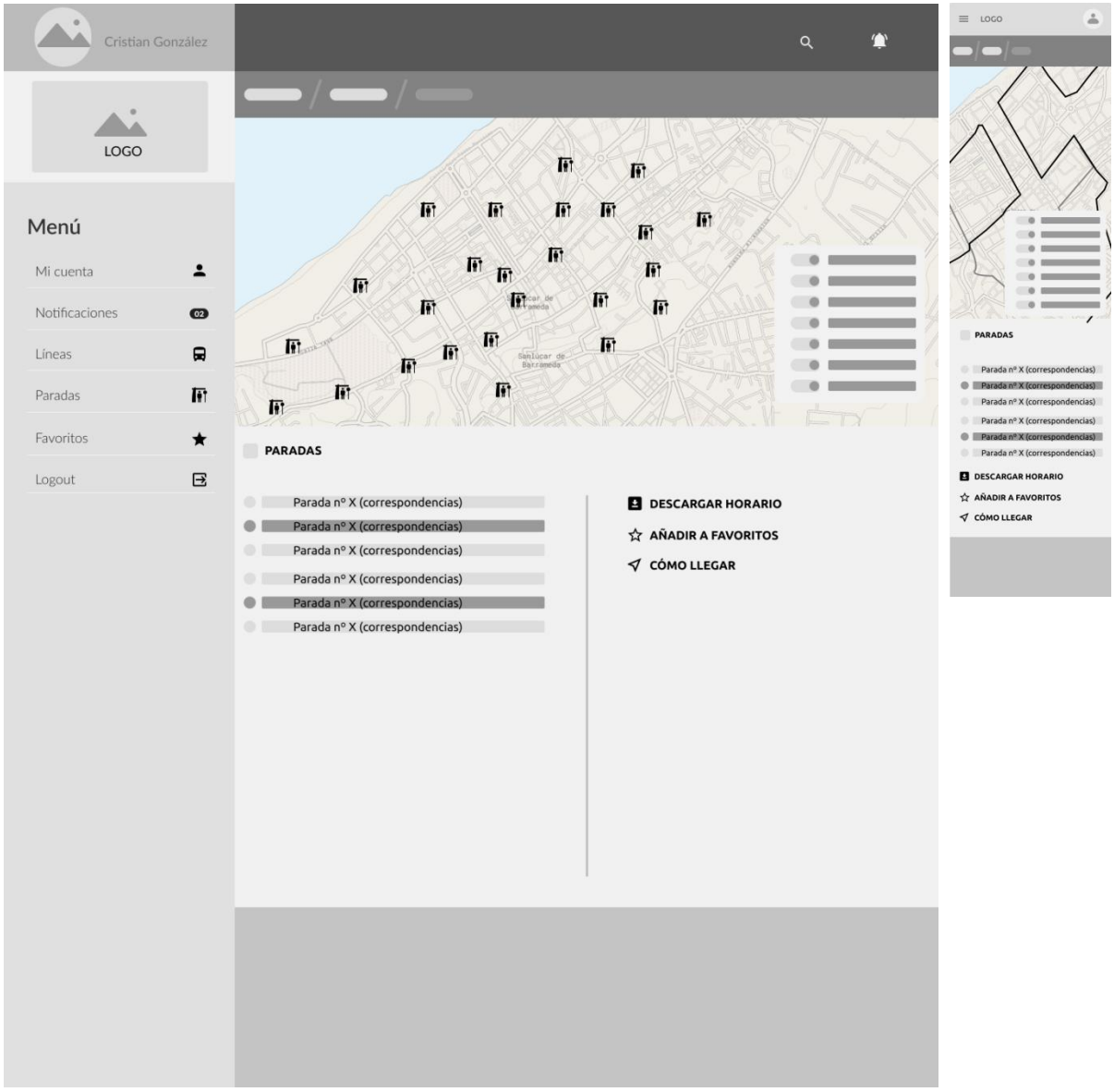


Figura 30. *Stops*. Versión escritorio y móvil.

En *Stops*, se muestran, por un lado, todas las paradas urbanas que existen en Sanlúcar de Barrameda a través del mapa interactivo, con su correspondiente filtro. En los apartados inferiores aparece información adicional sobre correspondencias de paradas, y un botón similar al que hay en ViewLine para añadir a favoritos.

También existe la posibilidad de encontrar el camino hasta la parada si se pulsa sobre CÓMO LLEGAR. En este caso, se tomará la ubicación del usuario como origen, y la parada señalada en el apartado de la izquierda como destino. Los resultados de la búsqueda aparecerán en la página de resultados de búsqueda *SearchResults*.

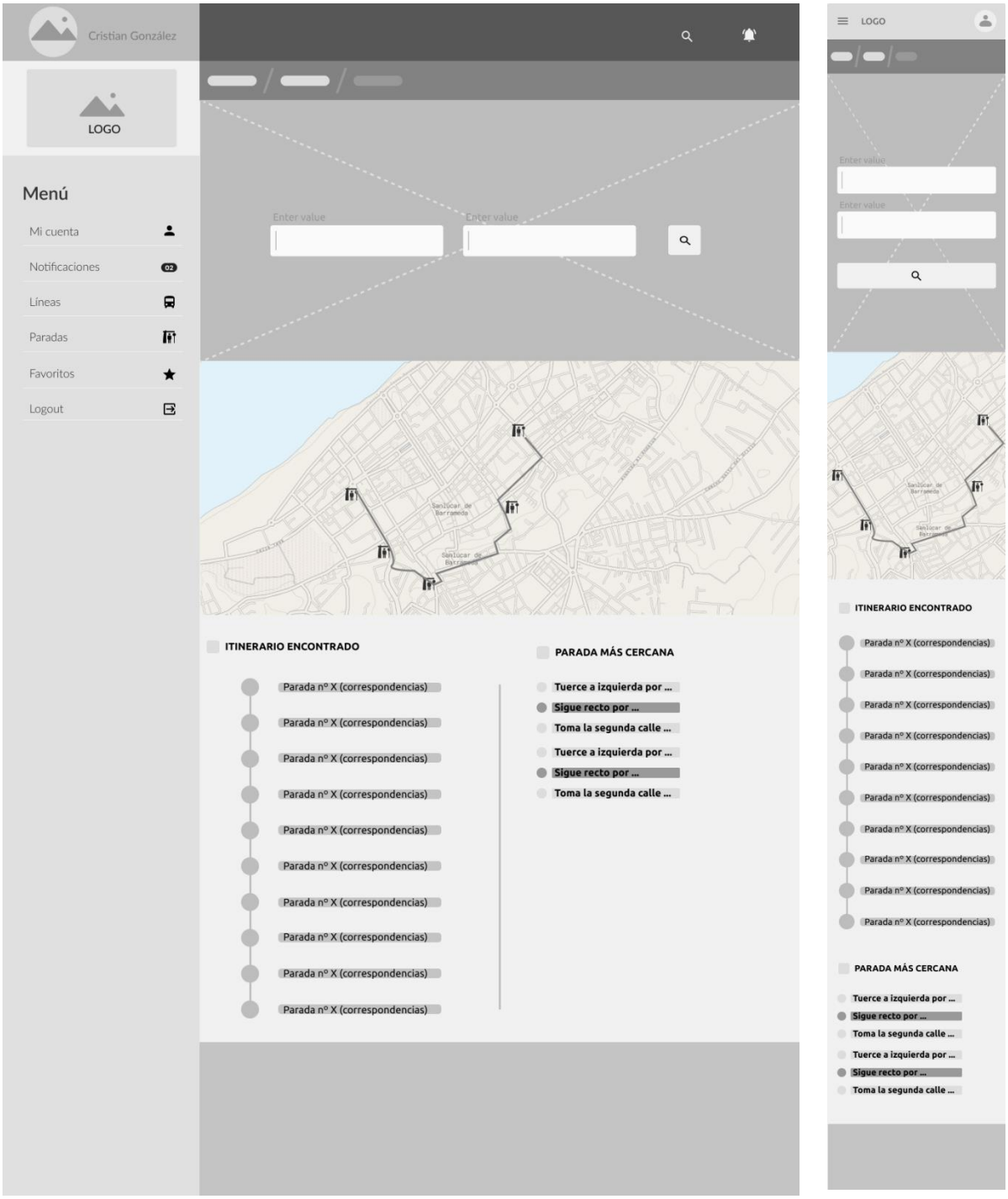


Figura 31. *SearchResults*. Vesión escritorio y móvil.

En *SearchResults,* se muestran los resultados obtenidos por el sistema tras la búsqueda de un itinerario. En el mapa se muestra el resultado con el itinerario y las paradas interactivas en forma de icono gráfico. En los apartados inferiores aparece a la izquierda un listado de las paradas que hay que recorrer (con la correspondencia con todas sus líneas) y, a la derecha, instrucciones sobre cómo llegar a la parada más cercana.

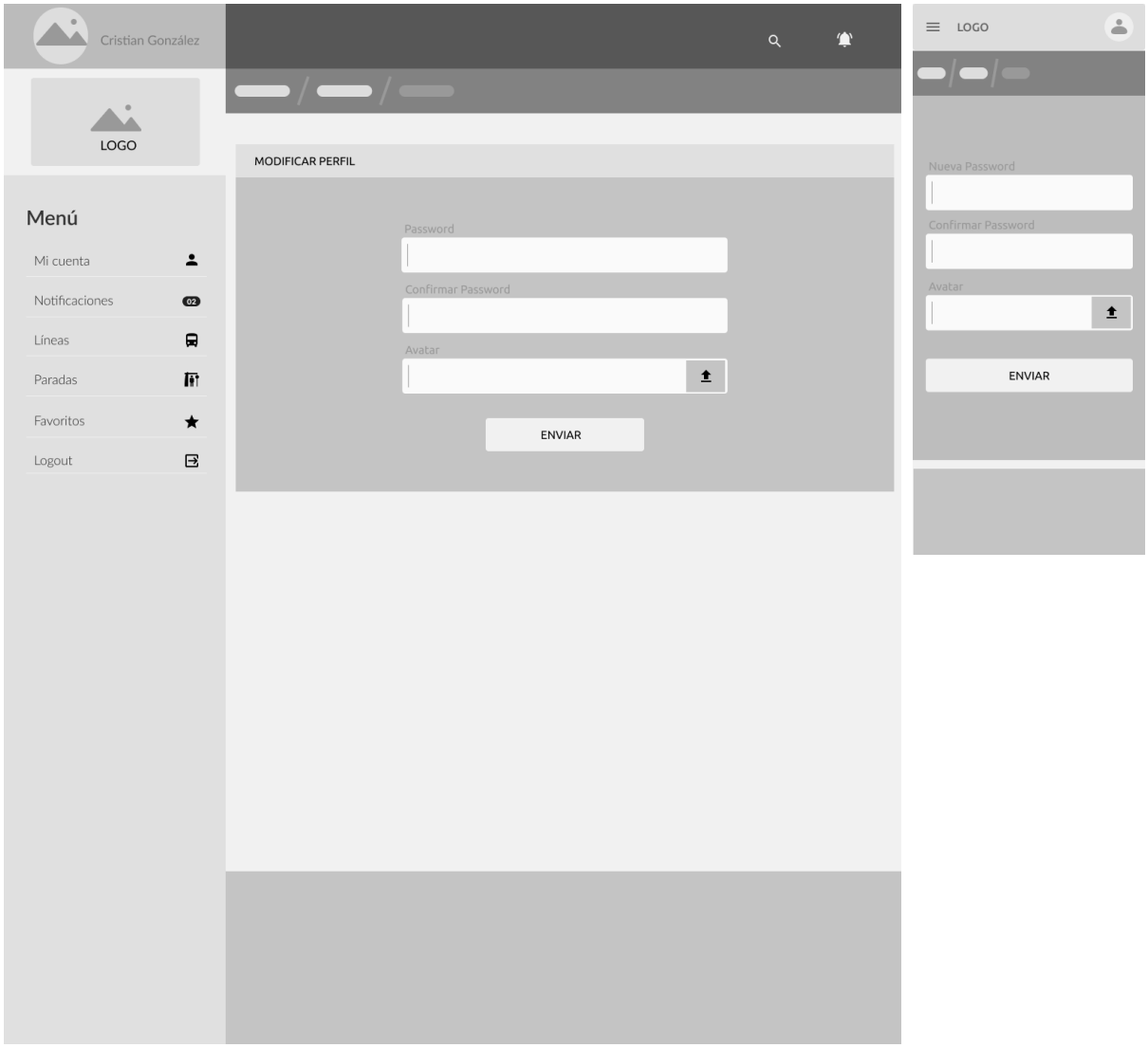


Figura 32. *User*. Versión escritorio y móvil.

User es una página que muestra un formulario de tres campos. Dos campos de texto y uno para envío de imagen de mapa de bits.

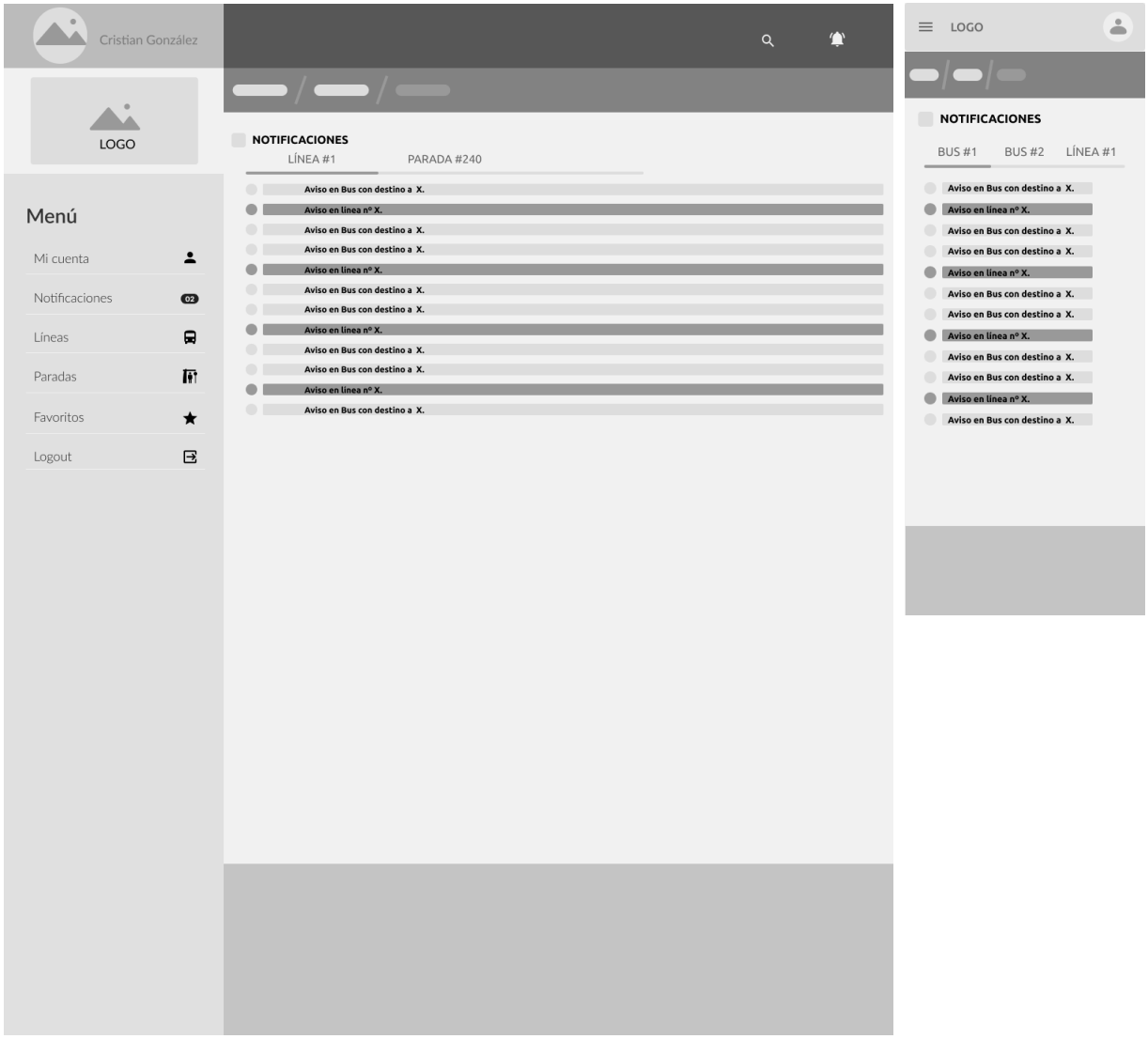


Figura 33 . Notifications. Versión escritorio y móvil.

Notifications muestra todas las notificaciones de líneas y autobuses en forma de listas. Incluye pestañas que clasifican los avisos según la procedencia.

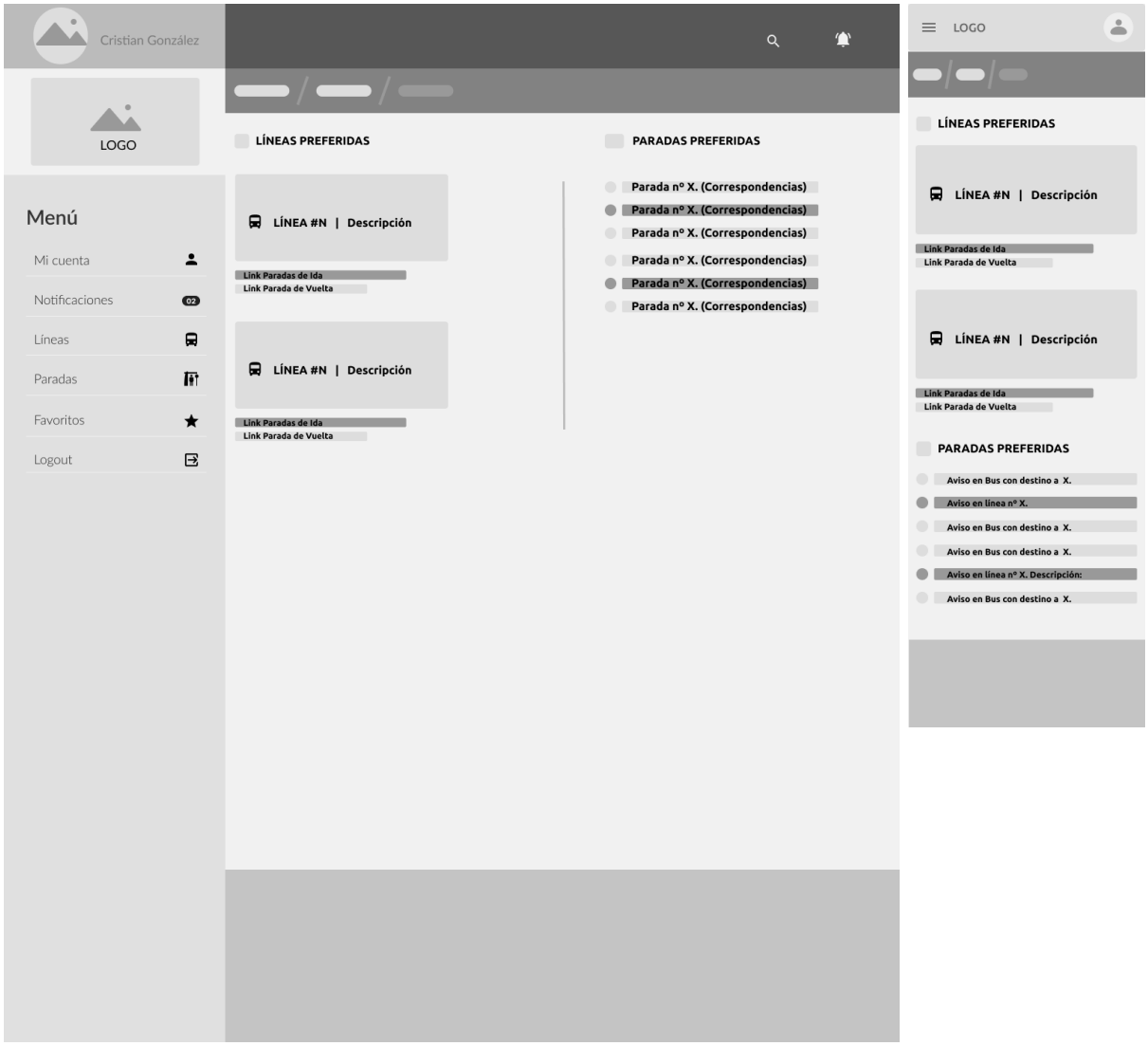


Figura 34. *Favorites*. Versión escritorio y móvil.

Favorites muestra los enlaces directos a paradas o líneas marcadas como favoritas por el usuario. Las líneas se muestran como una lista de *cards,* y las paradas como una lista desordenada.

6.2.3. Panel de Administración.

Los *wireframes* diseñados para la zona de administración son muy similares, así que se agruparán y mostrarán de manera más resumida.

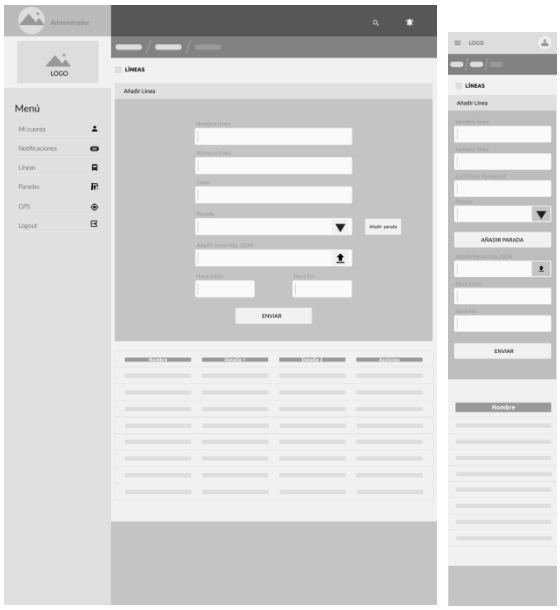


Figura 35. *Lines*. Versiones escritorio y móvil.

Dashboard reutiliza las *cards* para conceder al administrador acceso directo desde el *Dashboard* hacia el resto de apartados. *Lines*, por su parte, emplea un esquema muy repetido en el resto de páginas, ya que utiliza un formulario y una tabla para llevar a cabo todo el CRUD de las líneas de autobús.

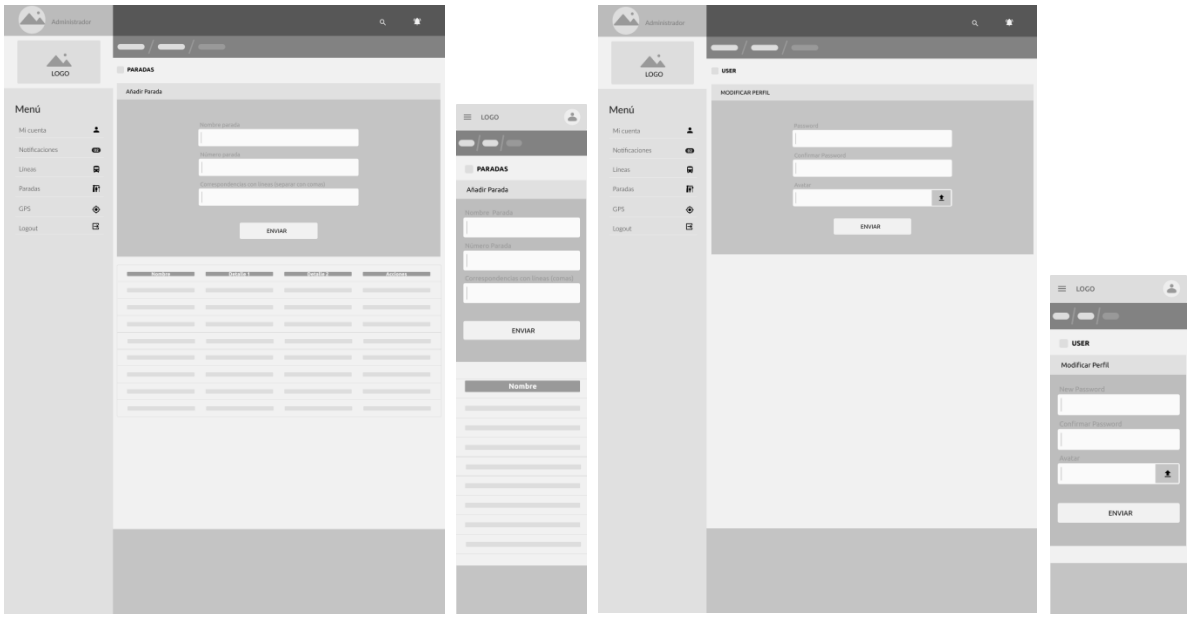


Figura 36. A la izquierda, *Stops*, y a la derecha, *User*. Versiones de escritorio y móvil.

Stops, repite el esquema de *Lines* con un formulario y una tabla para CRUD. *User*, en este caso, solo hace uso de un formulario.

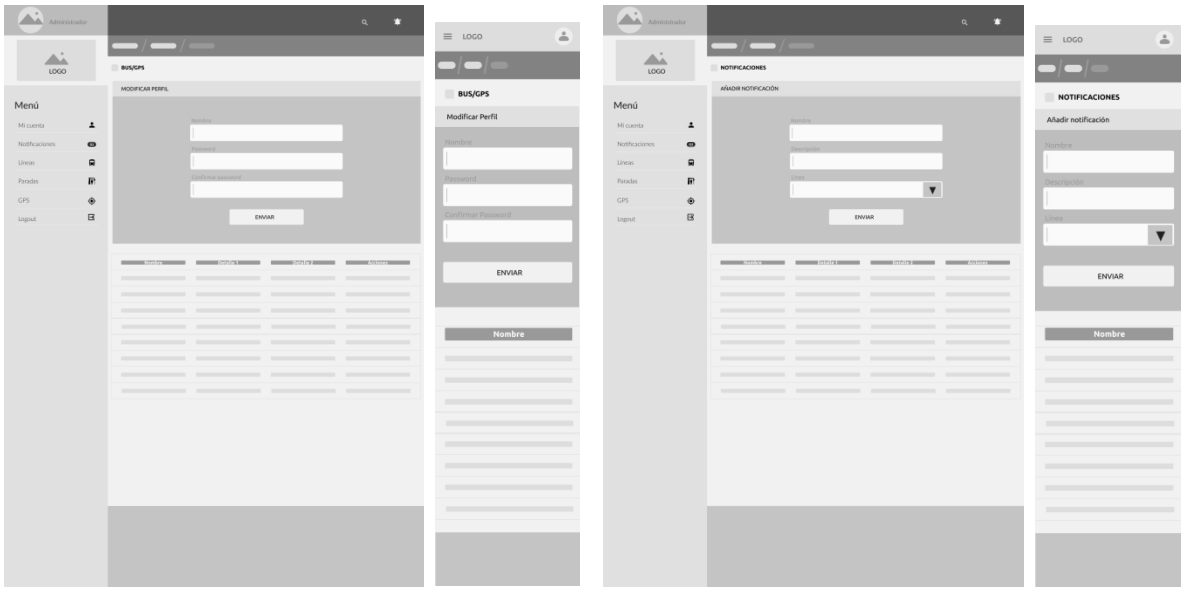


Figura 37. A la izquierda, *BUS_GPS*, y a la derecha, *Notifications*. Versiones de escritorio y móvil.

BUS_GPS sigue el mismo patrón CRUD que el resto de páginas. Consta de un formulario y una tabla. *Notifications* continúa el mismo patrón.

6.3. Ejemplos de uso del producto

6.3.1. Búsqueda de itinerarios en autobús.

Una vez conectados a la aplicación, en la primera página se encuentra un formulario de búsqueda de itinerarios. La búsqueda se puede llevar a cabo introduciendo paradas o lugares que no incluyan parada concreta. En el primer ejemplo, se llevará a cabo una búsqueda desde una parada de inicio y una parada de destino. El sistema se encargará de encontrar la ruta más corta entre líneas de autobuses y se visualizará en mapa el itinerario.

En este primer ejemplo, se ha introducido como parada de origen ROT-CANTILLO (Rotonda del Cantillo), y como parada de destino, EL SAUCE:

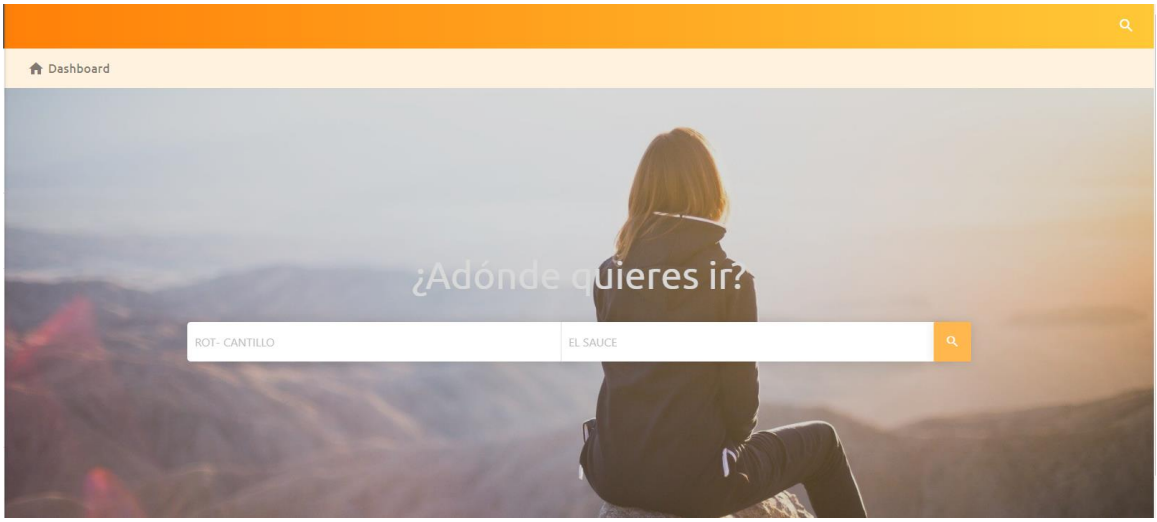


Figura 38. Formulario de búsqueda.

A continuación, el sistema hará el cálculo de la ruta y el resultado se visualizará tanto en el mapa, como en el apartado de itinerario y de ruta a pie hasta parada de origen:

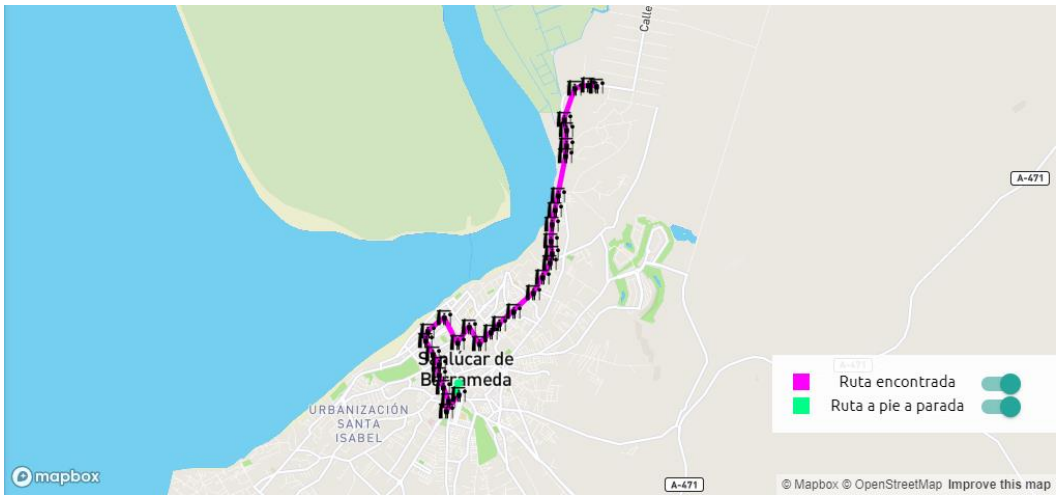


Figura 39. Resultado de búsqueda en mapa.

En rosa, aparece el itinerario de paradas que tienen que recorrer las líneas de autobús, y en verde se muestra la ruta a pie desde la ubicación real del usuario hasta la primera parada de destino. Además, se muestran en el mapa los iconos de todas las paradas encontradas para conformar el itinerario hasta el destino indicado.

A continuación del apartado de mapa, aparece, en el listado izquierdo, el itinerario de líneas de autobuses que hay que seguir para alcanzar destino, y a la derecha, la ruta a pie hasta la primera parada desde la ubicación del usuario según GPS.

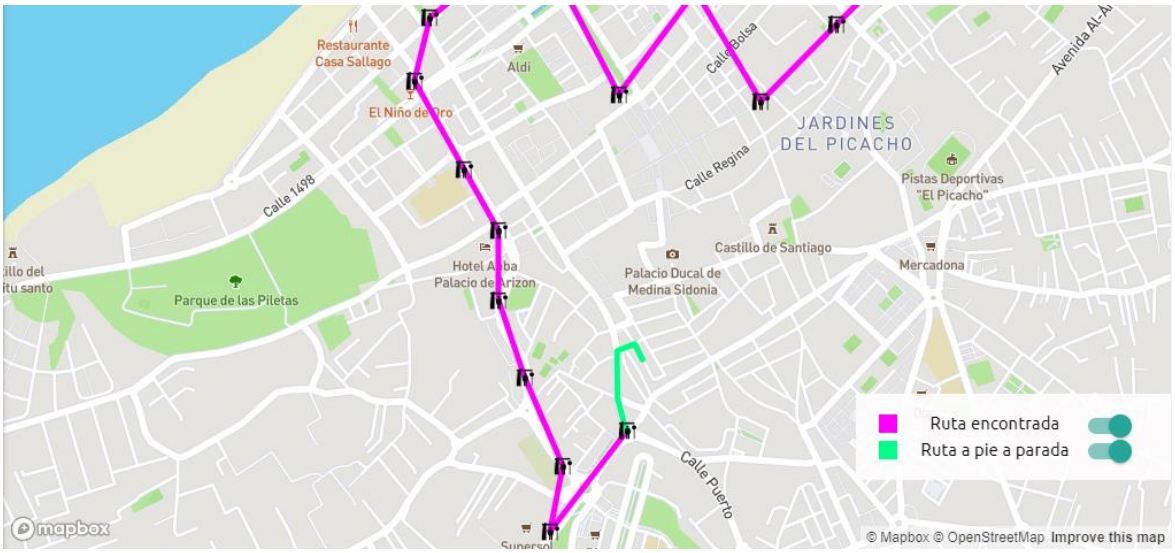


Figura 40. Itinerario con más detalle.

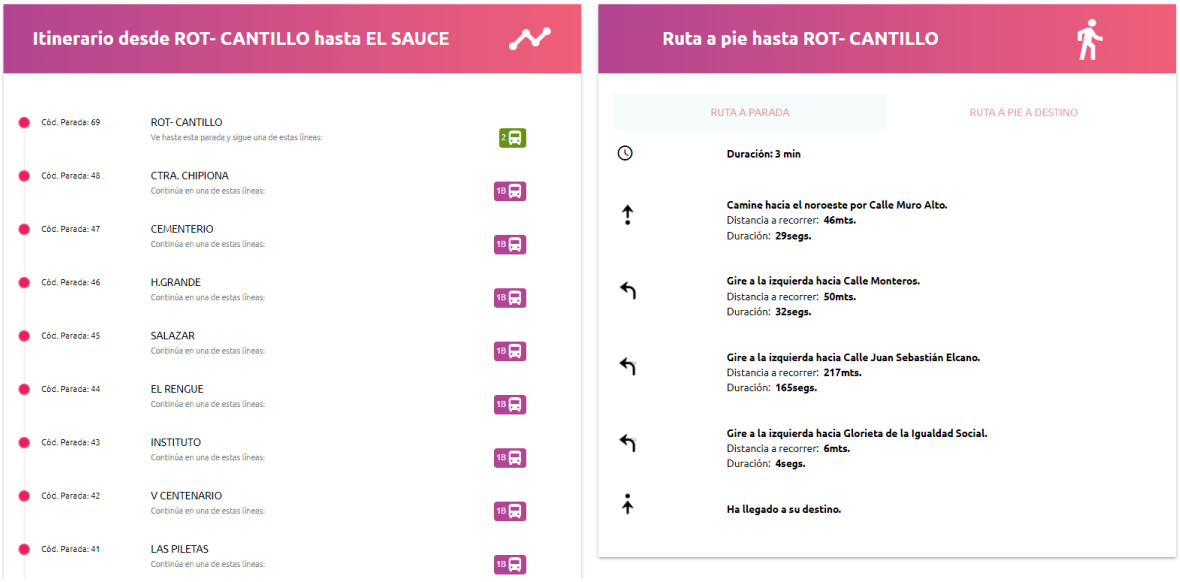


Figura 41. Instrucciones para llegar al destino. A la izquierda, el itinerario de autobús. A la derecha, las instrucciones de llegada parada origen.

En el segundo ejemplo, se hace una búsqueda de origen y destino sin buscar la parada. Es decir, se introducirá el nombre de una calle que no disponga de parada, y un destino en concreto con las mismas características. El sistema se encargará de trazar la ruta hasta la parada más cercana.

En el ejemplo que se expone a continuación, se lanza una búsqueda cuyo origen se sitúa en el Camino de Rompeserones, y finaliza en la Calle Fragata; ambas, sin parada de autobús.

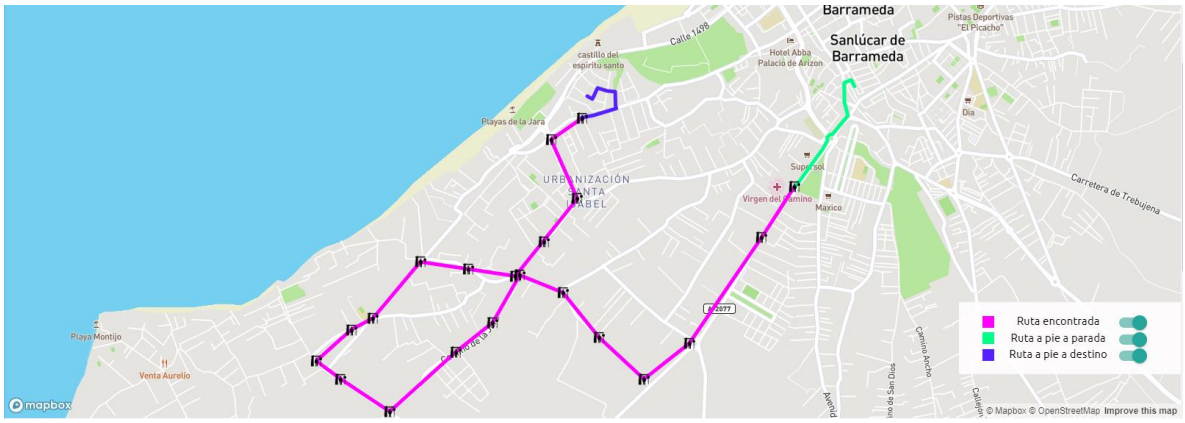


Figura 42. Itinerario encontrado para una búsqueda sin paradas.

En azul, se marca la ruta a pie hasta la Calle Fragata; en verde, la ruta hasta la primera parada. Para la Calle Fragata, el sistema encuentra la parada más cercana en Los Colonos, mientras que para Rompeserones, la parada más cercana es la del hospital.

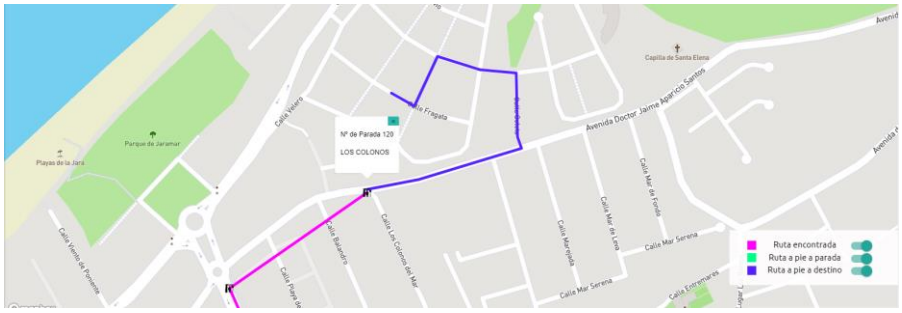


Figura 43. Última parada más cercana al destino.

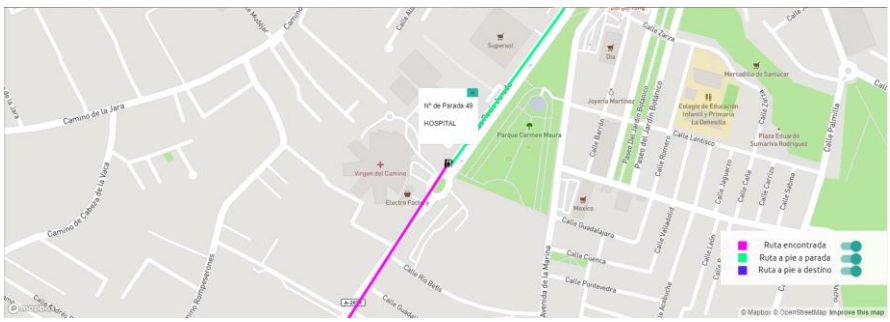


Figura 44. Parada más cercana al origen.

La aplicación también lanza información de la ruta a pie que hay que llevar a cabo hasta el destino indicado: la Calle Fragata.

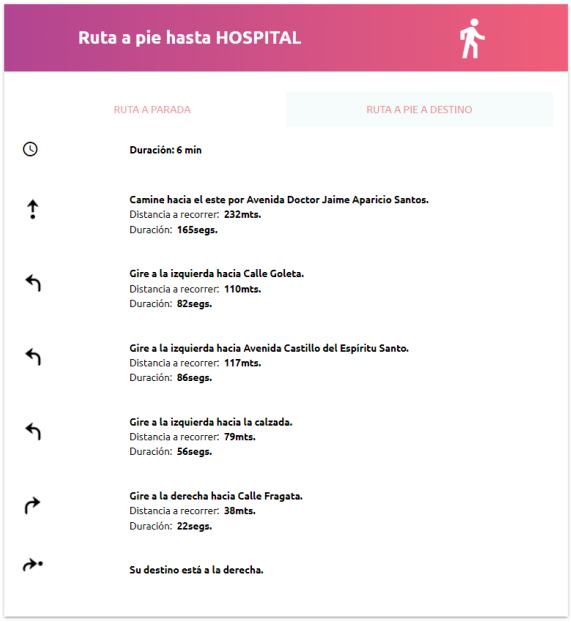


Figura 45. Ruta a pie hasta el destino solicitado.

6.3.2. Visualización de los autobuses en tiempo real.

Los autobuses son una simulación en tiempo real, por lo que hay que conectarse a la aplicación con la cuenta de autobús (**bus@bahiabus.com** | **tfg-Multimedia**) y acceder a la opción *Demo Buses* del menú de navegación. Se recomienda utilizar el navegador Mozilla Firefox, ya que el script emplea una función *setInterval* que lanza mensajes al *socket* cada pocos milisegundos, y en el caso de navegadores como Google Chrome, el proceso se duerme pasados unos minutos desde su lanzamiento si la ventana del mismo pierde el foco, lo que produce que los *scripts* dejen de emitir mensajes y desaparezca el efecto de los autobuses moviéndose en tiempo real. En Mozilla Firefox, no ocurre.

Figura 46. Autobuses en tiempo real.

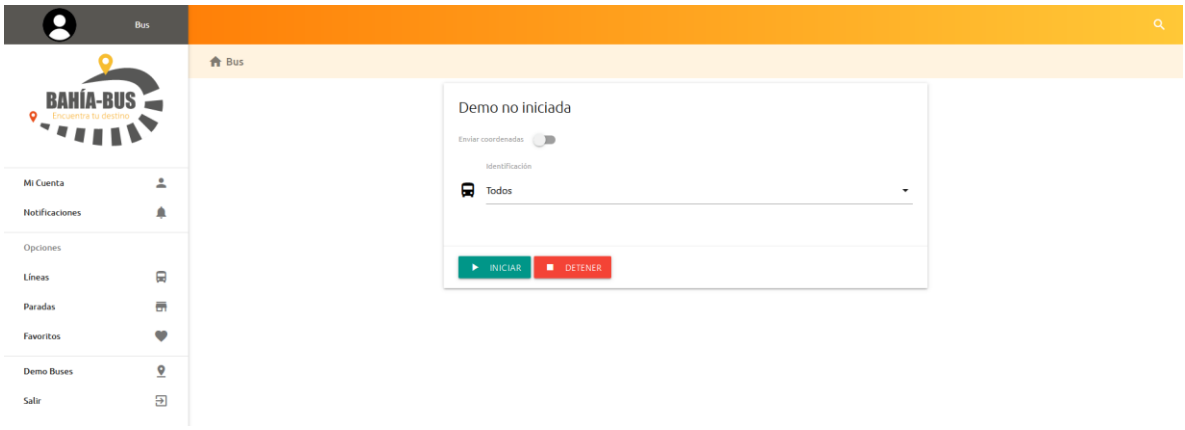


Figura 47. Zona de gestión para la demostración del movimiento de autobuses a través de *websockets*.

En la zona de gestión para la demo, hay que seleccionar todos los autobuses, si queremos ver a todos los autobuses moviéndose en todas las líneas, o uno en concreto si solo se quiere visualizar un autobús de una línea en particular. Si lo que se desea es ver todos los autobuses en movimiento, hay que marcar Todos en el desplegable y clicar en iniciar. Si solo se desea que un autobús realice el movimiento, hay que marcar cualquiera de los autobuses del desplegable y clicar iniciar.

Si lo que se desea es simular el movimiento del autobús real con nuestra propia ubicación, habrá que activar el *switch* de envío de coordenadas, pero, igualmente, es necesario seleccionar un autobús. Hay que tener en cuenta que la precisión del GPS del *smartphone* varía según la configuración establecida en el mismo y la calidad de los sensores.

Para visualizar el efecto de los autobuses moviéndose, ya en otra pestaña del navegador, se debe acceder a la aplicación con un usuario con rol user_rol, y visualizar el apartado líneas, o el de una línea en concreto.

7. Conclusiones y líneas de futuro

7.1. Conclusiones

7.1.1. Aprendizaje durante el trabajo

El trabajo ha supuesto una enseñanza desde dos puntos de vista: desde el punto de vista de la gestión y planificación del proyecto, ya que las diferentes estrategias ágiles seleccionadas para ejecutar cada iteración en cada tarea han supuesto una fuente de experimentación y conocimiento de los que carecía en la actualidad, ya que se aprende a priorizar tareas, organizarlas y a gestionar el tiempo, unos valores muy útiles en el desarrollo de futuros proyectos. Por otro lado, también ha sido enriquecedor desde el punto de vista técnico, ya que nunca había realizado un proyecto web empleando la tecnología MERN; de hecho, jamás había creado un proyecto en JavaScript, por lo que he precisado recibir formación complementaria al respecto.

En definitiva, me siento bastante satisfecho con las habilidades adquiridas hasta ahora y que tardaré tiempo en asimilar del todo, pero que una vez asentadas, me servirán de base para llegar a ser mucho mejor desarrollador.

7.1.2. Objetivos planteados

Los objetivos planteados desde el principio coinciden en su mayoría con los objetivos logrados en el desarrollo final. Si bien, todos y cada uno de ellos presentan un evidente margen de mejora. Aun así, la aplicación cumple los siguientes objetivos fundamentales:

- Es una herramienta digital que favorece la accesibilidad del autobús urbano, e integra las nuevas tecnologías en los procesos implicados para convertir a Sanlúcar de Barrameda en una ciudad inteligente.
- Agiliza la forma en la que los usuarios se relacionan con este medio y los mantiene informados en todo momento.
- Fomenta la movilidad a través del servicio de autobús urbano con el fin de apostar por una política de sostenibilidad, bajo coste y solidaridad.
- Aleja al usuario de las prácticas convencionales para obtener información actualizada del servicio de autobuses.
- Gestiona toda la red de transportes urbanos a través de un mapa virtual y la visualización de las líneas y paradas de autobuses mediante códigos de color.

7.1.3. Análisis crítico del seguimiento de la planificación y metodología a lo largo del proyecto:

La planificación, tratándose de un proyecto que utilizaba una metodología ágil, ha sido difícil de gestionar, ya que los requisitos podían cambiar en más de una ocasión, por lo que tareas como el desarrollo del Backend y FrontEnd ha sido muy iterativa y constante. Por ello, la metodología empleada ha sido la adecuada, al partir de unos requisitos casi desconocidos desde un principio. Tampoco resultaba adecuado emplear otro tipo de metodologías como las de tipo cascada para llevar a cabo el trabajo y que están planteadas para proyectos mucho más densos.

Los cambios que se han introducido han sido varios, sobre todo en lo que respecta al diseño de algunas de las vistas. Al final se decidió utilizar el mismo esquema de navegación tanto en búsquedas, como en líneas y paradas. Por otro lado, se ha eliminado la característica de guardar una parada en favoritos, ya que se llegó a la conclusión de que no tenía sentido en estos momentos. En el futuro, cuando se cuente con recursos que ayuden a implementar el tiempo de espera en una parada, sí lo tendrá.

El otro cambio importante es el del buscador, que, aunque está planteado en los objetivos, en un principio no se iba a tener en cuenta. Sin embargo, lo introduje como una herramienta bastante útil, ya que le otorga mucho valor al producto final.

7.2. Líneas de futuro

Las mejoras son las siguientes:

- Solucionar *bugs*. Al tratarse de una aplicación que depende mucho de una API externa, la asincronía puede generar efectos inesperados.
- La excesiva dependencia de una API externa como MapboxGL. Hay que tener en cuenta que el empleo de esta API no es gratuito, por lo que generar un plan de negocio con las limitaciones que implica la API es un riesgo que hay que saber gestionar en un futuro.
- Mejorar el algoritmo de búsqueda para que tenga en cuenta el cálculo del tiempo de espera de espera en parada. En el momento en el que se publica el proyecto, la aplicación no dispone de un servicio para calcular el tiempo que necesita el autobús para alcanzar una parada. Es un servicio que ha implantado la empresa del Grupo Avanza en sus autobuses de línea pero que no es accesible de manera pública.
- Añadir mejoras visuales y funcionales en el apartado personal de usuario. Tendría sentido poder realizar reseñas del servicio de autobús mediante la cuenta de usuario, o utilizar el perfil para contactar con la empresa de autobuses en caso de duda.
- Mejorar el registro de usuario con verificación de cuenta que, por falta de tiempo, no se ha podido implementar.

- Mejorar el aspecto de la zona de administración. Añadir filtros, paginación o scroll infinito son tareas que han quedado pendientes.
- Mejorar la comunicación entre la aplicación y el viajero a través de notificaciones de próximos eventos. Contar con un apartado de publicación de noticias relacionadas con el transporte en Sanlúcar de Barrameda y que tengan que ver con el servicio urbano de autobús sería un gran acierto.
- Optimizar la conexión de autobuses, clientes y servidor mediante *websockets* con el fin de mejorar el rendimiento y solucionar posibles *bugs*.
- Adaptar el apartado de notificaciones a ***websockets***. Actualmente, las notificaciones se envían por HTTP y la web necesita recargarse para visualizar las nuevas notificaciones.
- Integrar la aplicación con otros servicios de transporte como el servicio de taxis, trenes, barcos, etc.
- Refactorización de algunos apartados para hacerlos más escalables. Hacer mayor uso de los *custom hooks* ayuda a desacoplar la lógica de las vistas. Además de eso, sería necesario aprender a aplicar patrones específicos para programación funcional.

Bibliografía

- Ayuntamiento de Sanlúcar de Sanlúcar de Barrameda. (2020). *sanlucarpmus.es.* Recuperado el 03 de octubre de 2021, de http://sanlucarpmus.es/wp-content/uploads/2021/01/PMUS-Sanlucar-de-Barrameda_Fase-1_Memoria_red.pdf
- Climent, P. V. (s.f.). *Mapping GIS*. Recuperado el 23 de octubre de 2021, de https://mappinggis.com/2019/11/que-productos-y-servicios-ofrece-mapbox/
- Fowler, M. (2011). *Patterns of Enterprise Application Architecture.* Boston: Pearson Education, Inc.
- Groner, L. (2014). *Learning JavaScript Data Structures and Algorithms.*
- Kantor, I. (s.f.). *JavaScript.info*.
- Nielsen, J. (1993). *Usability Engineering.* London: Academic Press, Inc.
- React. (s.f.). *ReactJs.org*. Recuperado el 1 de enero de 2022, de https://es.reactjs.org/docs/hooks-state.html

Referencias en la web

- D.B.A.C. (2015). Redux - A predictable state container for JavaScript apps. | Redux. Recuperado 29 de noviembre de 2021, de https://redux.js.org/
- *Mongoose ODM v6.1.4*. (s. f.). Mongoose ODM. Recuperado 1 de enero de 2022, de https://mongoosejs.com/
- MongoDB. (s. f.). La base de datos líder del mercado para aplicaciones modernas. Recuperado 1 de enero de 2022, de https://www.mongodb.com/es
- *Fernando Herrera*. (s. f.). https://fernando-herrera.com/. Recuperado 1 de enero de 2022, de https://fernando-herrera.com/#/courses/React