



Instituto Tecnológico
de Buenos Aires

72.45 - PROYECTO FINAL DE INGENIERÍA INFORMÁTICA

MOVELO

Plataforma de cotización y reserva de
vehículos para transporte de carga

Autores

Nicolás Barrera (Legajo Nro. 57694)

Ezequiel Keimel (Legajo Nro. 58325)

Tutora

Hemilse Debrouvier

Fecha

Abril 2022

Resumen

El presente informe tiene por objetivo describir el proceso de desarrollo llevado adelante por el equipo de trabajo para la implementación de un sistema integrado de gestión de fletes, adecuado a las características y necesidades del emprendimiento *Movelo* (ex Fletes31), con el foco puesto en las decisiones tomadas, y las razones que motivan dichas decisiones, en el contexto de una visión integral no solo del producto, sino de la organización en general.

Índice

1. Introducción	4
2. Estado del arte	4
2.1. Servicios ofrecidos	5
2.2. Tamaño del sector	5
2.3. Principales competidores	6
2.4. Experiencia de usuario previa al proyecto	7
2.5. Estructura de la organización	8
3. Alcance del proyecto	9
3.1. Problemas detectados en la experiencia de usuario anterior al proyecto	9
3.2. Soluciones y objetivos propuestos	9
4. Producto desarrollado	10
4.1. Requerimientos	10
4.1.1. Requisitos funcionales	10
4.1.2. Requisitos no funcionales	11
4.2. Arquitectura	12
4.3. Backend	12
4.3.1. Aspectos técnicos	13
4.3.2. Patrones de diseño	13
4.3.3. Estructura del código	15
4.3.4. Manejo de secrets	25
4.4. Frontend	25
4.4.1. Aspectos técnicos	25
4.4.2. Patrones de diseño	26
4.4.3. Modelo de clases	28
4.4.4. Diseño de interfaces	31
4.4.5. Cotización de envíos	40
4.4.6. Seguimiento de envíos	42
4.4.7. Push notifications	44
4.4.8. JWT	44
4.4.9. Social Login	45
4.4.10. Chat	45
4.4.11. Funcionalidad específica por plataforma	46
4.5. Backoffice	47
4.5.1. Aspectos técnicos.	48
4.5.3. Funcionalidades implementadas	49

5. Métricas de evaluación de resultados	51
6. Manual de uso	52
6.1. Herramientas de desarrollo	52
6.1.1. Visual Studio	52
6.1.2. Android Studio	52
6.2. Pasos para realizar un deploy	52
6.3. Configuración del dominio movelo.com.ar	53
7. Desarrollos futuros	54
8. Referencias bibliográficas	56

1. Introducción

Movelo es una iniciativa conjunta entre el Gobierno de la Ciudad Autónoma de Buenos Aires y fleteros residentes en el Barrio Padre Mugica, que apunta a organizar a los transportistas individuales bajo una sola entidad para propiciar el desarrollo de su actividad mediante la centralización y optimización de los procesos administrativos, y del establecimiento de una imagen de marca bien cuidada. Comenzó oficialmente sus operaciones en el año 2020, transportando carga en la Ciudad y el Gran Buenos Aires, con un número de conductores y de paquetes transportados en continuo crecimiento.

Ahora, para ulterior profesionalización del servicio, *Movelo* y el equipo de desarrollo apuntaron a repensar íntegramente la infraestructura técnica y administrativa de la organización, con el foco puesto en la automatización de los procesos manuales con participación de los clientes para así ofrecer, de mínima, las facilidades que provee la competencia en cuanto a simplicidad de uso, seguridad y transparencia, eliminando así la desventaja competitiva actual que limita el crecimiento del proyecto.

2. Estado del arte

El proyecto *Movelo* se gestó en mayo de 2020, cuando se atravesaban las primeras etapas de la pandemia de Covid 19, en un contexto en el cual se evidenciaban problemas de logística dentro del Barrio Mugica y limitada movilidad en toda la Ciudad Autónoma de Buenos Aires. A partir de esta problemática, se comenzó a planificar de forma conjunta con fleteros del Barrio Mugica una posible alternativa que permitiera impulsar un desarrollo económico y social sostenible, a la vez que se resolvieran los problemas de logística que se presentaban.

Luego del diseño de un primer prototipo de la página web, el GCBA se contactó con el Instituto Tecnológico de Buenos Aires para buscar llevar adelante esa propuesta, y gracias al soporte brindado por la universidad, se logró implementar la plataforma que estuvo disponible durante ese año bajo el dominio *fletes31.com*. Esta primera versión brindaba la posibilidad de solicitar un envío, pero no de cotizarlo, y toda la coordinación del pedido se realizaba mediante WhatsApp. Una vez que el cliente se comunicaba por este medio, la cotización y asignación de un conductor se realizaba a través de un operador. Esto implicaba que el servicio resultara poco eficiente y muy difícil de escalar, dado que toda la gestión se daba en forma manual.

Al momento de comenzar el desarrollo del presente proyecto (en abril de 2021), eran 16 vecinos del Barrio quienes integraban *Movelo*, los cuales se dividían en 4 categorías en función de sus vehículos: mudanzas (camiones), camionetas, mini-fletes y mensajería (motos). Además, en el período que comprende de septiembre de 2020 a abril de 2021, se recaudó un total de 246 mil pesos.

La principal meta del proyecto es lograr su autonomía para fines de 2022. Esto quiere decir que se busca lograr que *Movelo* funcione independientemente del apoyo de la Secretaría de Integración Social y Urbana, pudiendo solamente estar involucrados en un rol de soporte a los vecinos.

2.1. Servicios ofrecidos

Movelo es una plataforma que, mediante una web intuitiva y simple de utilizar, permite cotizar y solicitar los siguientes servicios:

- **Mudanzas:** Servicio de mudanzas integral (puerta a puerta, con posibilidad de embalaje y ayudantes). Ideal para mudanzas grandes.
- **Camionetas:** Servicio de mudanzas pequeñas. Traslado de muebles, electrodomésticos u otros artefactos pesados.
- **Mini Fletes:** Servicio de traslados para objetos medianos y más livianos, que entren en camionetas tipo Partner o en autos.
- **Mensajería:** Servicio de mensajería en moto para trasladar objetos livianos.

2.2. Tamaño del sector

En la Argentina, el sector logístico se encuentra en constante crecimiento y más aún con las consecuencias derivadas de la crisis provocada por la pandemia del Covid-19, puntualmente de beneficio para el desarrollo de Movelo. Es una realidad que no todas las empresas o personas pueden entrar al barrio, por lo cual surge la necesidad de los vecinos del barrio de salir hacia la ciudad con sus necesidades para así encontrar una solución.

La logística urbana y el comercio electrónico han resultado vitales para la continuidad operativa de las actividades urbanas, al favorecer la distribución de alimentos y bienes necesarios para los largos períodos de confinamiento a los que ha estado sometida buena parte de la población. De este modo, los fleteros del barrio unidos en Movelo, podrán ofrecer una solución de primera mano ante este problema. Ellos cuentan con el conocimiento y experiencia necesarios, siendo vecinos del barrio y conociendo el rubro perfectamente, ya que muchos de ellos trabajan desde hace muchos años en esta profesión.

Puntualmente, notamos que el producto de mini-fletes demostrará un crecimiento más importante, ya que será el que marque la diferencia con las apps de delivery, pudiendo enviar paquetes más grandes, en más cantidad y así minimizar los costos.

En particular, tomando como referencia un análisis de mercado realizado por el Gobierno de la Ciudad de Buenos Aires, se resumen los siguientes puntos:

- **Mudanzas:** En mayo de 2020, se registraron en la ciudad 17.000 contratos de alquileres vencidos. Además, el 61% de los argentinos encuestados, afirmó que de cara al 2021 buscaría mudarse de hogar[1]. Además, de acuerdo a información disponible en La Nación, la cantidad de usuarios de algunas de las apps que están presentes en el mercado que alquilan su flota de autos o camionetas para que los usuarios realicen sus mudanzas, es:
 - **Awto:** 5.500 usuarios activos.
 - **MyKeego:** 10.000 descargas y 2.600 usuarios activos.
 - **Toyota Mobility Services:** 8.000 usuarios activos.

- **Fletes (camionetas y mini fletes):** Se estima que a raíz de la pandemia, la cantidad de personas que realiza sus compras online aumentó en un 30%. Uber cuenta hasta diciembre de 2020 con más de 2 millones de usuarios activos por mes[2]. En la costa argentina, en enero de 2021, se registra un crecimiento del 300% en la cantidad de viajes solicitados.
- **Mensajería:** Hasta octubre del año pasado, Uber Flash registró una demanda de 247.000 viajes. Cabify implementó su plataforma de envíos, ante la merma en el transporte de pasajeros, y así representan el 20% de sus operaciones actuales hasta mayo de 2020[3]. Rappi además creció en descargas durante los meses que van transcurriendo de la pandemia, y además se multiplicaron por 4 las entregas. Y se estima que las órdenes mensuales pasaron de 2,5 a 3,5 órdenes por persona[4].

Por otra parte, al menos 60.000 personas trabajan al día de hoy para Rappi y Pedidos Ya. Hacia marzo del corriente año, Rappi superó la barrera de los 20.000 repartidores, y además superó las 4 millones de descargas. Por su parte, Pedidos Ya cuenta con unas 11 millones de descargas, y afirma que cerrarán el 2020 con un crecimiento interanual del 50% en pedidos. De este modo, Pedidos Ya se consolida como líder en el mercado argentino. Antes de la pandemia, el usuario de Pedidos Ya en promedio realizaba 2,8 pedidos por mes, luego de la pandemia el promedio incrementó a 3,5 pedidos[5].

2.3. Principales competidores

Se detallan a continuación los principales competidores por sector, información que también se obtuvo como resultado del estudio de mercado realizado por el GCBA:

- **Mudanzas:**

Forma de contratación: Se completan los datos a través de la app y se obtiene un presupuesto en el momento.

- Fletalo[6]
- Toyota mobility service[7]
- Awto[8]
- MyKeego[9]

Forma de contratación: Se completan los datos a través de la web y se recibe una respuesta a las 24hs.

- Verga Hnos[10]

- **Camionetas:** También resulta una competencia dentro de esta sección por ofrecer traslados en camionetas.

- Fletalo

- **Mini Fletes:** También resulta una competencia dentro de esta sección por ofrecer traslados en camionetas pequeñas.
 - Fletalo
- **Mensajería:** Forma de contratación: Se completan los datos a través de la app y se obtiene un presupuesto en el momento.
 - Pedidos Ya
 - Rappi
 - Cabify envíos

2.4. Experiencia de usuario previa al proyecto

Originalmente, se contaba con un sitio web como punto de acceso al servicio, y con un administrador humano que coordinaba los viajes. A continuación se detalla, paso a paso, el proceso complejo bajo ese sistema:

1. El potencial cliente ingresa datos básicos del pedido en el sitio web, como las direcciones de origen y destino, y la descripción del producto a trasladar.
2. El sitio compone un mensaje de WhatsApp dirigido al administrativo central que contiene los datos ingresados, y utiliza la URI de WhatsApp para proponer al cliente enviar el mensaje.
3. El administrativo recibe el mensaje, extrae del mismo la información relevante para la cotización del viaje, y utiliza una hoja de cálculo de Excel precargada con fórmulas que calculan un valor.
4. El administrativo transmite por WhatsApp el costo del viaje.
5. El usuario recibe el mensaje con el costo, y si decide aceptar, se lo comunica, siempre por WhatsApp, al administrativo.
6. El administrativo contacta manualmente con conductores disponibles que cumplan con los requisitos para concretar el viaje.
7. Uno de los potenciales conductores decide aceptar y lo informa al administrativo.
8. El administrativo comparte con el cliente la información del conductor para permitirles comunicarse.
9. Conductor y cliente coordinan detalles del envío por el medio de su preferencia.
10. El conductor inicia el viaje.
11. Conductor y cliente se informan sobre el estado del viaje mediante algún medio de comunicación a su gusto.

12. El conductor finaliza el viaje.
13. El conductor informa por WhatsApp a la central del final del viaje.
14. El cliente tiene la posibilidad de hacer llegar quejas o comentarios a través de WhatsApp al administrativo.
15. El administrativo tiene la posibilidad de reaccionar frente a una queja que considera válida, mediante una amonestación o bien sancionando al conductor.

2.5. Estructura de la organización

La estructura de la organización durante el desarrollo del presente proyecto se puede dividir en:

- **Área comercial:** Compuesta por integrantes de la Secretaría de Integración Social y Urbana, con el objetivo de concretar iniciativas comerciales y de inclusión financiera.
- **Mudanzas:** Integrada por vecinos del Barrio Mugica (para el mantenimiento operativo de la plataforma), y por los autores de este informe (para llevar a cabo el desarrollo tecnológico).

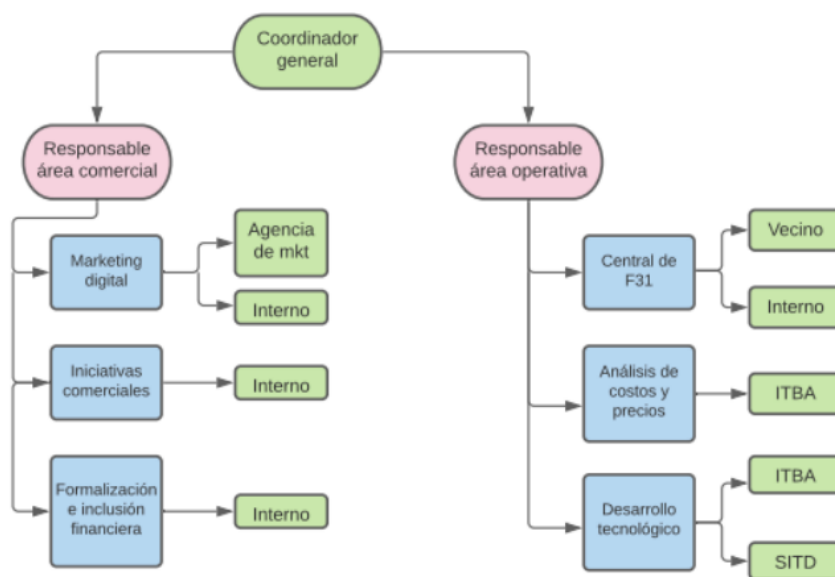


Figura 1: Esquema organizacional de MoveLO.

3. Alcance del proyecto

3.1. Problemas detectados en la experiencia de usuario anterior al proyecto

Para la detección de problemas a encarar en el proyecto aquí discutido, se contemplaron las visiones del equipo de Movelo, el análisis del equipo de desarrollo, y las críticas y comentarios de un grupo de 4 personas que fueron expuestas a la solución previa. Las siguientes fueron las conclusiones alcanzadas:

- El proceso manual preexistente, en el que se requiere de la comunicación explícita mediante mensajes entre el cliente y administradores y conductores, implica tiempos muertos entre respuestas, retrasando el proceso y atentando contra la paciencia de los clientes, en un contexto en el que la competencia ofrece procesos automatizados y, en consecuencia, mucho más inmediatos.
- El uso de WhatsApp como medio de comunicación, y la comunicación textual en general, pueden atentar contra la óptica de seriedad del servicio, además de ser inherentemente propensa a errores de comunicación.
- El proceso manual preexistente se inicia desde el sitio web, pero requiere después abandonarlo en pos de un medio de comunicación (WhatsApp, en este caso) no controlado por Movelo. Esto transmite la idea de que Movelo no controla la totalidad del proceso, lo que puede generar desconfianza frente a competidores que sí gestionan todo íntegramente desde su plataforma.
- La interfaz del sitio web original resulta poco vistosa y confusa para el usuario, según varios usuarios entrevistados a los que se les pidió evaluarla.
- Se requiere de un administrativo siempre alerta para que el servicio funcione.
- El sistema no escala: en caso de crecimiento de la demanda, el administrativo podría verse sobrecargado. La posibilidad de contratar más existe, pero el uso de WhatsApp como medio de contacto hace que se deba optar por tener varios administrativos trabajando sobre el mismo WhatsApp, o utilizar una cuenta de empresa para distribuir los mensajes entrantes, lo cual no solo implica un costo y capacitación, sino que aún deben mantenerse sincronizados en el estado de los conductores disponibles por medio de alguna herramienta como Excel.

3.2. Soluciones y objetivos propuestos

- Implementar la nueva imagen de marca produciendo un sitio atractivo visualmente y sencillo en su uso, que transmita confianza y seriedad.

- Implementar un cotizador automático basado en el formulario existente, que pueda asignar un precio a un viaje de manera automática e inmediata.
- Ofrecer una experiencia de pedido que reduzca al mínimo la intervención humana requerida, cerrando la brecha con los competidores existentes. Los únicos tiempos muertos deben ser la asignación de un conductor al pedido (indispensable debido a que son ellos los que deben aceptar el mismo), y las esperas hasta la hora de inicio del pedido, y posteriormente la entrega final.
- Ofrecer un canal de comunicación por texto integrado en la plataforma, de forma tal que en ningún momento deba el cliente operar por fuera de la plataforma. Esto además permitirá la auditoría de dicha comunicación para resolución de conflictos.
- Promover la transparencia del servicio en todo momento, en especial en lo que respecta al estado actual del pedido y la geolocalización del conductor en tiempo real. Permitir también acceder al historial de uso en todo momento.
- Tener llegada a la mayor cantidad de usuarios posibles: el servicio debe ofrecerse tanto a través de una WebApp como de aplicaciones disponibles, de mínima, en las 2 plataformas móviles más populares, Android e iOS.
- Ofrecer a los administradores herramientas sencillas para la gestión de clientes, conductores, y administradores, incluyendo listado con filtro, alta, baja y modificación.
- La aplicación tendrá un modo cliente y un modo conductor. Ambos modos se presentarán en un mismo ejecutable, y será el frontend el que se adapte dependiendo del tipo de usuario que inicie sesión en la aplicación.

4. Producto desarrollado

La solución desarrollada consta de 3 componentes principales:

- **Aplicativo de clientes y conductores:** disponible para la Web, Android e iOS, permite participar del flujo de pedido de principio a fin.
- **Backoffice:** aplicación web que permite a los usuarios administradores gestionar los usuarios existentes.
- **Backend:** REST API que sirve a los dos frontends, procesando, almacenando y recuperando información.

4.1. Requerimientos

4.1.1. Requisitos funcionales

- La plataforma debe permitir a los clientes cotizar sus envíos instantáneamente, valorando las necesidades del cliente en términos de sus costos asociados para el conductor, proponiendo un precio competitivo y económicamente viable.
- Debe notificar de forma instantánea a los conductores disponibles, desplegando la información del envío y permitiéndole aceptarlo.
- Debe garantizar al cliente la posibilidad de un seguimiento en tiempo real del estado de su envío.
- Debe proveer un canal de comunicación privado y en tiempo real entre el cliente y el conductor.
- Debe ofrecer una interfaz de administración lo bastante poderosa como para permitir la delegación total de la gestión de la plataforma a los usuarios encargados.

4.1.2. Requisitos no funcionales

- Debe ofrecerse una interfaz atractiva y sencilla de utilizar, que promueva el uso del producto.
- Debe soportarse el servicio en el mayor número de plataformas para ampliar la potencial base de usuario. De mínima, deben soportarse los dos sistemas operativos móviles más importantes, iOS y Android, así como la web.
- Debe ser lo bastante ligera en términos de espacio de almacenamiento y consumo de recursos para funcionar correctamente en una amplia gama de dispositivos.
- El sistema debe proporcionar mensajes de error que sean informativos y orientados al usuario final.
- El sistema debe funcionar de manera continua e ininterrumpida, de forma tal que se ajuste graciosamente a la oscilante demanda.

4.2. Arquitectura

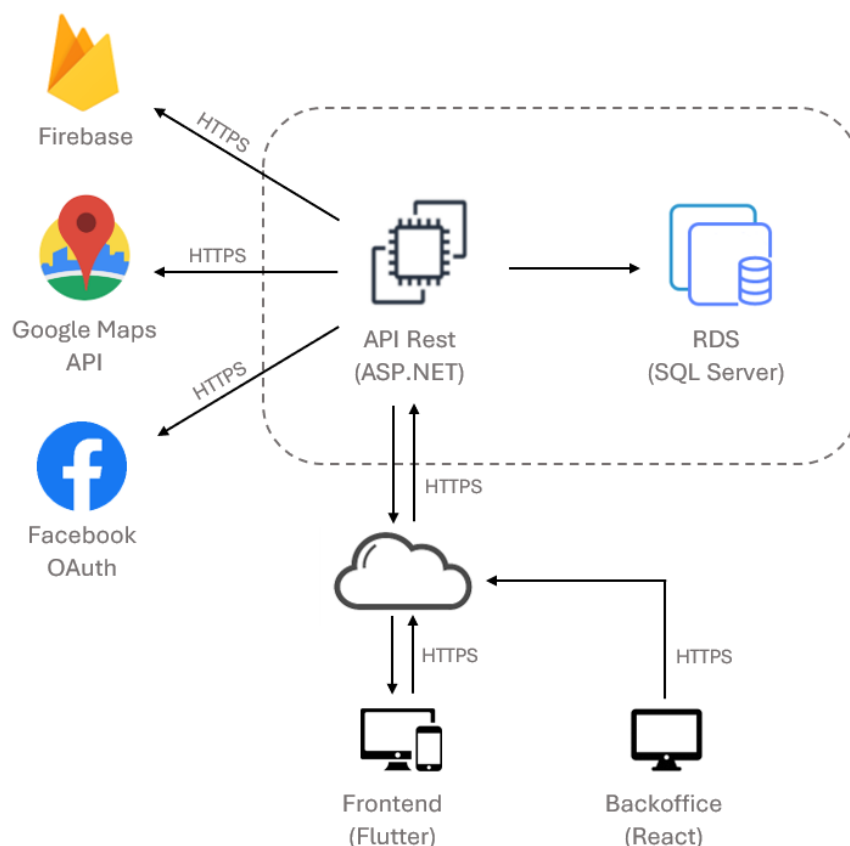


Figura 2: Diagrama de arquitectura de la plataforma. Fuente: elaboración propia.

Actualmente, el sitio web se encuentra hosteado en el servicio SmarterASP.NET, bajo el dominio *movelo.com.ar*. A continuación se desarrollan en detalle los distintos componentes de la arquitectura.

4.3. Backend

El backend de la plataforma tiene la responsabilidad de ofrecer una interfaz de comunicación consumible tanto por el frontend de usuarios y conductores (aplicación Flutter) como por el backoffice (aplicación web React), por lo que definir dicha interfaz y su protocolo de comunicación asociado resultó la elección primordial.

Debe además, poder comunicarse con servicios externos, incluyendo al menos un servicio de mapas y uno de envío de notificaciones móviles.

También debe gestionar el almacenamiento de largo plazo y consulta de información relevante.

El crecimiento proyectado a partir de las mejoras que trae aparejado este proyecto elevan la extensibilidad como requerimiento clave, puesto que tanto desde el equipo de desarrollo como desde la organización misma se prevé desde ya la necesidad futura de implementar ulteriori funcionalidades, así como ajustar las existentes a las necesidades del producto.

4.3.1. Aspectos técnicos

Para el desarrollo del backend de la plataforma se optó por el framework web ASP.NET 5[11], desarrollado por Microsoft como parte de su suite .NET. ASP.NET 5 permite utilizar diversos lenguajes de programación para el desarrollo de sitios web o de web APIs modernas.

En lo que hace puntualmente al desarrollo de web APIs siguiendo el estándar REST, ofrece funcionalidades útiles como validación y serialización/deserialización de clases modelo, documentación automática de endpoints, autenticación y autorización personalizable y flexible para endpoints y controladores, y un avanzado sistema de inyección de dependencias que facilita enormemente la integración del extenso catálogo de librerías gratuitas disponibles a través del gestor de paquetes NuGet.

El lenguaje de programación más popular, y en consecuencia el mejor documentado, en el entorno .NET es C#, conocido ya por los integrantes del equipo, lo que significó una ventaja adicional a favor de ASP.NET.

Varias otras alternativas fueron contempladas, aunque posteriormente descartadas. Spring resultó un fuerte contendiente debido a la experiencia del equipo en el uso del mismo para el desarrollo de APIs en el pasado, pero una comparativa minuciosa de las facilidades ofrecidas por el framework en cuestión y ASP.NET reveló que el uso del primero hubiese requerido de librerías y, en consecuencia, configuraciones adicionales que hubiesen tornado el desarrollo más tedioso y propenso a errores. Express también fue considerado, pero la comparativa con ASP.NET no arrojó ningún potencial beneficio que compense el desconocimiento del equipo en su uso.

4.3.2. Patrones de diseño

A fin de evitar rispideces en el desarrollo, se optó por priorizar el uso de patrones recomendados por la documentación oficial de ASP.NET, contemplando los argumentos a favor y en contra esgrimidos por integrantes de la comunidad de desarrolladores para tomar una decisión.

Cabe destacar que el propio ASP.NET ya se encarga de facilitar la implementación de Inversion of Control y Dependency Injection, y no utilizarlos sería ir en contra de los propios principios del framework, por lo que no los consideramos patrones directamente elegidos por el equipo y no se ahonda en ellos ulteriormente.

4.3.2.1. Patrón N-Tier

Divide el proyecto en capas de presentación, lógica y acceso a datos separadas lógicamente y físicamente. Se define una jerarquía en la cual una determinada capa solo depende y puede comunicarse con la inmediatamente subsiguiente, desacoplando las dependencias e independizando los distintos componentes.

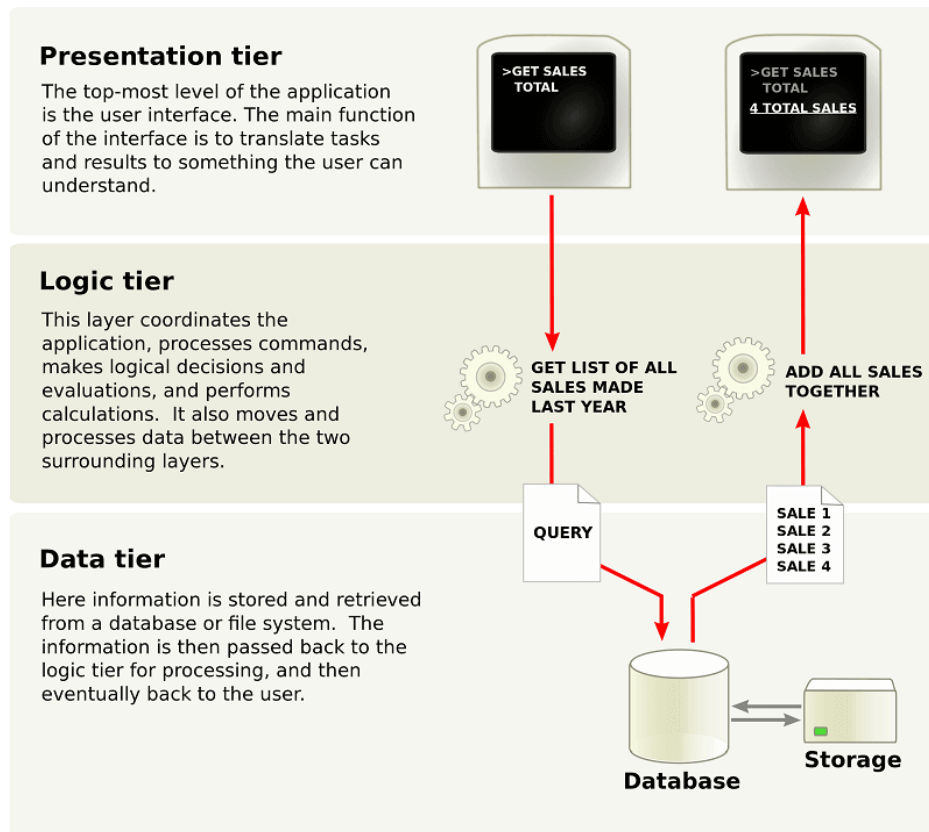


Figura 3: Esquema representativo del patrón *N-Tier*. Fuente: [12].

La aplicación del patrón resultó también enormemente beneficiosa para el trabajo simultáneo de los integrantes del equipo sobre el código del backend: cada capa ofrece a la inmediatamente superior una interfaz, por lo que se evita trabajar contra implementaciones concretas, permitiendo la modificación de módulos enteros de una de las capas sin alterar de manera alguna las demás. Además, este acercamiento facilitó la detección de errores aislando los distintos estratos y permitiendo su testeo individual.

En secciones subsiguientes se presentan en detalle las distintas capas, así como las responsabilidades que competen a cada una.

4.3.2.2. Patrón Repository

Permite abstraer las operaciones de acceso a datos de las implementaciones concretas que permiten dicho acceso. De esta forma, se desacopla el mecanismo puntual de acceso a datos (en este caso, el ORM Entity Framework), y los particulares de la base de datos elegida (la instancia específica de SQL Server, a los efectos del proyecto), permitiendo la modificación de la configuración e incluso sustitución, total o parcial, del mismo.

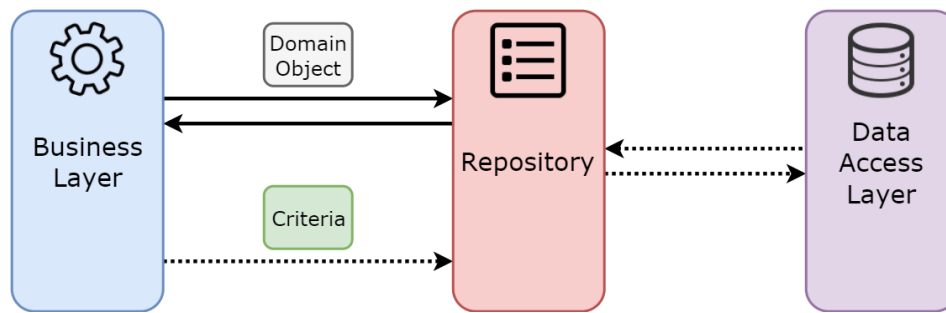


Figura 4: Esquema representativo del patrón *Repository*. Fuente: [13].

Además, facilita implementar las operaciones comunes a todas las entidades (por ejemplo, acceso por clave primaria, conteo de registros, o paginado) una sola vez para cada una de las entidades, por lo que cada una implementará únicamente los comportamientos específicos de la misma, reduciendo así casos de código duplicado.

4.3.2.3. Unit of Work

Asocia operaciones de acceso a datos directamente a las entidades que modelan los mismos, además de abstraer el concepto de transacción a un objeto C# tradicional sobre el que se opera posteriormente.

Funciona como punto de acceso único a los distintos Repositorios, y a la operación de guardado de cambios efectivo en el mecanismo de persistencia asociado a los mismos.

4.3.3. Estructura del código

Siguiendo los lineamientos definidos por el patrón N-Tier, la solución se dividió en 4 proyectos distintos. Cada uno de los proyectos incluye su propio árbol de dependencias, y solo se expone a la capa inmediatamente superior por medio de interfaces, a excepción de la más superficial, que debe estar directamente expuesta a los clientes.

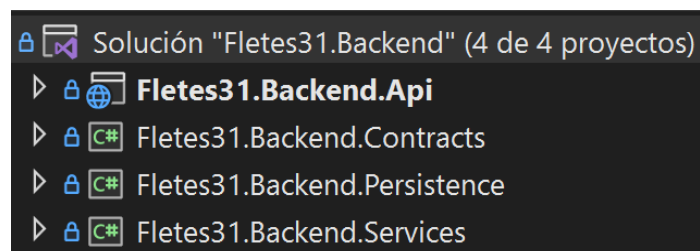


Figura 5: Los cuatro proyectos que componen el backend.

Estructura y contenido de cada una se detalla a continuación.

4.3.3.1. Contratos (Proyecto *Contracts*)

Único proyecto que no constituye una capa per se, sino que se encarga de almacenar las interfaces y modelos que luego los demás proyectos implementarán. Es, además, el único

proyecto que será dependencia de todos los demás, publicando dichas interfaces a través del sistema de inyección de dependencias de ASP.NET.

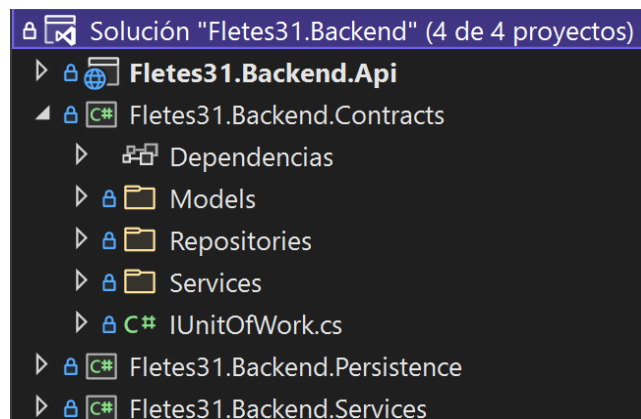


Figura 6: Contenidos del proyecto *Contracts*.

Contiene una carpeta destinada a almacenar los modelos internos, y otras dos que agrupan las interfaces de los Servicios y de los distintos Repositorios (más información sobre estos últimos dos en sus respectivas secciones), incluyendo la interfaz genérica *Repository* que define los métodos de acceso a datos genéricos comunes a todos los repositorios.

Finalmente, el proyecto contiene también la interfaz correspondiente a la Unidad de Trabajo del patrón *Unit of Work* anteriormente descrito.

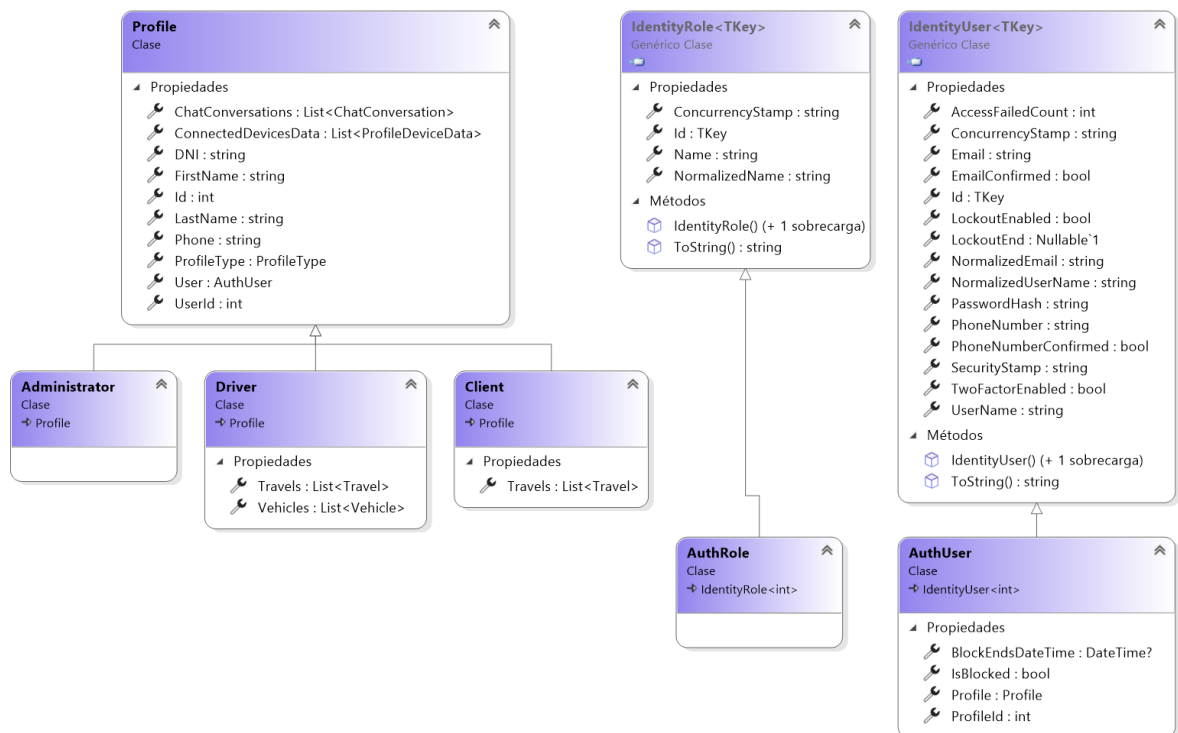


Figura 7: Modelos de tipos de usuario y de autenticación.

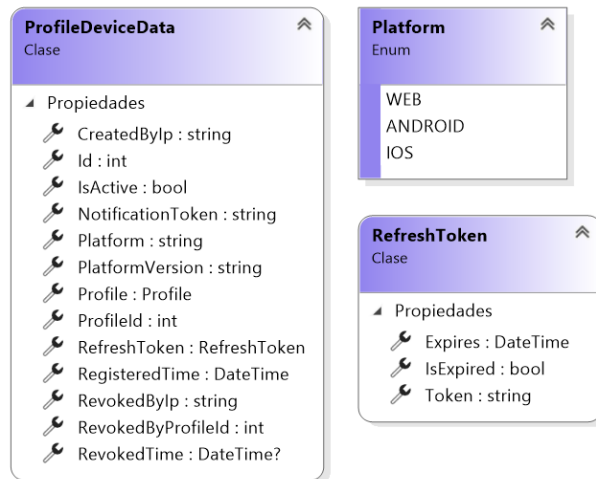


Figura 8: Modelos de datos de sesión e información del dispositivo del usuario.

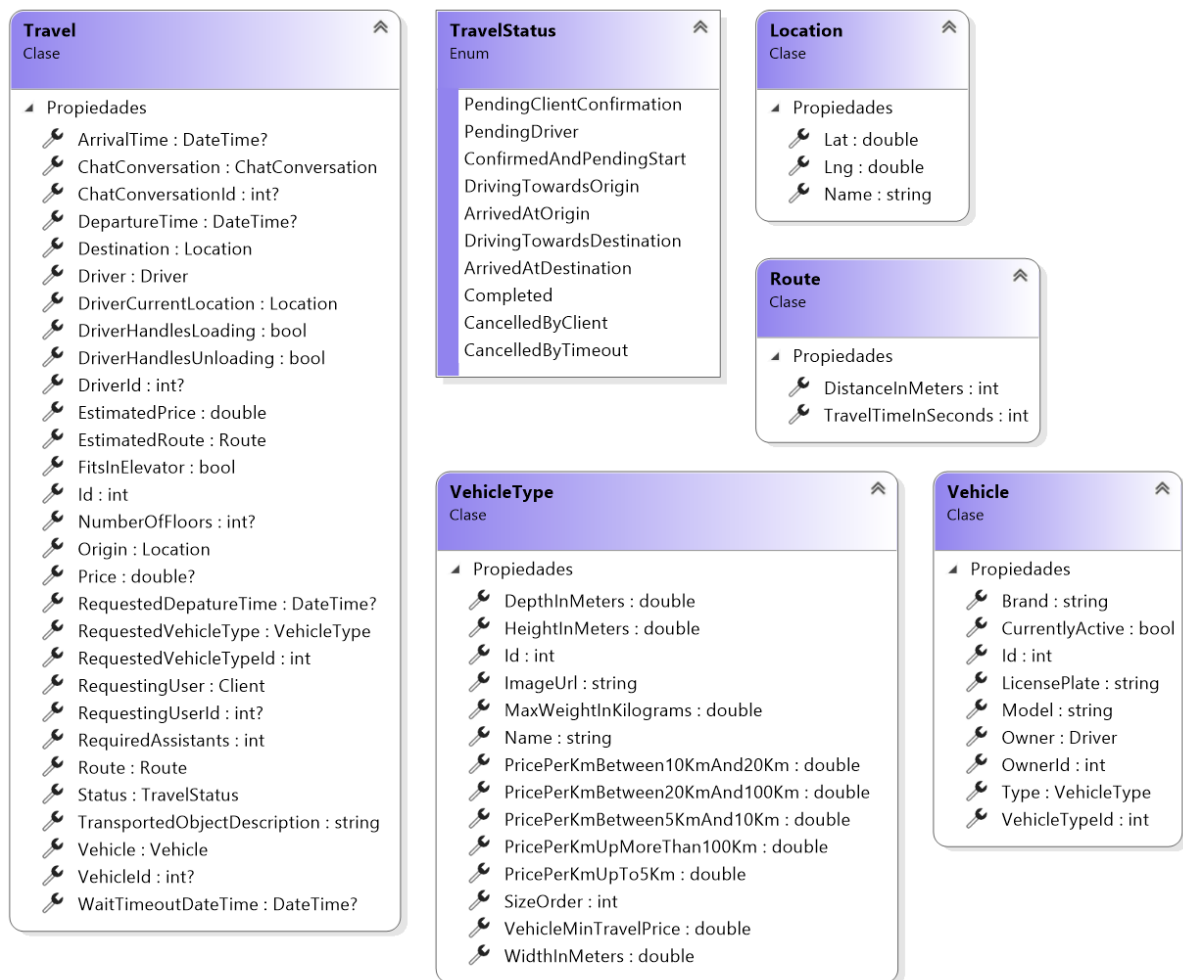


Figura 9: Modelos de viaje y asociados.

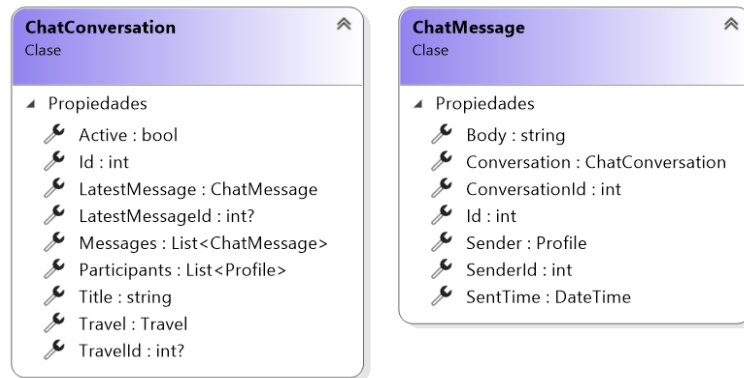


Figura 10: Modelos de conversación de chat y mensajes.

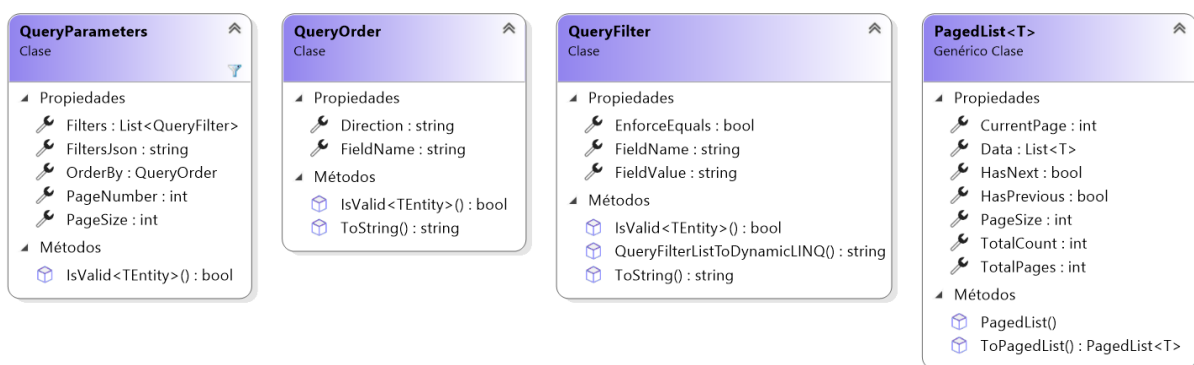


Figura 11: Modelos de parámetros que soporta la API para consultas personalizadas sobre la información, así como de la clase genérica de lista paginada que los métodos de consulta devuelven.

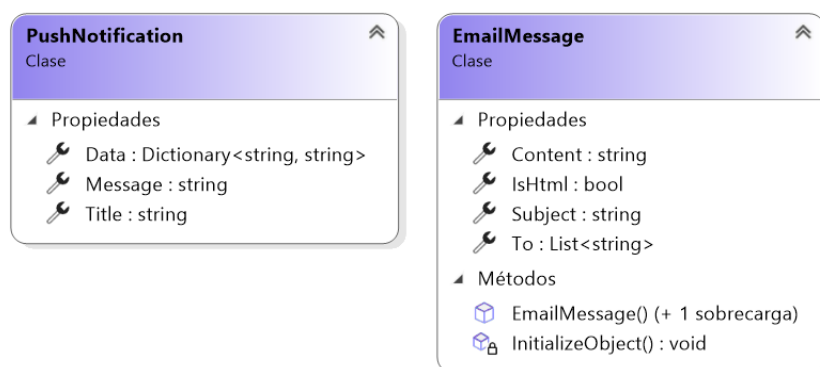


Figura 12: Modelos internos de un mensaje de correo electrónico y de una notificación push. Se mapean posteriormente a los modelos de los servicios que envían los correos y notificaciones en cuestión.

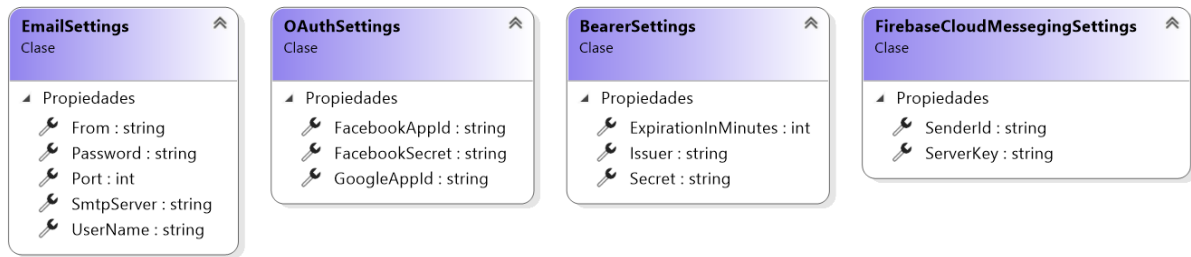


Figura 13: Clases de parámetros de configuración para las distintas dependencias.

4.3.3.2. Persistencia de datos (Proyecto *Persistence*)

Contiene todas las clases destinadas a hacerse cargo de la escritura y lectura de datos en el medio de persistencia elegido.

Para ejecutar las operaciones específicas sobre el medio de persistencia, se optó por utilizar el ORM Entity Framework, incluido con ASP.NET y recomendado unánimemente por la documentación del framework y por la comunidad. El ORM se encargará de mantener actualizado el esquema de base de datos con los modelos mediante un avanzado sistema de Migraciones. Cabe destacar que Entity Framework está desarrollado con las características de orientación a objetos de C# en mente, por lo que funcionalidades del lenguaje como herencia o uso de colecciones se encuentran totalmente soportadas y fueron aprovechadas a lo largo y ancho del proyecto; un ejemplo de esto es la utilización de herencia para definir los distintos tipos de usuario extendiendo una clase común que define las características compartidas por todos.

Actualmente, se utiliza el motor de base de datos Microsoft SQL Server para almacenar información. El mismo fue elegido por sobre otras alternativas como PostgreSQL debido a su facilidad de uso, gratuidad, alto grado de integración con el framework y con el entorno de desarrollo Visual Studio, y omnipresencia en la oferta de todas las soluciones de hosting que soportan ASP.NET de manera específica. No se contemplaron opciones no relacionales debido a la alta normalización de la información a almacenarse. Cabe destacar, en todo caso, que la abstracción ofrecida por Entity Framework por sobre el medio de almacenamiento, así como su compatibilidad con decenas de distintos motores, relacionales y no relacionales, simplificaría enormemente cualquier futuro esfuerzo por migrar a una alternativa a SQL Server.

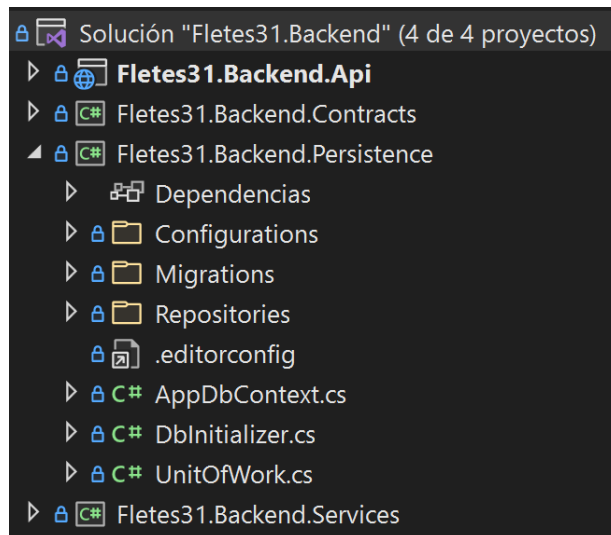


Figura 14: Contenidos del proyecto *Persistence*.

En cuanto a su contenido, la clase *DbInitializer* contiene la lógica de inicialización de la base de datos en caso de que la misma no pueda ser encontrada y deba ser creada desde cero, incluyendo el cargado de datos iniciales (como los distintos tipo de vehículo estándar).

La carpeta *Configurations* contiene clases de configuración de modelos de Entity Framework, que permiten aplicar ajustes específicos a cada una; por ejemplo, la auto inclusión de determinadas entidades relacionadas (como la lista de vehículos de cada conductor) en cada consulta a la misma, para que no sea necesario cargarlas por separado.

AppDbContext es la clase de configuración del ORM Entity Framework. Se encarga de establecer cuáles serán los modelos sobre los que se podrán realizar consultas para exponer los métodos correspondientes, y de aplicar las clases de configuración de modelo mencionadas anteriormente.

Al momento de la redacción de este informe, los modelos configurados en *AppDbContext* son *VehicleTypes*, *Travels*, *Profiles* (genérico), *Drivers*, *Clients*, *Administrators*, *Vehicles*, *ChatConversations* y *ChatMessages*. Cabe aclarar que otros modelos referenciados por alguno de los anteriores también tendrán su correspondiente tabla en la base de datos, pero no podrán ser directamente consultados sin pasar por su entidad padre.

La carpeta *Migrations* contiene el historial completo de clases de migración, que corresponden a cada actualización de los modelos, y son generados a petición del programador, luego de modificarse los mismos. Cada una de las migraciones incluye una marca de tiempo, comentarios explicando el motivo de ser de la misma, y código para aplicar la modificación o deshacerla. Las migraciones se generan, eliminan, aplican o deshacen por medio de herramientas incluidas en el paquete de Entity Framework instalado mediante NuGet. Cabe destacar que *DbInitializer* aplica todas las migraciones, de la primera a la última, cuando crea la base de datos, garantizando siempre la sincronía modelo-esquema de SQL Server.

La carpeta *Repositories* contiene las implementaciones de la interfaz genérica *Repository*, y de sus interfaces herederas, los repositorios específicos de cada uno de los

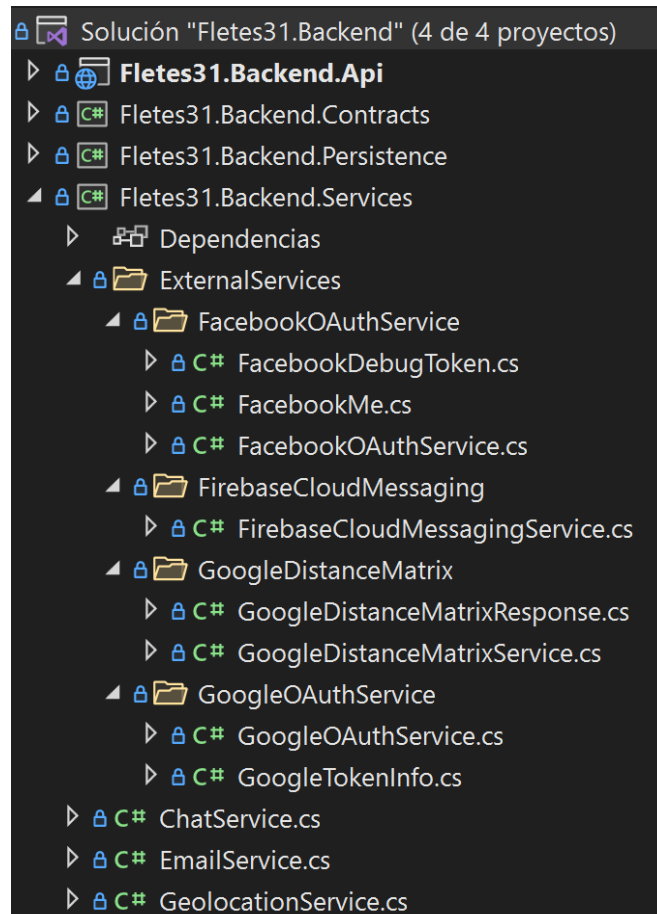
modelos sobre los que se pueden ejecutar consultas directamente, todas ellas definidas en el proyecto *Contracts*.

Finalmente, la clase *UnitOfWorkWork* implementa la interfaz homónima del proyecto *Contracts*, inicializa los repositorios a petición, e implementa los métodos de guardado final de los cambios realizados durante la transacción, abstrayendo los conceptos de transacción, y desacoplando al propio Entity Framework como dependencia, permitiendo su reemplazo sin requerir modificaciones en las capas superiores.

4.3.3.3. Servicios internos y externos (Proyecto *Services*)

Contiene las implementaciones las distintas interfaces de servicios definidos en el proyecto *Contracts*; es decir, las operaciones que se pueden realizar sobre el estado del sistema (registro de un usuario, finalización de un viaje, actualización de la posición de un conductor, entre tantos otros), además de código relativo a interactuar con servicios externos (por ejemplo, el servicio de notificaciones de Firebase), y servicios auxiliares (como todo lo relativo al manejo de tokens JWT).

Varios de los servicios contienen únicamente código relativo a consultas, agregado o modificación de instancias del modelo: por ejemplo, *VehicleTypeService* solo existe para facilitar la obtención del listado de tipos de vehículo y para permitir modificar sus parámetros. Debido a esa simplicidad, este informe solo se detendrá en aquellos servicios que, el equipo entiende, requieren de alguna explicación no trivial.



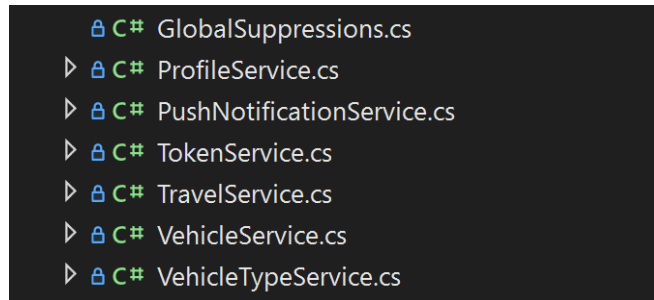


Figura 15: Contenidos del proyecto *Services*.

La carpeta *ExternalServices* contiene una subcarpeta por cada servicio externo consumido, que contiene el código específico de interacción con dicho servicio así como cualquier modelo o clase auxiliar necesario para facilitar dicha comunicación.

Al momento de la redacción de este informe, el backend aquí detallado interactúa con cuatro servicios externos. *Google Distance Matrix* se utiliza para cálculo de una ruta entre dos posiciones geográficas, en un determinado momento del tiempo, y con la posibilidad de evitar expresamente peajes, y *Firebase Cloud Messaging*, para el envío de notificaciones, ya sea globales, o dirigidas a una lista de tokens de dispositivo específica. Cabe destacar que, en ambos casos, la decisión de optar por estos servicios puntuales frente a alternativas se debió a su gratuidad en el caso de uso contemplado (*Google Distance Matrix* tiene un costo, pero solo a partir de un número de consultas que habría implicado un crecimiento tal en el uso de la plataforma que su solvento no implicaría un desafío económico), y su abrumador porcentaje de adopción entre la comunidad, lo que redundó en más y mejor documentación. Además, existen dos adicionales, *FacebookOAuthService* y *GoogleOAuthService*, que lidian con la validación de sus respectivos tokens OAuth para la implementación del inicio de sesión mediante redes sociales.

GeolocationService y *PushNotificationsService* son los servicios que ofrecen los métodos genéricos que, a su vez, consumen los servicios externos. Existen con el fin de desacoplar el código relativo a los mismos; es decir, su propósito es permitir un futuro reemplazo de, por ejemplo, *Google Distance Matrix* por alguna alternativa como *OpenStreetMap*, sin necesitar modificar código por fuera de los servicios mismos.

ChatService es el servicio que provee las operaciones de creación y consulta de conversaciones de chat y mensajes, que se almacenan en la base de datos. Incluye interacción con el servicio de notificaciones para alertar a los destinatarios de mensajes cuando resulta apropiado.

ProfileService implementa las operaciones estándar de consulta, alta y baja de distintos tipos de usuario, pero también la generación de tokens de sesión (interactuado con el servicio correspondiente), y el validado de tokens OAuth de Google y Facebook, para el inicio de sesión mediante redes sociales, mediante los respectivos servicios externos, ya mencionados con anterioridad. Además, es el encargado de interactuar con la librería de manejo de usuarios estándar del framework, ASP.NET Identity, que a su vez se integra directamente con Entity Framework para el almacenamiento de credenciales e información del usuario. Destáquese que todos los usuarios, incluso aquellos que inician sesión mediante red social, se gestionan localmente a través de Identity: el token OAuth sirve como alternativa a la

tradicional contraseña, y de hecho, un usuario de estas características puede en cualquier momento establecer una contraseña local y obviar el inicio de sesión por red social en favor de la misma.

EmailService se encarga del envío de correos electrónicos (por ejemplo, el correo de recuperación de contraseña), tanto de texto plano como HTML, desde la casilla de Movelo, utilizando la librería MailKit, la más recomendada, documentada, y simple de usar entre todas las opciones contempladas.

TravelService es quizá el servicio más complejo, y contiene toda la lógica de cotización, consulta y manejo de estado de los viajes.

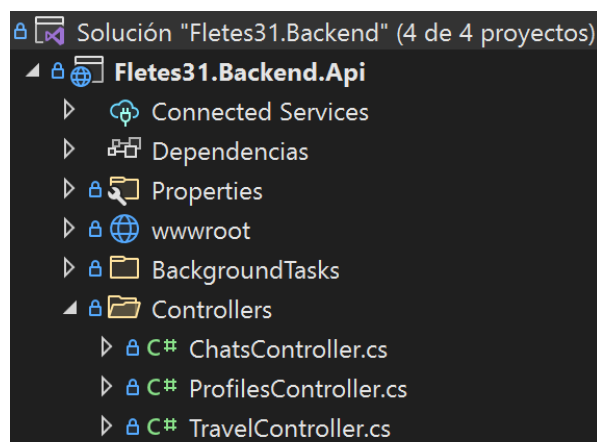
La función de cotización recibe los datos de un pedido potencial y los utiliza, en conjunto con información de un camino potencial para el envío propuesto por el *GeolocationService*, para asignarle un precio al viaje mediante una fórmula propuesta por Movelo. Devuelve al usuario un token JWT, que incluye el precio cotizado y los datos del viaje, que puede posteriormente ser usado para confirmar un viaje y, ahí sí, guardarlo en la base de datos a la espera de un conductor que lo acepte. Cabe destacar que este token tiene una validez de cinco minutos, previniendo así el inicio de un viaje basado en una cotización potencialmente desactualizada.

Existe también un método que cancela todos los viajes pendientes de un conductor que no encuentran uno pasado un determinado tiempo de espera, pensado para ser invocado desde una función recurrente.

Contiene, además, métodos para permitir a un conductor encontrar viajes potenciales pendientes de un chofer según los vehículos de los que dispone, así como para tomar un viaje, cancelarlo, modificar entre los estados intermedios de envío, o registrar una actualización en la posición actual del conductor para un envío en curso. Se integra también con el *PushNotificationService* para notificar a las partes de modificaciones en el estado del viaje. De esta manera, se envía una notificación cada vez que cambie el estado de un envío para que así los usuarios se mantengan informados respecto del estado del pedido en cuestión.

4.3.3.4. API Rest (Proyecto *Api*)

Es el proyecto de arranque de la solución, y el que referencia al framework ASP.NET directamente.



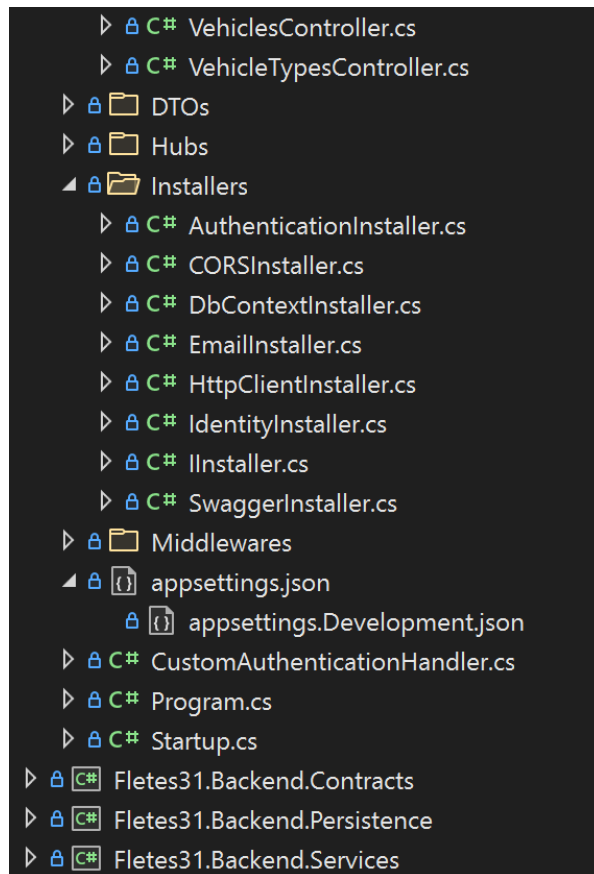


Figura 16: Contenidos del proyecto *Api*.

Contiene el archivo *appsettings.json*, que define todas las variables de configuración del sistema, como por ejemplo, la duración de los tokens JWT de inicio de sesión, existiendo este archivo en dos versiones: una para pruebas de depuración local, y otra para lanzamiento a producción.

La clase *Program* contiene la rutina de inicialización del framework mismo, incluyendo las llamadas para la verificación de existencia (y creación, de ser necesario) de la base de datos, y de los roles de usuario (cliente, conductor, administrador).

La clase *Startup*, por otro lado, se encarga de configurar las dependencias del framework mismo, así como las agregadas por el usuario. Permite, entonces, elegir y establecer parámetros para módulos integrados de ASP.NET, como Identity, CORS, o el de resolución de llamadas HTTP (que debe configurarse para mapear, manualmente, las rutas hacia archivos estáticos, como el frontend de la WebApp, o el backoffice). Es importante resaltar que, en la medida de lo posible, la configuración relativa a dependencias externas fue transportada a archivos de instalación individuales de las mismas, que se encuentran en la carpeta *Installers*, con el objetivo de evitar un caso de código inflado en la clase. Además, es aquí donde se inyectan las interfaces *IUnitOfWork* y los distintos servicios con su respectiva implementación, para que puedan luego accederse desde donde corresponda.

La carpeta *BackgroundTasks* contiene código que corre recurrentemente, en segundo plano. Actualmente, la única clase existente se ocupa de cancelar los pedidos de transporte pendientes de un conductor que superaron su tiempo máximo de vida.

La carpeta *Controllers* contiene los controladores de la API, existiendo uno por cada recurso que se exponga mediante la misma. Luego, existirá un método por cada acción que pueda ejecutarse sobre ese recurso. ASP.NET permite anotar cada uno de ellos con el método HTTP al que corresponde, la URI del recurso en el contexto del controlador, incluyendo los parámetros de URI, y las condiciones de accesibilidad en cuanto a la autenticación del usuario actual (en este caso, restringimos algunos métodos sólo a usuarios autenticados, y otros a roles de usuario específicos, como administradores).

Finalmente, cabe destacar que la API no expone directamente los modelos internos del backend, sino DTO (data transfer objects), que son transformaciones de esos modelos que existen para evitar exponer subconjuntos de información disponibles en el modelo, pero no fundamentales en el contexto de la consulta en cuestión. Esos DTO se encuentran definidos en esta misma capa, y se almacenan en la carpeta homónima.

4.4. Frontend

Para la implementación del frontend de la aplicación se tuvieron en cuenta dos aspectos fundamentales que definieron y delimitaron la elección de tecnologías a utilizar, y guiaron el proceso de desarrollo de comienzo a fin.

En primer lugar, se debía hacer foco en la usabilidad de la plataforma. Esto significa que un usuario que accede al sitio o aplicación por primera vez debe ser capaz de entender rápidamente el funcionamiento de la misma, y debe poder cotizar un envío exitosamente con la menor fricción posible. A su vez, la aplicación debe ser visualmente atractiva, ya que esto genera una experiencia más agradable, maximizando las posibilidad de retener al cliente, que vuelva a utilizarla en un futuro, y la recomiende a otras personas.

En segundo lugar, los recursos de desarrollo eran muy limitados en tiempo y tamaño del equipo, a la vez que se debía implementar la aplicación para web, iOS y Android. Por lo tanto, la elección de las herramientas de desarrollo requerían optimizar la utilización de estos recursos, pero permitir también un resultado agradable visualmente y simple de utilizar.

4.4.1. Aspectos técnicos

El frontend de la aplicación se desarrolló utilizando el framework *Flutter* [14] (en su versión 2.0), el cual es un SDK de código abierto desarrollado por Google que permite crear aplicaciones compiladas de forma nativa para Android, iOS y Web, utilizando el mismo código fuente.

Flutter se basa en el lenguaje de programación *Dart* [15], también de código abierto y desarrollado por Google, que nació como una alternativa a *JavaScript* con el objetivo de ser suficientemente flexible para la programación web, pero más estructurado que este último. Como resultado, *Dart* se encuentra optimizado para UI, lo cual en la práctica se puede evidenciar en algunas características clave que ofrece el lenguaje, tales como: *hot-reload*, esquema *async-await* robusto, funcionalidad *null-safety*, y compilación directa a JavaScript. Además, implementa una sintaxis familiar y muy simple de aprender si se parte de conocimiento previo en lenguajes como *C* o *JavaScript*.

La decisión de utilizar *Flutter + Dart* se basó principalmente en la necesidad de utilizar una herramienta cross-platform para reducir el tiempo de desarrollo, dado que escribir código fuente por separado para las tres plataformas hubiese consumido un tiempo mucho más extenso, lo cual hubiera obligado a recortar funcionalidad en la versión final del producto. Asimismo, esta decisión permitió lograr una experiencia totalmente unificada para todas las plataformas, dado que la interfaz gráfica se visualiza y se comporta exactamente igual en todas ellas.

Cabe destacar que, dadas las necesidades del producto, el desarrollo se llevó a cabo teniendo como prioridad la necesidad de obtener un diseño *responsive*, es decir, que la interfaz gráfica se adapte perfectamente a cualquier tamaño de pantalla. En la mayoría de los casos, el framework proveyó los medios necesarios para alcanzar este comportamiento, y en otros resultó necesario definir de manera programática los cambios de layout para lograr esto.

Se consideró como alternativa utilizar *React Native* [16], muy popular también para desarrollar aplicaciones cross-platform nativas, pero se consideraron algunos aspectos claves que ofrece *Flutter* que lo tornaron más idóneo para nuestro caso de uso, entre los cuales se encuentran: una mayor cantidad de componentes *out-of-the-box*, librerías nativas de geolocalización y mapa, y funcionalidad *built-in* para compilación web. Respecto de este último punto, si bien existe *React Native for Web* [17], el mismo es un *plug-in* desarrollado y mantenido por un tercero, mientras que *Flutter* tiene soporte oficial de compilación web.

Otras alternativas que se tuvieron en cuenta fueron *Ionic* y *Electron*, pero se determinó que los mismos no aportan una ventaja significativa que justifique su elección.

4.4.2. Patrones de diseño

En primer lugar, se realizó una investigación exhaustiva para determinar los patrones de arquitectura más populares y recomendados para aplicar utilizando Flutter. Como resultado, se determinó que se utilizaría *BLoC + RxDart* para el manejo de las vistas, y *Retrofit + Dio + JsonSerializable* para el manejo de las *API calls*.

4.4.2.1. BLoC + RxDart

El patrón *BLoC* (**B**usiness **L**ogic **C**omponent) es un patrón de arquitectura similar a MVVM o MVP. Se centra en mantener toda la lógica de negocio fuera de la UI, y reunirla en un solo lugar utilizando clases *BLoC*, con el objetivo de lograr una mejor separación de responsabilidades y dando como resultado un código más simple de producir y mantener[18] [19].

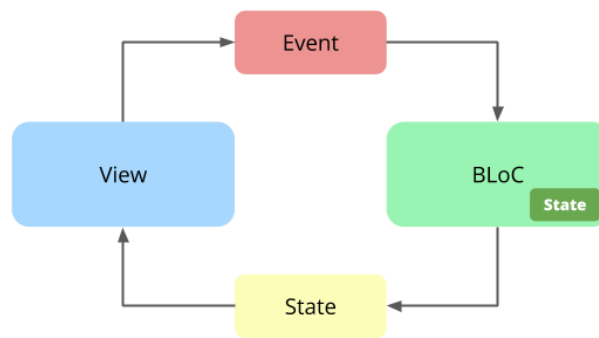


Figura 17: Esquema representativo del patrón *BLoC*. Fuente: [18].

Para lograr este comportamiento, el patrón se apoya en el paradigma de programación reactiva. Es decir, la información puede fluir del BLoC a la UI o de la UI al BLoC en forma de *streams*, los cuales permiten enviar y consumir datos en forma de eventos ordenados, notificando al receptor cuando un nuevo evento está disponible para ser consumido.

Por lo tanto, para la implementación de este patrón se utilizó también la librería *RxDart* [20] que ofrece clases y métodos enfocados en el paradigma de la programación reactiva. Si bien es posible implementar el patrón *BLoC* sin utilizar esta librería, es altamente recomendable su uso dado que de esta manera se obtiene un código más compacto y legible.

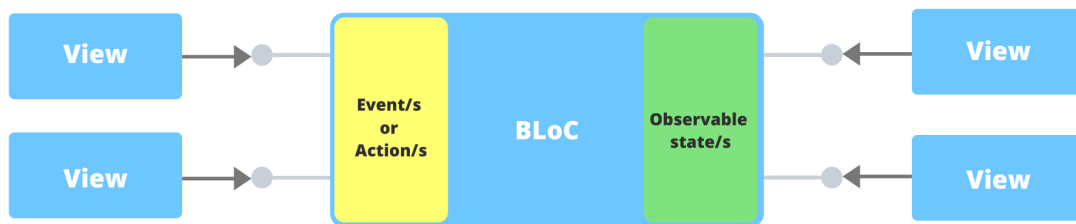


Figura 18: Representación del patrón *BLoC* con *RxDart*. Fuente: [19].

4.4.2.2. Retrofit + Dio + JsonSerializer

Se utilizó el paquete *retrofit* [21] como capa de abstracción del cliente http *dio.dart*, el cual permite definir todas las llamadas a la API utilizando únicamente interfaces de Dart y *annotations*. Este paquete provee una funcionalidad muy completa, dando la posibilidad de personalizar el método http, *path params*, *query params* y *headers* de cada request que se necesite definir.

Además, para la serialización en forma de JSON de los modelos, se utilizó el paquete *json_serializable*[22]. El mismo soporta todos los tipos nativos del lenguaje dart, así como también la serialización de mapas, listas y enumerables. También permite serializar clases de forma recursiva (es decir, una clase que contiene como campo a otra clase serializable) y clases con *generics*, como se puede ver en el caso de la clase *PagedList<T>*.

Un aspecto muy importante a tener en cuenta para entender el funcionamiento de *retrofit* y *json_serializable* es que Dart no soporta reflexión. Esto quiere decir que el código no puede generarse en tiempo de ejecución, como ocurre por ejemplo en Java, sino que debe generarse de manera estática. Para ello se requiere ejecutar un comando que lee los archivos del proyecto en búsqueda de *annotations* de *retrofit* y *json_serializable*, y genera en los mismos directorios archivos con el código resultante correspondiente listo para ser ejecutado. En la sección 6 del presente informe se detalla el comando en cuestión y su modo de uso.

Todas las librerías mencionadas se encuentran entre las de mayor reputación y soporte dentro de la comunidad de Flutter, por lo que son actualizadas regularmente y resulta fácil encontrar documentación de su uso y respuestas a preguntas frecuentes.

4.4.2.3. Estructura del proyecto

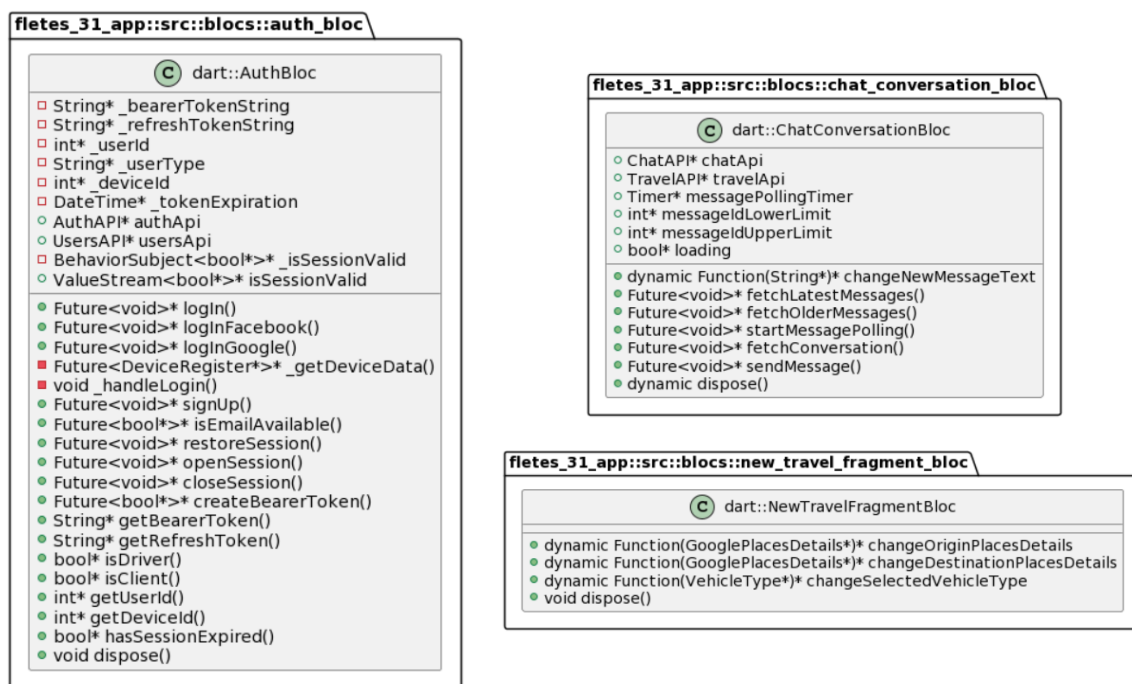
Teniendo en cuenta lo mencionado anteriormente, la estructura de directorios del proyecto resulta:

lib

src

- ▢ **blocs:** contiene todos los archivos correspondientes al patrón bloc.
- ▢ **models:** contiene los modelos y DTOs del proyecto.
- ▢ **network:** contiene los clientes http para comunicarse con el backend.
- ▢ **ui:** contiene las vistas del frontend.
- app.dart:** define el *scaffolding* de la UI.
- main.dart:** define el método *main()* que se ejecuta al iniciar la aplicación.

4.4.3. Modelo de clases



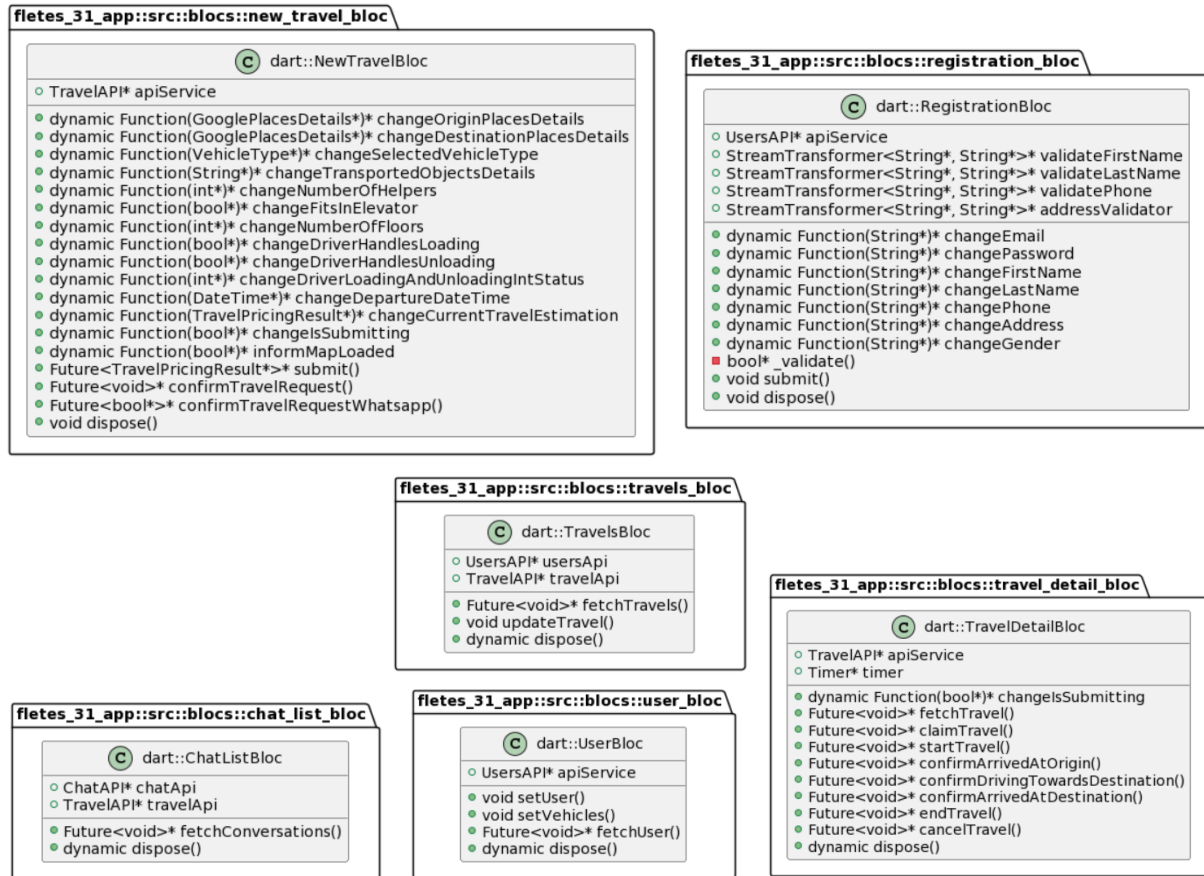
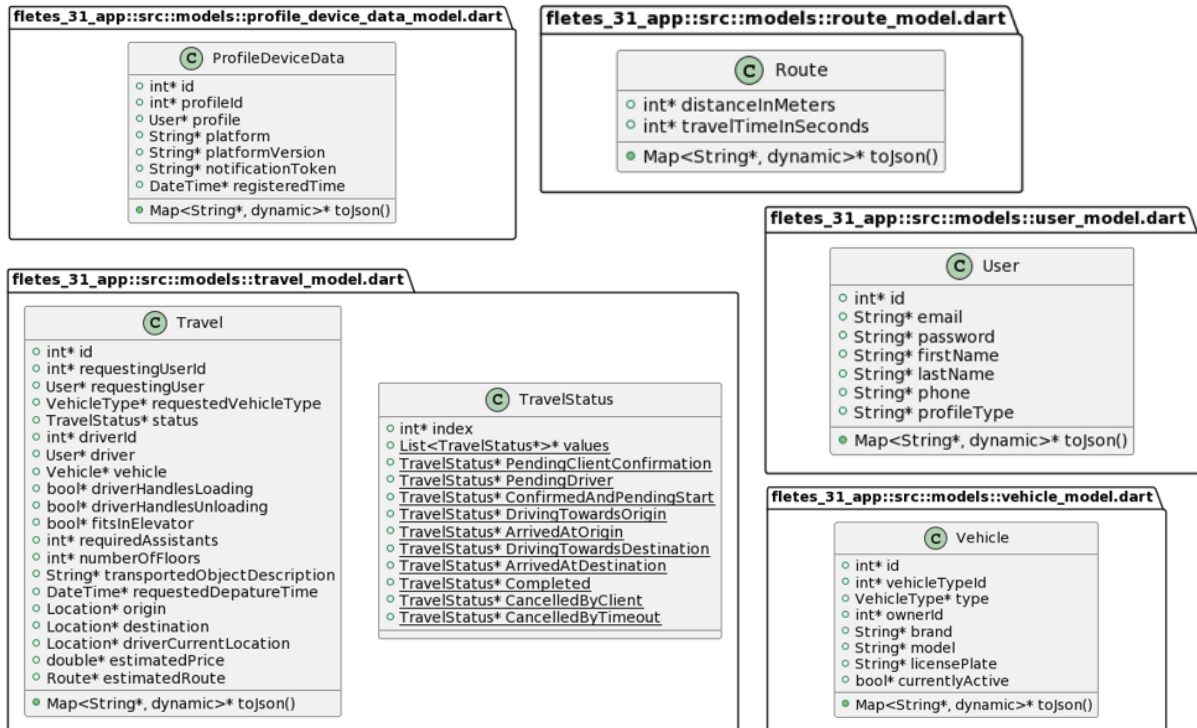


Figura 19: Diagrama UML de las clases BLoC. Se omiten los *streams*.



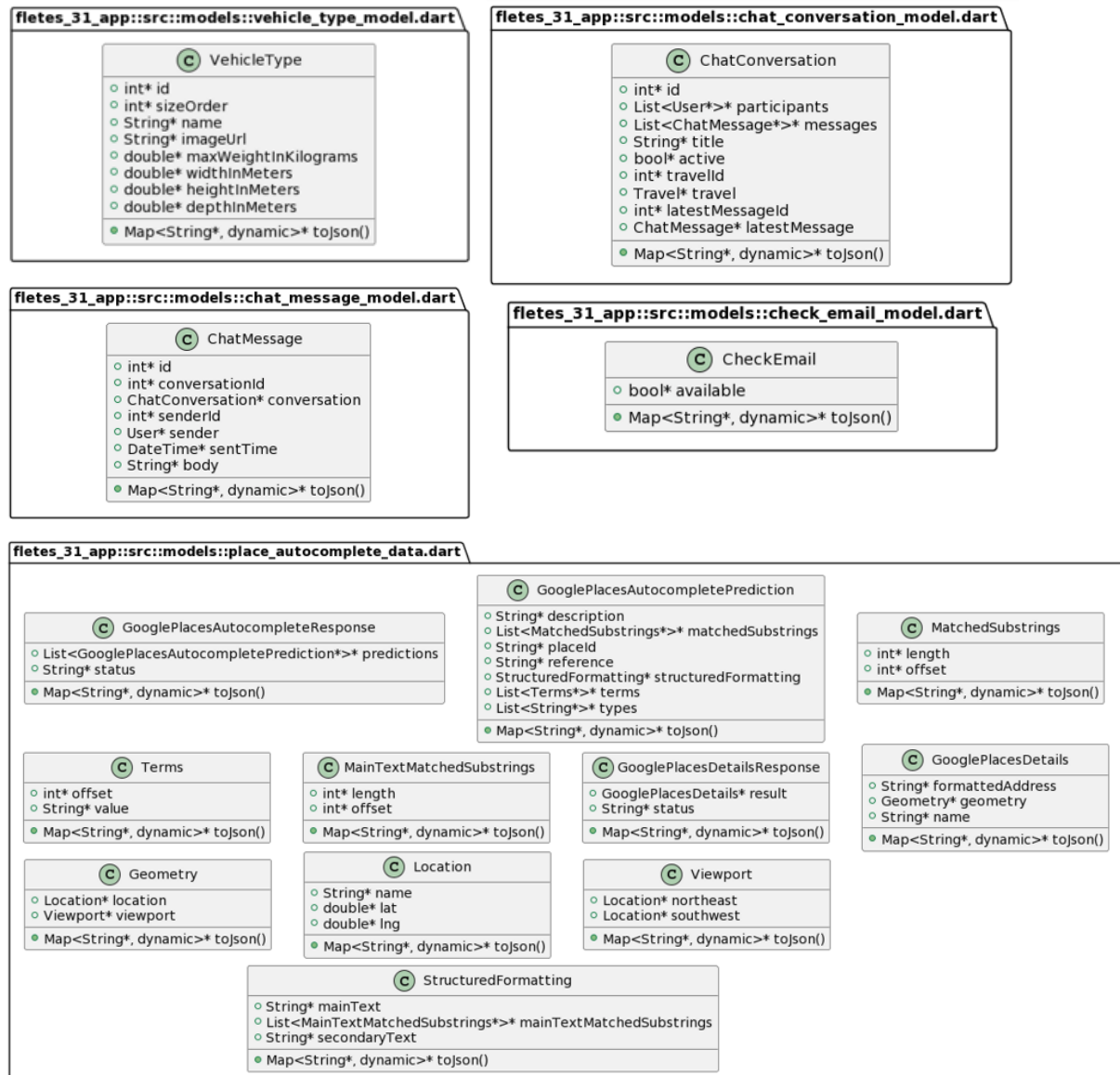


Figura 20: Diagrama de clases UML de los modelos.

4.4.4. Diseño de interfaces

En esta sección se describen aspectos generales de la interfaz implementada para la aplicación. El diseño de estas interfaces fue maquetado en primer lugar por una diseñadora gráfica aportada por el Gobierno de la Ciudad de Buenos Aires, quien se encargó de producir los recursos gráficos necesarios tales como íconos, logos, paleta de colores e imágenes usadas en la portada y los banners. Todo este proceso fue realizado teniendo en cuenta la usabilidad de la aplicación como una de las prioridades, y buscando una experiencia fluida y consistente para el usuario final.

En primer lugar, se estableció la necesidad de producir dos interfaces diferentes para los clientes y para los conductores. La funcionalidad más importante para el cliente es la de poder cotizar y solicitar un envío, mientras que para el conductor es la de poder aceptar envíos y consultar envíos pendientes. Cabe aclarar que para todas las pantallas se proveyeron maquetas para tres tamaños de pantalla: móvil, tablet y escritorio.

Dado que el proyecto experimentó un proceso de *rebranding* durante el desarrollo del mismo, las muestras visuales que se exponen a continuación corresponden a los cambios definitivos que se incluyeron en el producto final.



Figura 21: Logo completo (a la izquierda) y reducido (a la derecha).

El color principal de la aplicación es un púrpura, como se puede apreciar en el logo, y como color secundario un verde claro. La elección de esta paleta de colores se hizo en coordinación con el GCBA junto con los fleteros, con la intención de darle una apariencia más animada y llamativa a la aplicación.

El diseño de la pantalla principal de la aplicación cuenta una *landing* en la versión web, para hacer más atractiva y fluida la experiencia de un usuario que accede al sitio por primera vez. En la misma se explica de forma simple y concisa los pasos requeridos para cotizar un envío, así como también un detalle de los servicios ofrecidos por la plataforma y un listado de empresas que utilizan el servicio mediante convenios con el GCBA, para brindar una mayor confianza a los potenciales clientes que acceden al sitio.



Figura 22: Sección superior de la landing en pantalla grande.

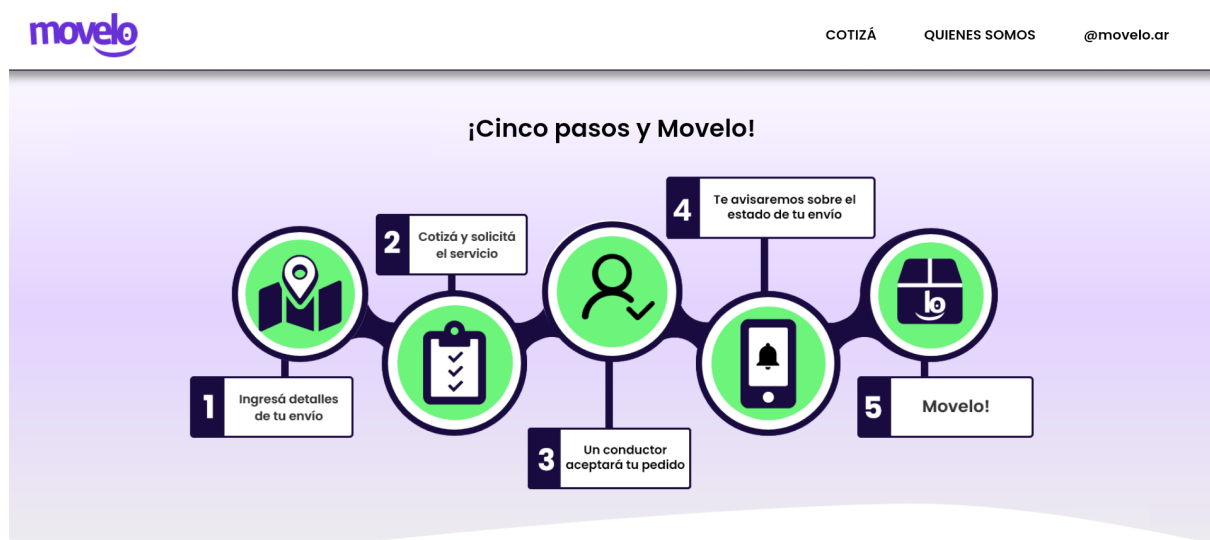


Figura 2: Sección media de la landing en pantalla grande (instrucciones).

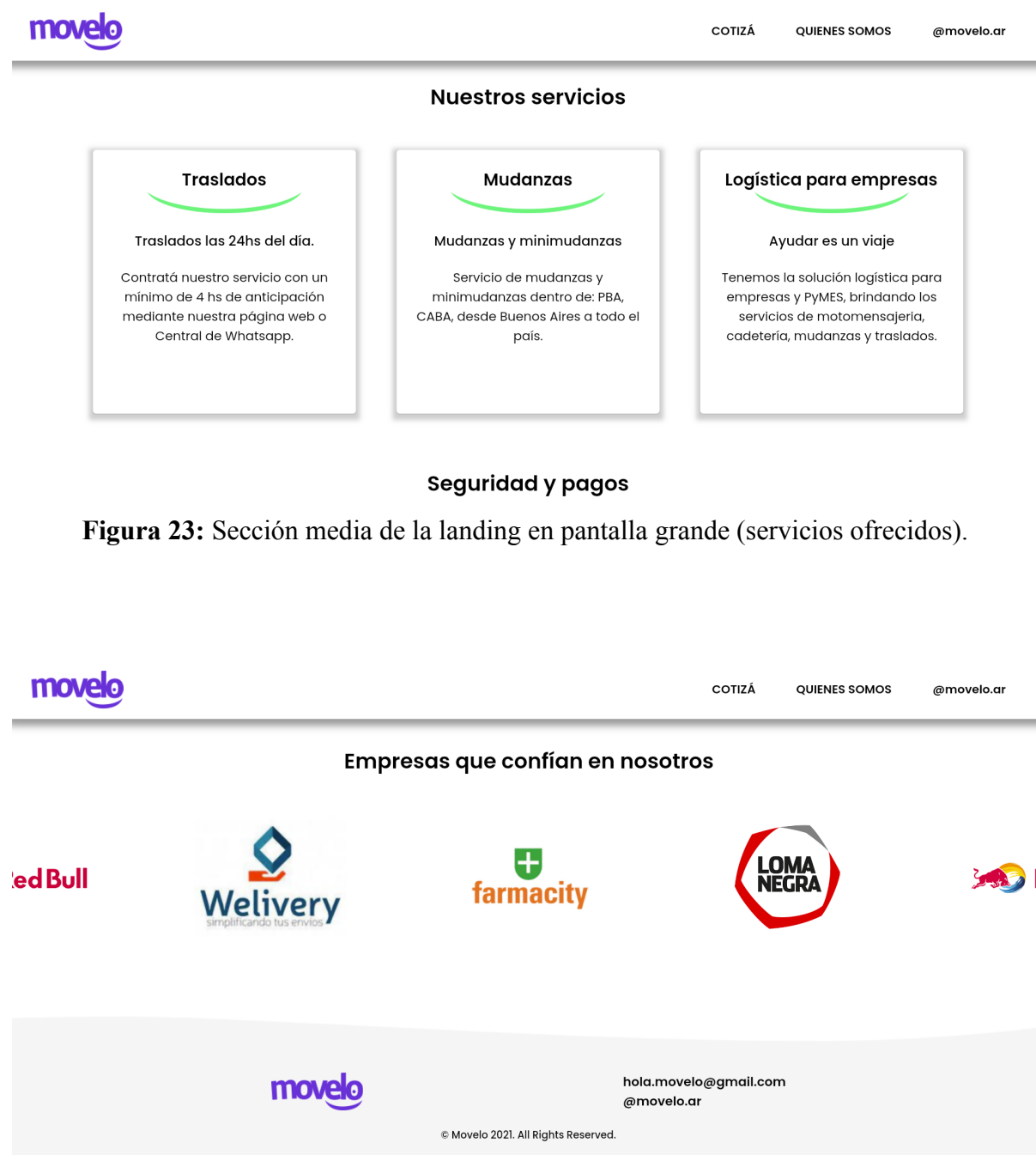


Figura 23: Sección media de la landing en pantalla grande (servicios ofrecidos).

Figura 24: Sección inferior de la landing en pantalla grande.

En la landing se muestra en primer plano un diseño reducido del cotizador, que le solicita al usuario una mínima cantidad de datos para que se pueda cotizar el envío. Luego, al seleccionar “Cotizar ahora” se lo redirige a una pantalla con un cotizador más completo que además muestra un mapa para visualizar el origen y destino del envío que se desea realizar.

En este cotizador más completo se puede configurar otros parámetros necesarios para obtener una cotización acorde a las necesidades del usuario, como es el caso de si se requiere la carga y descarga de los objetos transportados. Se muestra también en esta pantalla un mapa con marcadores de origen y destino, de manera tal que permita al cliente validar rápidamente que las direcciones ingresadas hayan sido correctamente detectadas por el cotizador. Junto

con esto, se muestra sobre el mapa el tiempo estimado que dura el viaje desde el origen hasta su destino y la cantidad de kilómetros a recorrer. Dicha información se obtiene del backend junto con el precio del envío.

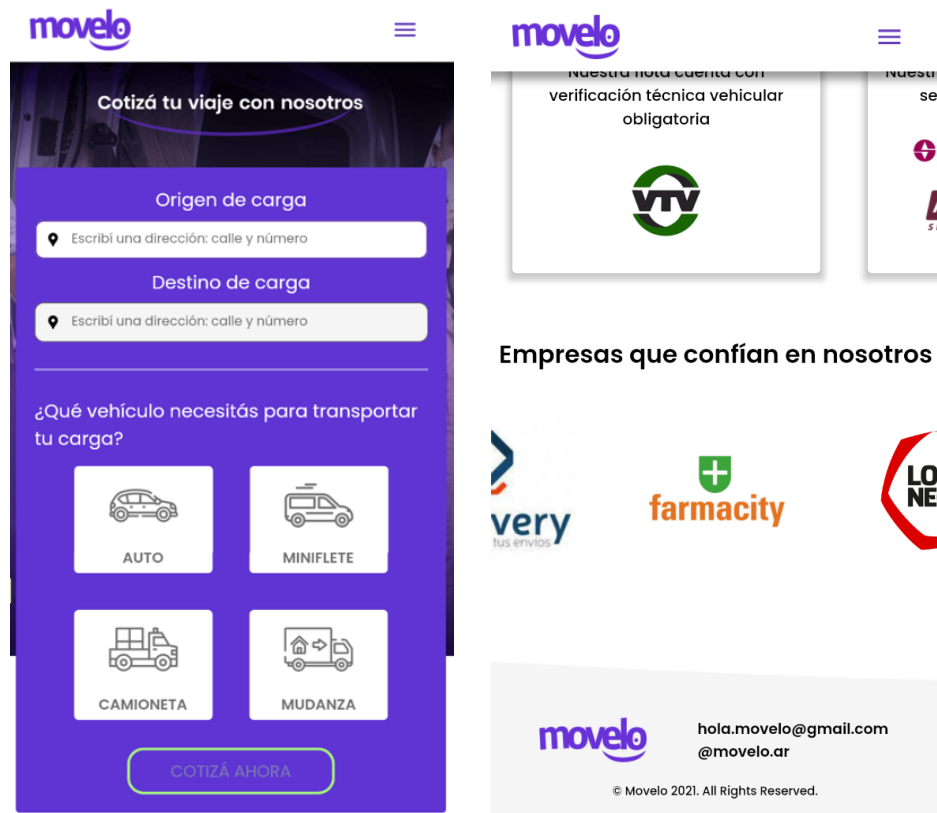


Figura 25: Secciones superior e inferior de la landing en pantalla chica.

El cotizador recibe además como parámetros la descripción de los objetos que requieren ser transportados y la cantidad de pisos que deben bajarse los objetos en caso de que se requiera ayuda del conductor y que estos objetos no puedan bajarse por ascensor.

Como se puede observar en las figuras a continuación, el cotizador modifica la disposición de sus elementos de acuerdo al tamaño de pantalla. Cuando el mismo se presenta en pantalla grande, el mapa se muestra a la derecha del formulario de cotización para obtener un mejor aprovechamiento del espacio disponible. En cambio, en pantalla chica los elementos tienen una disposición vertical, de forma tal que los datos del formulario se puedan cargar a medida que se desliza la pantalla hasta llegar al final con la cotización y los botones de confirmación del pedido. De esta manera, se busca obtener un diseño lo más óptimo posible para cada plataforma, haciendo foco en la usabilidad.

Otras de las pantallas principales de la aplicación son las de inicio de sesión y registro. Dado que la plataforma permite hacer un seguimiento de los envíos en todas sus etapas, es necesario iniciar sesión para asociar estos envíos a un usuario y mantener un contacto directo con el cliente sobre los cambios de estados del mismo.

Cabe destacar, además, que tanto a conductores como a clientes corresponde la misma aplicación. Como este informe especifica en cada caso, las pantallas accesibles variarán, una

vez iniciada la sesión, según el tipo de usuario actual, permitiendo a cada uno concretar las acciones que le corresponden.

Figura 26: Vista del cotizador completo en pantalla grande.

Figura 27: Vista del cotizador en pantalla chica.

Teniendo en cuenta que se quiere disminuir lo máximo posible la barrera de entrada de los usuarios a la aplicación, y generar la menor cantidad de fricciones posibles, la plataforma permite cotizar pedidos de manera anónima sin necesidad de iniciar sesión previamente. Es al momento de confirmar el pedido que se solicita al usuario iniciar sesión en la aplicación, en caso de que no estuviera previamente iniciada.



Figura 28: Inicio de sesión (izquierda) y registro (derecha) en pantalla chica.

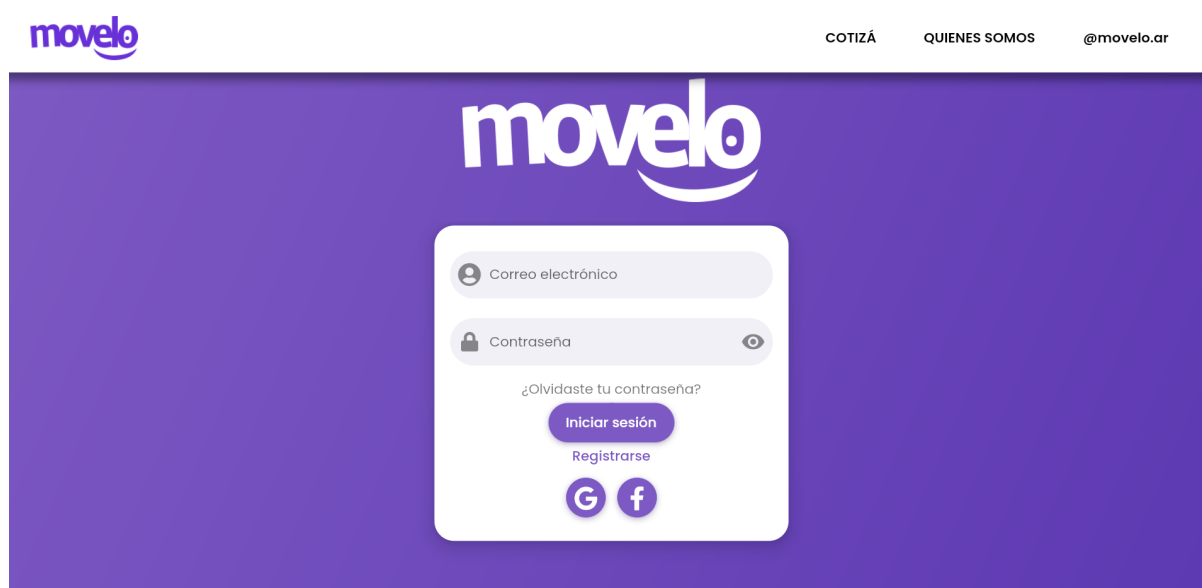


Figura 29: Inicio de sesión en pantalla grande.

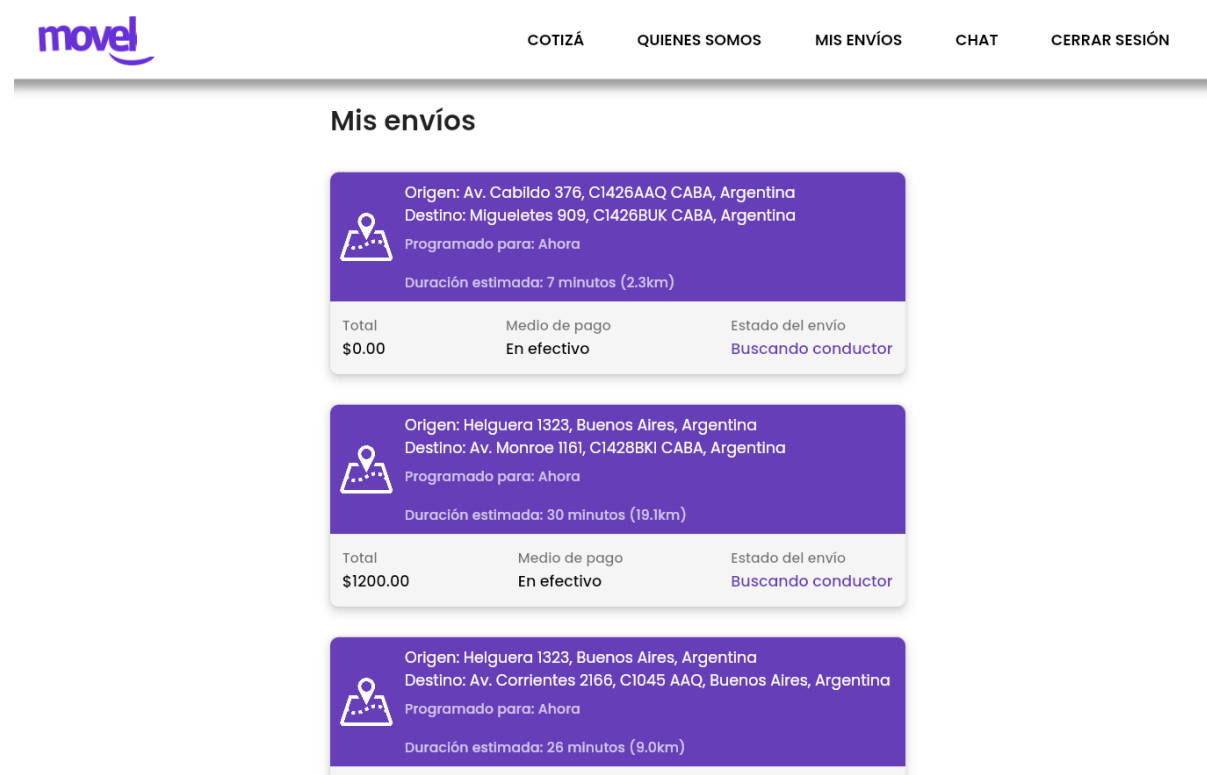


Figura 30: Lista de envíos en pantalla grande.

Una vez cotizado y aceptado un envío, se redirige a la pantalla de detalle de envío, el cual muestra el mapa con los marcadores de origen y destino en pantalla completa, y un panel inferior deslizante donde se puede ver más datos del envío, tales como: estado, conductor (en caso de haberse asignado uno), datos ingresados en el cotizador y número de contacto en caso de tener un problema. Además, se incluye un botón de cancelación del envío que se encuentra operativo cuando aún no se ha asignado un conductor, y en el mapa se muestra la ubicación actual del conductor siempre y cuando el envío se encuentre en curso.

Esta pantalla es casi idéntica a la que ve el conductor, salvo que en vez de ver un botón de cancelación, se muestra un botón que permite aceptar el envío o avanzar el estado del envío, según corresponda, y en vez de mostrar el nombre del conductor se muestra el nombre del cliente.

Por otro lado, se puede ver un listado de todos los envíos solicitados (en caso de ser un cliente), o de todos los envíos aceptados (en caso de ser un conductor). En este listado se muestra únicamente el origen y destino del envío, el tiempo y distancia estimado, y si el envío se encuentra programado o se solicitó para el momento. Esta información se muestra en forma de tarjetas que redirigen a la pantalla de detalle de envío al presionarlas.



Figura 31: Pantalla de detalle de un envío (mapa).

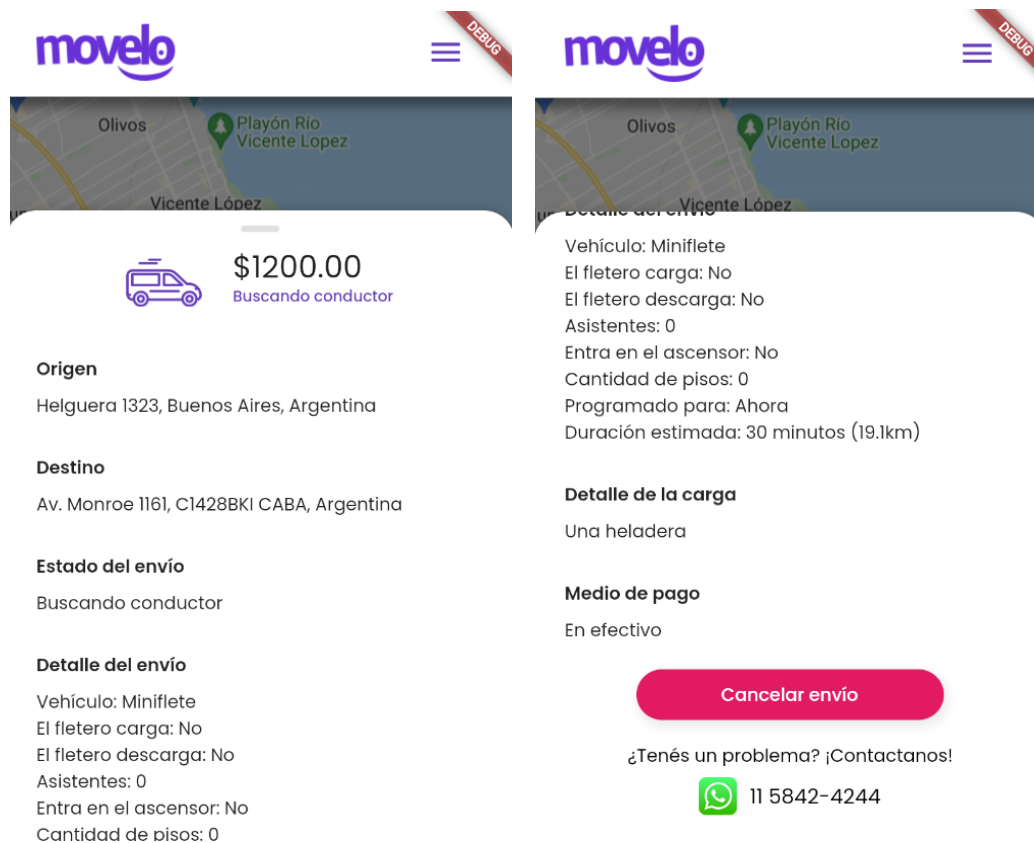


Figura 32: Pantalla de detalle de un envío (panel deslizante).

Se incorpora en todas las pantallas una barra de navegación que permite navegar mediante las funciones principales de la aplicación. Dicha barra de navegación se encuentra diseñada para que se adapte de acuerdo al tamaño de pantalla. De esta forma, cuando el ancho de la pantalla no permite que todas las opciones se muestren horizontalmente, se compactan en un menú vertical desplegable que las contenga.

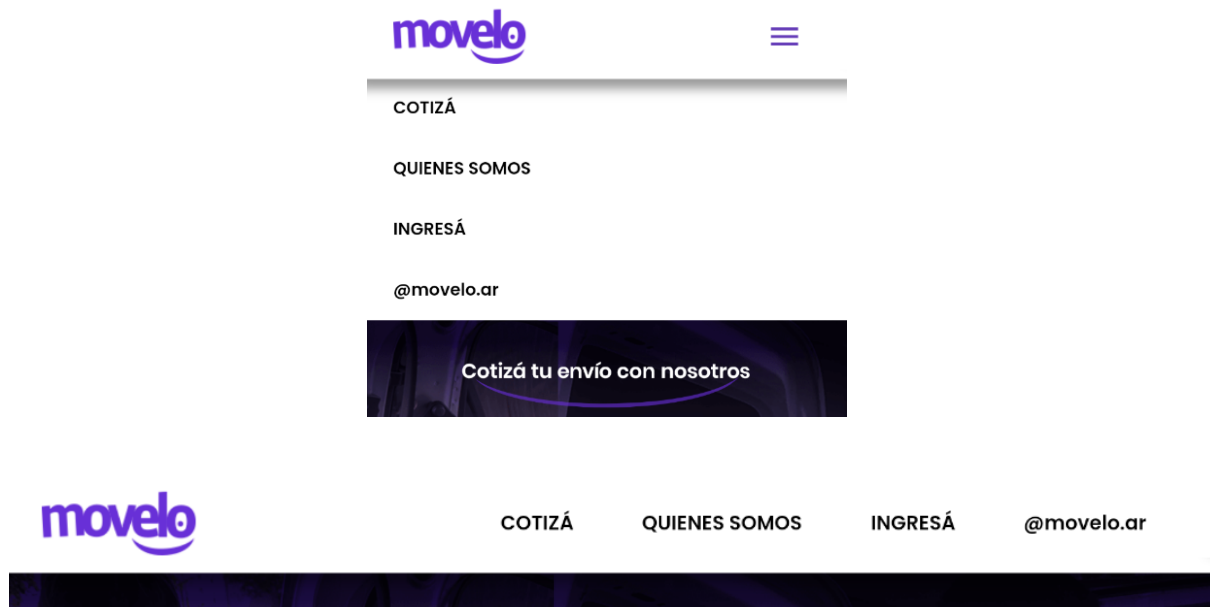


Figura 33: Barra de navegación en pantalla chica (arriba) o grande (abajo).

Por último, se incluyeron pantallas de “Quiénes somos” y “Contactanos”.

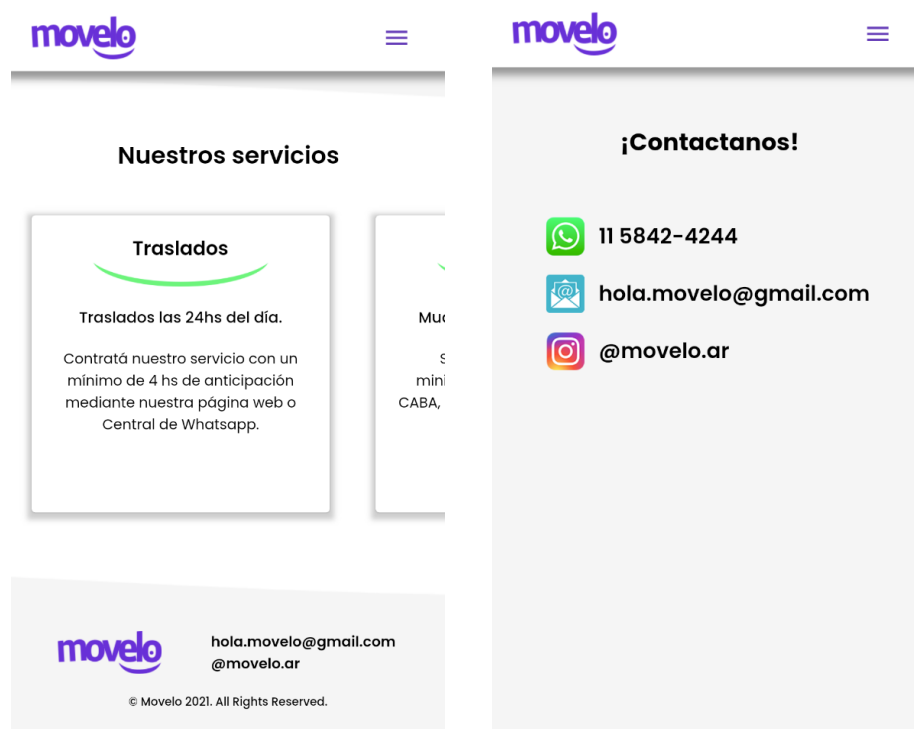


Figura 34: “Quiénes somos” (a la izquierda) y “Contactanos” (a la derecha).

4.4.5. Cotización de envíos

Para realizar la cotización de un envío, el *backend* provee un *endpoint* que recibe los parámetros del envío y retorna un objeto con el resultado de la cotización, junto con un token de validación. Este token sigue el estándar JWT y cumple el propósito de evitar tener que guardar cada cotización en la base de datos, pero a la vez permitir al usuario congelar el precio de dicha cotización. Es decir, al confirmar una cotización, el frontend envía un token que se encuentra firmado por el backend, en el que figuran no solo los datos de la cotización, sino también el precio de la misma. De esta forma, no es necesario almacenar el precio de cada cotización realizada para verificar que este no fue manipulado.

La ventaja de este acercamiento es, por un lado, que se ahorra espacio en la base de datos ya que solo se almacenan los envíos confirmados. Y por otro lado, que si un usuario primero cotiza y luego se registra por primera vez en la plataforma, su cotización queda vigente para ser aceptada ya que el token no se asocia a ningún usuario en particular.

Cabe destacar que estos tokens tienen un tiempo de expiración de unos pocos minutos, con el propósito de evitar que sea posible confirmar un envío mucho después de haber sido cotizado, debido a que el precio puede ir quedando desactualizado a lo largo del tiempo.

A continuación se muestra un ejemplo de request y response del endpoint mencionado:

POST /travels

Request:

```
{
  "vehicleTypeId": 2,
  "origin": {
    "name": "Helguera 1323, Buenos Aires, Argentina",
    "lat": -34.6175176,
    "lng": -58.4800575
  },
  "destination": {
    "name": "Av. Monroe 1161, C1428BKI CABA, Argentina",
    "lat": -34.5504598,
    "lng": -58.44705059999999
  },
  "departureDateTime": null,
  "driverHandlesLoading": false,
  "driverHandlesUnloading": false,
  "fitsInElevator": false,
  "numberOfFloors": 0,
  "requiredAssistants": 0
}
```

Response:

```
{
```



```
"travelPricingToken": "eyJh...9FbE",
"travel": {
  "id": 0,
  "origin": {
    ...
  },
  "destination": {
    ...
  },
  "requestedVehicleTypeId": 2,
  ...
  "status": "PendingClientConfirmation",
  "estimatedRoute": {
    "distanceInMeters": 19138,
    "travelTimeInSeconds": 1820
  },
  "estimatedPrice": 3122.40
}
```

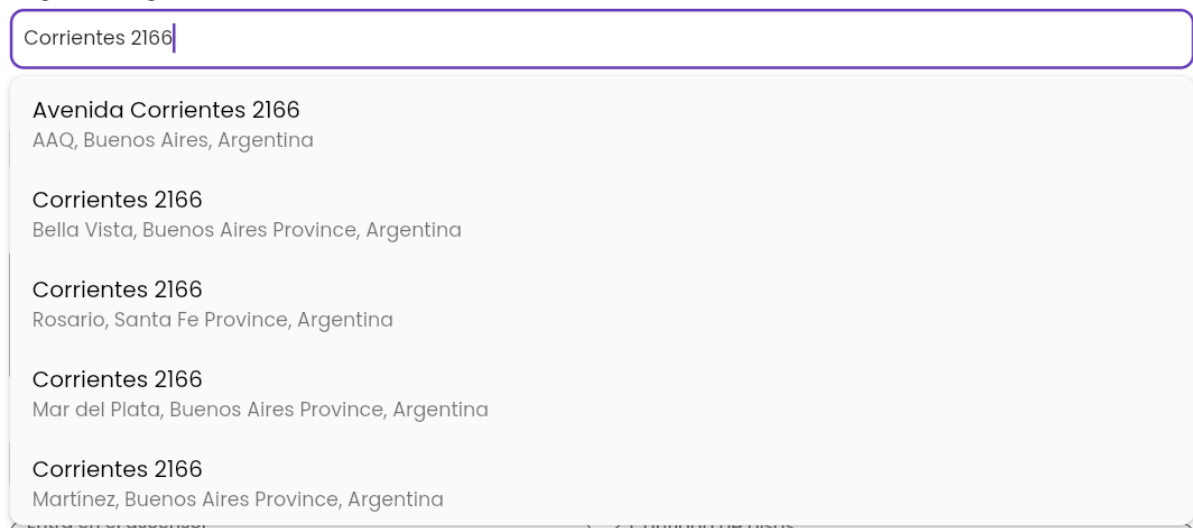
Una vez que se completan los datos del cotizador, se ejecuta el request utilizando el endpoint descrito previamente, y al obtenerse la respuesta se muestra el precio, la distancia y el tiempo estimado dibujado sobre el mapa. En este momento se habilita el botón de confirmación del pedido que ejecuta el request correspondiente al presionarlo, el cual guarda el envío y lo hace visible para los conductores que pueden eventualmente aceptar el mismo.

Si el cliente no tiene su sesión iniciada al presionar este botón, se muestra un modal con un mensaje que le solicita iniciar sesión. Una vez iniciada la sesión, se retorna al cotizador manteniendo los datos y la cotización realizada.

Otro detalle a tener en cuenta es que la cotización se realiza automáticamente al completar o actualizar el formulario, y mientras se está esperando la respuesta del backend se muestra una barra de progreso sobre el botón de confirmación. De esta forma, se obtiene una experiencia de uso más fluida ya que evita la necesidad de presionar un nuevo botón cada vez que se desea refrescar la cotización, mostrando siempre la cotización correspondiente a los parámetros ingresados.

Otro de los componentes del cotizador con un comportamiento particular son los campos donde se ingresa el origen y destino del envío. Los mismos muestran un menú desplegable a medida que se completan donde se muestra una sugerencia de las posibles direcciones a las que puede estar haciendo referencia el valor ingresado. Es necesario seleccionar uno de estos valores para que el campo sea considerado como completado. Cuando ambos campos (origen y destino) se encuentran completos, se dibujan los marcadores correspondientes en el mapa que se muestra en esta misma pantalla.

Origen de carga



The image shows a web form with a label 'Origen de carga' above a text input field. The input field contains the text 'Corrientes 2166'. Below the input field, a dropdown menu is open, displaying five suggestions. Each suggestion consists of a bolded address followed by a location in parentheses. The suggestions are: 'Avenida Corrientes 2166 (AAQ, Buenos Aires, Argentina)', 'Corrientes 2166 (Bella Vista, Buenos Aires Province, Argentina)', 'Corrientes 2166 (Rosario, Santa Fe Province, Argentina)', 'Corrientes 2166 (Mar del Plata, Buenos Aires Province, Argentina)', and 'Corrientes 2166 (Martínez, Buenos Aires Province, Argentina)'. The dropdown menu has a light gray background and rounded corners.

Suggestion
Avenida Corrientes 2166 AAQ, Buenos Aires, Argentina
Corrientes 2166 Bella Vista, Buenos Aires Province, Argentina
Corrientes 2166 Rosario, Santa Fe Province, Argentina
Corrientes 2166 Mar del Plata, Buenos Aires Province, Argentina
Corrientes 2166 Martínez, Buenos Aires Province, Argentina

Figura 35: Campo autocompletable de origen del envío.

Para desplegar la lista de sugerencias, se utiliza la API de Google Maps, la cual provee un endpoint para permitir el autocompletado de esta clase de campos, retornando una lista de ubicaciones que matchean con la búsqueda realizada.

4.4.6. Seguimiento de envíos

El seguimiento del envío tiene principalmente dos partes: la actualización del estado del envío, y la actualización de la posición actual del conductor.

En primer lugar, el estado del envío siempre se actualiza por una acción del conductor. Desde la aceptación del envío hasta la finalización del mismo, es un conductor quien debe indicar que se pasa de un estado al siguiente. El único caso en el que un cliente puede alterar el estado de su pedido es en el caso de cancelación, el cual como se mencionó anteriormente solo es posible cuando el envío aún no se ha iniciado.

Estas actualizaciones se realizan siempre desde la pantalla de detalle de un envío, utilizando el botón que se encuentra en la parte inferior del menú desplegable. Al presionarlo, se muestra siempre un cartel de confirmación para evitar que se presione accidentalmente el botón, ya que no existe forma de deshacer esta operación.

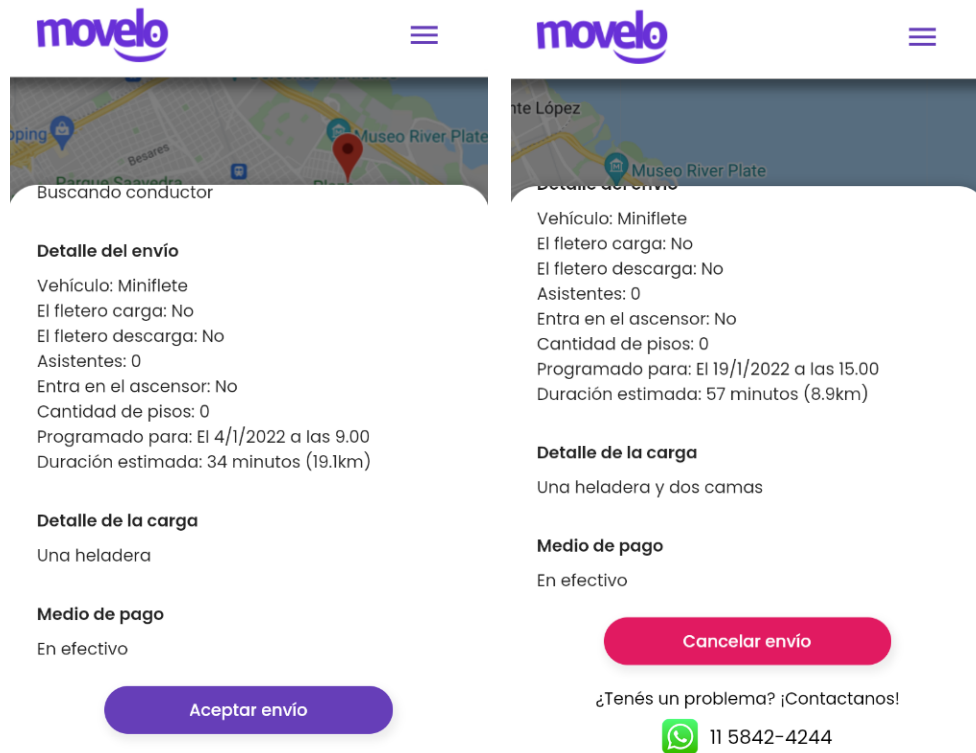


Figura 36: Botón de aceptación de viaje (a la izquierda) o de cancelación de viaje (a la derecha), que se muestra dependiendo del rol del usuario (cliente o conductor).

Los posibles estados que puede tomar un envío son:

```
[
  "PendingDriver",
  "ConfirmedAndPendingStart",
  "DrivingTowardsOrigin",
  "ArrivedAtOrigin",
  "DrivingTowardsDestination",
  "ArrivedAtDestination",
  "Completed",
  "CancelledByClient",
  "CancelledByTimeout"
]
```

Cuando el envío se encuentra en estado de *DrivingTowardsOrigin*, *ArrivedAtOrigin*, *DrivingTowardsDestination* o *ArrivedAtDestination*, se realiza un seguimiento de la ubicación del conductor y se muestra dibujada en el mapa.

Para ello, se utiliza una función recurrente que se ejecuta cada 15 segundos en el frontend la cual se encarga de sincronizar esta información entre el conductor y el cliente. En el caso del cliente, se realiza un GET al backend para actualizar el estado del envío y posición del conductor, y en el caso del conductor se realiza un PUT al backend para persistir su ubicación actual.

Para obtener la información de ubicación del conductor, se utiliza el paquete *geolocator* de Flutter, el cual abstrae para todas las plataformas el acceso al servicio de localización, resolviendo de forma transparente la solicitud del permiso de localización y retornando una promesa con el dato requerido.

En una primera instancia, se consideró utilizar SignalR (que a su vez utiliza WebSockets) para la actualización del estado del envío, ya que resulta más eficiente en uso de red y consumo de recursos que hacer *polling* (como se hace actualmente). Sin embargo, se optó por descartar esta opción debido a que Flutter no cuenta aún con un paquete lo suficientemente maduro para el uso de SignalR.

4.4.7. Push notifications

Para el manejo de notificaciones en el frontend, se hace uso de los paquetes *firebase_core* y *firebase_messaging*. Con estos paquetes, se puede obtener un token único de dispositivo generado por Firebase, y se lo envía al momento de iniciar sesión en la aplicación. De esta forma, se puede escuchar activamente al servicio de Firebase para responder ante una señal de recepción de notificación, a través de eventos definidos por el backend.

Para ello, se puede definir un comportamiento en caso de recibirse la notificación con la aplicación abierta, minimizada o cerrada. Por defecto, el servicio de Firebase muestra el título y contenido de la notificación utilizando el formato nativo de cada plataforma. En el caso del presente proyecto, como solo se requiere avisar al usuario de un cambio en un estado de su envío o de la recepción de un mensaje en un chat, no se definió ningún comportamiento personalizado, manteniéndose el comportamiento por defecto.

4.4.8. JWT

Dado que ya se explicó previamente, en la sección de backend, el funcionamiento específico de los tokens JWT para la autenticación del usuario en la aplicación MoveLo, en esta sección solo se mencionan los aspectos particulares de frontend vinculados al almacenamiento y uso del token.

Para el almacenamiento del token JWT se utiliza la librería *shared_preferences*, la cual abstrae funcionalidad de almacenamiento persistente para cada plataforma (*NSUserDefaults* en iOS, *SharedPreferences* en Android y *LocalStorage* en web). Para ello, provee métodos para almacenar y recuperar pares clave-valor, que se utilizan para almacenar el token en cuestión, así como otros datos asociados a la sesión, tales como el id del usuario, el tipo de usuario (cliente o conductor), y el id del dispositivo.

Cuando se inicia sesión en la aplicación, se almacena toda la información mencionada en el almacenamiento persistente. Cuando la aplicación se reinicia, se verifica si hay almacenado un token de sesión, y se lo utiliza para hacer un GET al backend del usuario. Si el mismo es exitoso, significa que el token continúa siendo válido y se almacena esa información mientras la aplicación se mantenga abierta. En caso de error, o en caso de que no exista ningún token almacenado, se borra toda la información del almacenamiento persistente, y se considera que no hay ninguna sesión iniciada.

Para autenticar los requests, se debe incluir el token de sesión en el *header* “Authorization” de los mismos. Para ello, se utiliza la funcionalidad de *interceptors* que provee el paquete *Dio*, el cual permite ejecutar una determinada acción antes o después de cada request, que permite por ejemplo inyectar un header específico. Por lo tanto, se configura un *interceptor* que verifica si el usuario tiene una sesión iniciada, y se agrega el header mencionado con el token correspondiente para autenticar la llamada. Cabe mencionar que dicho interceptor solo se encuentra configurado para los endpoints de usuario, envíos y chat, que son los que requieren autenticación del usuario.

Por otro lado, para el flujo de sesiones de la aplicación se utiliza un segundo token de refresco. El mismo se utiliza cuando el token de sesión expira (el cual dura solo unos minutos), para obtener uno nuevo. Este funcionamiento responde a la necesidad de bloquear usuarios (ya sean clientes o conductores) por mala conducta o por no respetar las normas de uso de la aplicación. De esta forma, cuando un usuario es bloqueado, solo podrá seguir utilizando la aplicación durante los pocos minutos que tarda en expirar el token de sesión, pero al intentar refrescar el mismo, el backend le denegará el request.

Para refrescar el token de sesión se utilizan los mismos *interceptors* mencionados anteriormente, de forma tal que se verifica antes de realizar el request si el mismo se encuentra vigente comprobando la fecha y hora de expiración. Si se encuentra expirado, se realiza previamente una llamada al endpoint de refresco del token y cuando se obtiene la respuesta, se prosigue con la llamada original.

4.4.9. Social Login

La aplicación permite iniciar sesión, no solo mediante una cuenta propia en la plataforma Moveo, sino también a través de una cuenta de Facebook o de Google. Para facilitar esta tarea, se utilizan los paquetes *google_sign_in* y *flutter_facebook_auth*. Se deben utilizar los sitios *developers.facebook.com* y *console.firebase.google.com* para registrar la aplicación en Facebook y en Google, respectivamente.

Por defecto, se puede acceder a información básica del perfil (nombre, apellido, email y foto), pero se pueden solicitar más permisos para acceder a otro tipo de datos, como la fecha de nacimiento o domicilio del usuario. Para más información: Facebook y Google.

En ambos casos, el frontend obtiene un token (una vez que el usuario introduce sus credenciales de acceso correspondientes), el cual es enviado al endpoint de autenticación del backend, que se encarga de validarlo contra los servidores de Facebook o Google y dar finalmente acceso al usuario.

4.4.10. Chat

La plataforma cuenta además con un sistema integrado de chat entre clientes y conductores. De esta forma, los usuarios pueden utilizar esta herramienta para realizar preguntas o indicaciones que tengan que ver con el servicio brindado. Esta funcionalidad puede resultar muy útil siempre que surjan dudas acerca de la carga transportada, de la dirección de origen o destino, del tiempo estimado de entrega, o en caso de una eventualidad que modifique las expectativas de entrega.

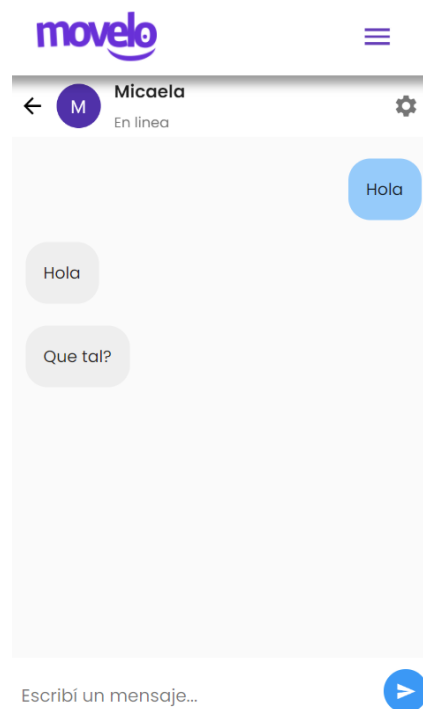


Figura 37: Pantalla de chat.

4.4.11. Funcionalidad específica por plataforma

Al ser Flutter un framework multiplataforma, este permite que se realicen configuraciones o que se ejecute código específico por plataforma. Se describen a continuación aquellas funcionalidades que requirieron un tratamiento específico en alguna plataforma.

4.4.11.1. Permisos de Geolocalización

Este permiso se utiliza para registrar la ubicación actual del dispositivo donde se ejecuta la aplicación. En Android se requirió modificar el archivo **AndroidManifest.xml** y añadir las siguientes líneas:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Para iOS la configuración es similar. En este caso se modificó el archivo **Info.plist** añadiendo las siguientes líneas:

```
<key>NSLocationWhenInUseUsageDescription</key>
<string>This app needs access to location when open.</string>
<key>NSLocationAlwaysUsageDescription</key>
<string>This app needs access to location when in the background.</string>
```

4.4.11.2. Permisos de ejecución de links

Para permitir ejecutar links desde las aplicaciones móviles (por ejemplo para abrir un link a Whatsapp o Instagram), es necesario declarar los permisos correspondientes a esta acción. La configuración en Android es:

```
<queries>
  <intent>
    <action android:name="android.intent.action.VIEW" />
    <data android:scheme="https" />
  </intent>
  <intent>
    <action android:name="android.intent.action.SENDTO" />
    <data android:scheme="mailto" />
  </intent>
</queries>
```

Y en iOS es:

```
<key>NSLocationTemporaryUsageDescriptionDictionary</key>
<dict>
  <key>YourPurposeKey</key>
  <string>The App requires temporary access to the device location.</string>
</dict>
```

4.4.11.3. Ruteo de URLs en Web

Para la aplicación web, es importante que cada sección se corresponda con una URL distinta. Por ejemplo, tenemos la URL `/cotizar` para el cotizador, `/login` para iniciar sesión o `/envios/{id_envio}` para ver el detalle de un envío. Sin embargo, Flutter provee un soporte limitado para incluir parámetros en las URL (como en el caso de la del detalle de un envío).

Por lo tanto, para alcanzar el resultado deseado, se utilizó una configuración que permite definir programáticamente un comportamiento personalizado a la hora de interpretar una URL. Esta se puede ver en el archivo `/lib/src/app.dart`, dentro del parámetro `onGenerateRoute` del componente `MaterialApp`. Para ello se utilizan expresiones regulares que descomponen la URL para obtener el identificador correspondiente y se lo pasa al constructor de la clase que se debe construir.

4.5. Backoffice

El backoffice es el panel de control del sistema, que expone a través de interfaces gráficas las herramientas de administración de la plataforma, fundamentalmente la gestión de

usuarios. Como tal, permite el acceso únicamente a los usuarios de categoría Administrador. Es, además, el único lugar desde donde pueden darse de alta conductores y otros administradores.

Lo acotado de su base de usuarios y lo específico de su función torna innecesario el desarrollo de una aplicación para celulares, considerando el esfuerzo extra que ello implicaría, por lo que se definió implementarlo como una aplicación web. Sí se consideró necesario que el backoffice sea responsive, y esté razonablemente optimizado para uso desde dispositivos móviles, puesto que debe garantizarse en cuanto sea posible la posibilidad de acceso de un administrador ante cualquier eventualidad.

4.5.1. Aspectos técnicos.

El backoffice se desarrolló como una aplicación web, eligiéndose el framework React como base para el mismo.

React es una biblioteca Javascript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página. Es mantenido por Facebook y la comunidad de software libre. Es, también, una de las herramientas de desarrollo de aplicaciones single page dominantes en el mercado, por lo que abundan documentación y ejemplos de uso para todos los casos que el backoffice contempla, tanto actualmente como en las potenciales extensiones futuras.

El equipo de desarrollo cuenta con experiencia previa en el uso de la misma, lo cual resultó el argumento determinante a la hora de elegirla por sobre alternativas como Vue o Angular.

Se optó por adoptar TypeScript durante el desarrollo a fin de evitar errores comunes relacionados a la falta de tipado explícito del lenguaje JavaScript.

Se utilizó la librería React para interfaces gráficas MUI, que ofrece decenas de componentes React útiles, todos alineados estéticamente según los lineamientos del lenguaje de diseño Material Design de Google. Esta elección se basó, nuevamente, en la experiencia previa del equipo en el uso de la librería, y en el hecho de que los componentes que ofrece son suficientes para implementar las características necesarias, ofreciendo siempre una experiencia limpia y visual intuitiva para el usuario.


4.5.2. Estructura del proyecto


Teniendo en cuenta lo mencionado anteriormente, la estructura de directorios del proyecto resulta:


Fletes31.Backoffice

src

 **Components:** Contiene los componentes React reutilizables en distintas páginas.

 **Models:** contiene los modelos y DTOs del proyecto.

 **Pages:** contiene las distintas páginas de la aplicación.

 **Helpers:** clases de apoyo, como *PagesList* o *QueryParameters*.

- 📁 **Stores:** contiene las clases asociadas a persistencia de datos local.
- index.tsx:** componentes raíz de React e inicialización de configuraciones globales.
- App.tsx:** define la navegación del sitio y configura librerías de alcance general.

4.5.3. Funcionalidades implementadas

El backoffice se encuentra protegido detrás de una pantalla de inicio de sesión que sólo habilitará el acceso a usuarios en la categoría de Administradores.



Figura 38: Pantalla de inicio de sesión.

Una vez dentro del backoffice, la interfaz presenta una barra de navegación izquierda, con acceso a las distintas secciones del programa, que puede colapsarse para ahorrar espacio en pantallas pequeñas. El menú también contiene el botón de cierre de sesión. El resto del espacio disponible pertenece a la página de navegación seleccionada.

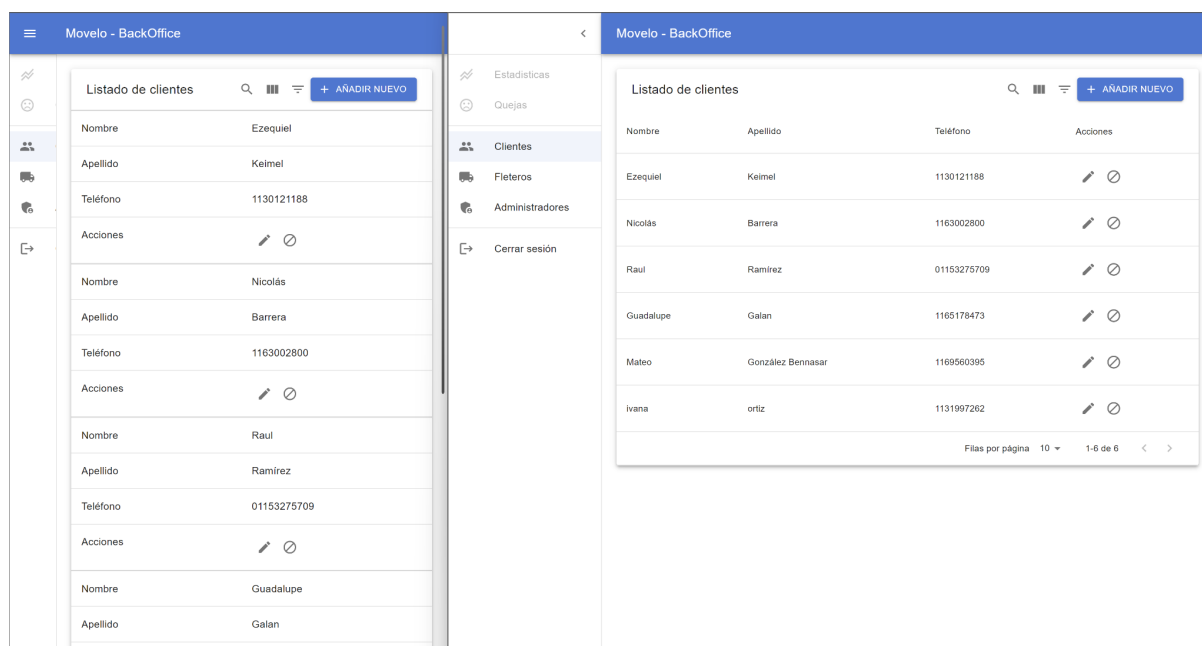


Figura 39: Pantalla de listado de Clientes. A la izquierda, vista móvil. A la derecha, vista de escritorio.

Los listados de Clientes, Conductores y Administradores son accesibles como páginas separadas. Cada uno es filtrable, ordenable y paginable, y cada registro de usuario, a su vez, posee botones de edición y de bloqueo. Este último permite prohibir a un usuario el acceso al sistema, tanto de manera permanente como durante un período expresado en días.

Existe también un botón de “añadir nuevo” que permite dar de alta un usuario. El formulario expone los campos correspondientes a datos comunes a todos los tipos de usuario, y también los específicos, que varían dependiendo del tipo de usuario que se dé de alta. De momento, son los conductores los que cuentan con una sección adicional, la de Vehículos del conductor, desde donde se agregan, modifican, o eliminan vehículos de cada uno.

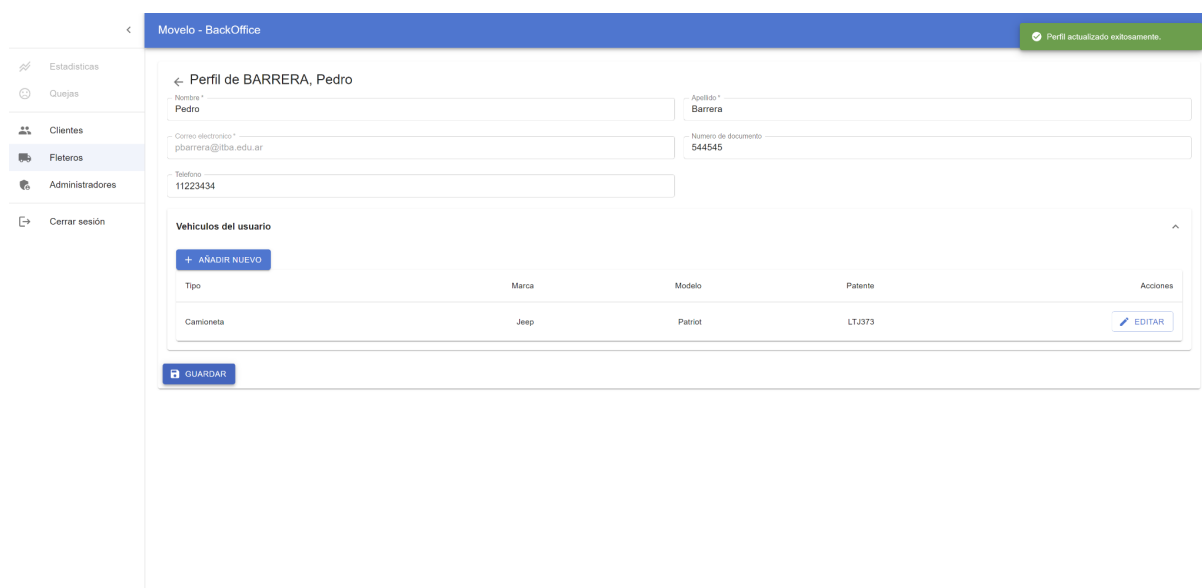


Figura 40: Pantalla de edición de un Conductor, con el listado de vehículos exclusivo de su tipo de usuario.

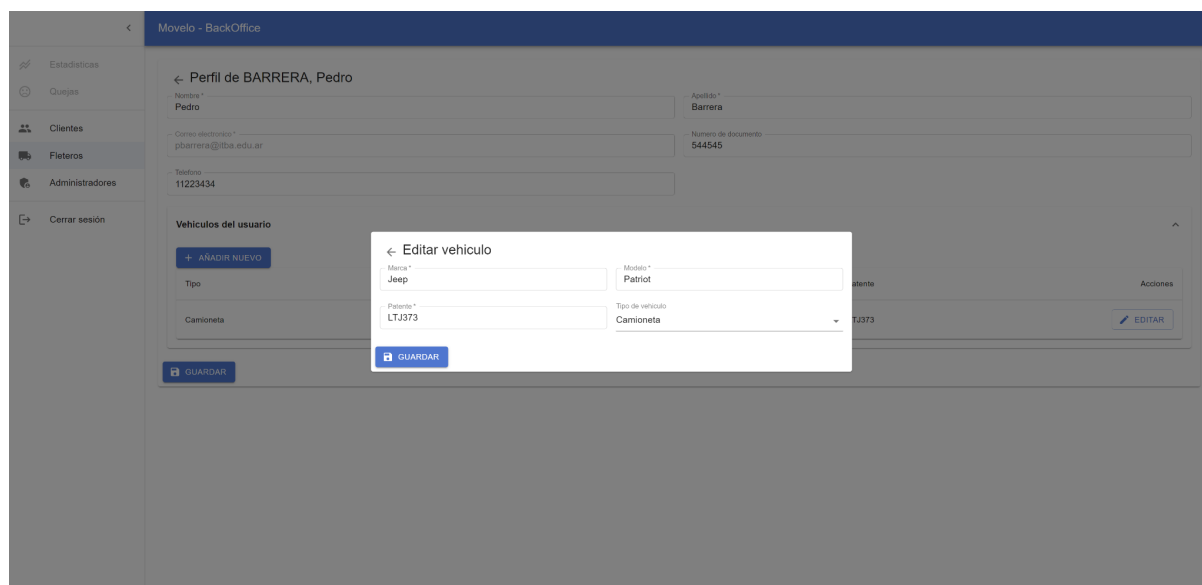


Figura 41: Formulario de edición de un vehículo.

El backoffice actualmente brinda únicamente herramientas de administración de usuarios. Ante la necesidad de concretar un producto funcional en el tiempo de desarrollo disponible, se optó por priorizar la aplicación para usuarios y conductores, y el backend, limitando el backoffice a las funcionalidades absolutamente indispensables. Destacan, entre los módulos de futura implementación, los siguientes:

- Control de quejas: el backoffice debe permitir a los administradores visualizar las valoraciones negativas que un conductor pueda recibir, para proceder a interceder con el objetivo de preservar la imagen de marca ante reiterados desmanejos. Se prevé el uso de la función de bloqueo, ya implementada, como herramienta de sanción ante potenciales infracciones.
- Estadísticas: la mejora continua del software y del servicio en general será el principal objetivo de cara a lograr crecer en el mercado. Para este propósito, conocer los patrones de uso del sistema, y sus métricas cuantificables, resulta esencial. El backoffice debe, eventualmente, ser capaz de generar reportes que expresen esta información.

5. Métricas de evaluación de resultados

El producto definitivo no ha sido lanzado a producción al momento de la redacción de este informe, por lo que no se cuentan aún con números concretos que permitan evaluar los resultados del trabajo. Sin embargo, se han definido las métricas con las que el impacto del futuro lanzamiento habrá de ser evaluado:

- Se comparará el **tiempo promedio desde que un cliente pide una cotización hasta que se le asigna un conductor**, antes y después del lanzamiento. Siendo que el sistema automático reemplazará la necesidad de comunicarse con la central administrativa para solicitar una cotización, que será ahora inmediata, el único eventual tiempo de espera debería darse entre la confirmación del pedido y la toma del mismo por parte de un conductor.
- Se comparará el **porcentaje de pedidos de cotización que se concretan en un envío**, antes y después del lanzamiento. Esta es la métrica más importante, puesto que pondrá de manifiesto si, como se espera, la nueva experiencia es valorada por los clientes en contraste con la anterior.
- Se comparará el **porcentaje de los usuarios que siguen requiriendo de comunicación con la central** frente a cualquier problema, para así determinar si la plataforma cumple con su ambición de automatización del proceso completo, y en qué medida.
- Entrevistas con usuarios (clientes y fleteros) para conocer su evaluación acerca de la mejora de la experiencia, sobre la transparencia y seguridad que la plataforma tiene que brindar.
- Las **reseñas en Play Store** brindarán una nueva fuente de feedback.

6. Manual de uso

El servicio de hosting utilizado para desplegar la aplicación es SmarterASP.NET[23]. El mismo ofrece facilidades específicas para aplicaciones desarrolladas en ASP.NET Core, lo que resultó especialmente conveniente para este proyecto. Además, este hosting se encuentra entre los recomendados por Microsoft para desplegar aplicaciones desarrolladas con esta tecnología [24].

Se contrató el plan .NET Advance, el cual además incluye un certificado SSL permitiendo mostrar el sitio como seguro desde un navegador web. Se configura además el dominio `www.` utilizando las reglas de redireccionamiento que provee el hosting, ya que el certificado SSL solo permite autenticar el dominio `movelo.com.ar` (sin el prefijo `www`). Cabe destacar que, por limitaciones del servicio de hosting, esta regla se debe reconfigurar cada vez que se realiza un deploy.

6.1. Herramientas de desarrollo

6.1.1. Visual Studio

Se utilizó esta herramienta para el desarrollo del backend, la cual provee un soporte muy completo para aplicaciones en ASP.NET.

Para ejecutar localmente la aplicación se debe abrir el proyecto con Visual Studio, hacer click derecho en el módulo *Fletes31.Backend.Api* y elegir la opción *Set as Startup Project*. Una vez hecho esto, ya se puede ejecutar el proyecto utilizando el botón verde con la leyenda *IIS Express* que se encuentra en la barra de herramientas del programa. A su izquierda se puede además elegir entre modo *Debug* o modo *Release*.

Cuando se ejecute en modo *Debug*, se utilizarán automáticamente la configuraciones que se listen en el archivo *appsettings.Development.json*. En modo *Release*, se utilizará aquellas que figuren en *appsettings.json*.

6.1.2. Android Studio

Esta herramienta se utilizó en el desarrollo del frontend, que es hasta el momento la más avanzada para utilizar con el framework *Flutter*. Para utilizarla, basta con importar el proyecto, y Android Studio se encargará de escanear el archivo *pubspec.yaml* y descargar las dependencias que figuren en el mismo. Para ejecutarlo, se debe seleccionar la opción *Run* 'main.dart'.

Además, en la barra de herramientas se muestra un menú desplegable que permite elegir para qué plataforma se desea compilar y ejecutar el proyecto (por ejemplo web o Android).

6.2. Pasos para realizar un deploy

1. En primer lugar se debe compilar el frontend (web) para luego integrarlo con el backend. Para ello, en la raíz del proyecto frontend ejecutar el siguiente comando:

```
$> flutter build web --web-renderer canvaskit
```

2. El resultado de la compilación es almacenado en la carpeta */build/web/*.
3. Copiar el contenido de esta carpeta en */Fletes31.Backend.Api/wwwroot/* del backend. Esto permite contener ambos proyectos en uno solo, y así deployar la app completa en una sola instancia.
4. Ingresar en el sitio *smarterasp.net*.
5. Dirigirse a la opción *Customer Login*.
6. Introducir las credenciales de acceso de MoveLo.
7. Seleccionar el botón *Control Panel* del proyecto con nombre *moveLo-001*.
8. En la lista de *websites*, seleccionar el ícono con tres puntos y elegir la opción *VS webdeploy*.
9. Verificar que en campo *Status* figure *On*. En caso contrario, encenderlo.
10. Presionar el botón *Get Publish Settings*.

11. Se descargará un archivo con extensión *.PublishSettings*.
12. Desde Visual Studio, hacer click derecho sobre el módulo *Fletes31.Backend.Api* y seleccionar la opción *Publish*.
13. En la nueva pantalla que se muestra, elegir la opción *New* y luego *Import Profile*, y seleccionar el archivo descargado anteriormente.
14. Al aceptar, se mostrará un cartel donde se piden las credenciales de acceso al hosting.
15. Es posible que luego del paso anterior, se presente un error con el certificado. En ese caso, hacer click en la opción *Show all*, elegir la solapa *Connection* y luego *Validate Connection*. Se deberán introducir nuevamente las credenciales para el deploy.
16. Finalmente hacer click en el botón *Publish*, el cual publicará todos los cambios realizados al servicio y lo desplegará en el servidor.

6.3. Configuración del dominio **movelo.com.ar**

Se registró y delegó el dominio *movelo.com.ar* a través de la autoridad de registro de dominios de la República Argentina, NIC.ar, para apuntar a los servidores DNS autoritativos del hosting elegido para alojar el sistema. Luego, se configuró dicho servicio para resolver los nombres *www.movelo.com.ar* y *movelo.com.ar* hacia la dirección IP del sitio web. Finalmente, se instaló un certificado SSL para validar la identidad del sitio frente a los navegadores.

7. Desarrollos futuros

Debido al establecimiento de fechas límite para el desarrollo, resultó importante priorizar funcionalidades esenciales con el fin de garantizar la usabilidad del producto lanzado desde su primera versión pública. En ese proceso, varias extensiones y funcionalidades quedaron pendientes y forman parte de futuras etapas de desarrollo, a encararse como continuidad del proceso actual.

Se listan, a continuación, en el orden de prioridad que surge del análisis funcional llevado a cabo inicialmente:

1. Lanzamiento de la aplicación móvil en iOS para alcanzar a los usuarios finales de esta plataforma (la falta de equipos Mac, esenciales para el desarrollo en la plataforma de Apple, impidió el avance en este frente).
2. Sistema de reseñas de viaje y quejas de usuario: los administradores deben tener herramientas para detectar conflictos en el servicio, asociados tanto a la plataforma como a quienes trabajan a través de ella, que impacten en la imagen de marca.
3. Soportar el escenario de un viaje que incluya paradas intermedias entre el origen y el destino final, con cargas y descargas en cada una.

4. Permitir a los administradores emitir códigos de descuento desde el backoffice, permitiendo de esta forma compensar a un cliente en caso de problemas, y articular con el área de marketing para generar incentivos de adopción de la plataforma.
5. Generación de informes estadísticos que permitan estudiar el uso de la plataforma por parte de los usuarios, información invaluable para tomar cualquier tipo de medida de negocio.
6. Adoptar un framework de comunicación en tiempo real, como SignalR, en detrimento de la actual implementación *long polling* (consultar al servidor continuamente sobre cambios en lugar de mantener una conexión activa y que solo envíe información cuando sea necesario), donde sea conveniente, por ejemplo, en las pantalla de chat o de estado actual del pedido.
7. Permitir edición avanzada del perfil de los usuarios, soportando, por ejemplo, avatares que acentúen la imagen social del servicio.
8. Lanzar una aplicación nativa en Windows, Mac y Linux, aprovechando las facilidades que el framework de Flutter ofrece.
9. Manejar los pagos por dentro de la plataforma, a través de algún proveedor de servicios financieros como MercadoPago. Aunque se lo considera un próximo paso lógico en el desarrollo, conlleva dificultades legales y administrativas que impiden avanzar de momento.

8. Conclusiones

Las primeras pruebas parecen dar fe de que el desarrollo, desde lo meramente tecnológico, cumple con su principal objetivo: el de reimaginar la experiencia de usuario para reducir a un mínimo los tiempos muertos, además de aportar a la confianza percibida por los clientes brindando mecanismos de control como el seguimiento en tiempo real de los pedidos.

Estos dos puntos son centrales puesto que constituyen la ventaja más manifiesta a favor de la competencia, y que ahora se nivela, permitiendo al proyecto competir desde la profesionalidad de sus recursos humanos, antes limitados por el lastre que implicaba toda la burocracia previa al inicio del traslado mismo, que en muchos casos impedía cualquier tipo de concreción ante la posibilidad del cliente de recurrir a una alternativa en segundos y sin complicaciones.

Las pruebas en dispositivos testigo permiten también reafirmar el acierto en lo que la elección de Flutter como stack de tecnología para el desarrollo del frontend respecta: tanto en la web como en Android, la aplicación responde con fluidez e implementa apropiadamente el diseño conceptualizado por el equipo de imagen de marca. Además, su flexibilidad hace viable, incluso en el corto plazo, el desembarco de la aplicación en todas las plataformas de uso masivo, maximizando la llegada a potenciales clientes. Todo esto, en un tiempo de desarrollo acotado que no dio lugar a priorizar la optimización fina del código.

Con estas barreras superadas, la eficiencia del negocio en sí pasa a primer plano, por lo que el éxito del proyecto se verá determinado por el progreso en cuatro ejes principales:

- Aumento de la base de conocimiento, y luego de uso, de la plataforma: a través de estrategias de marketing, debe el servicio darse a conocer para pasar a estar en el radar de usuarios de la competencia y de primerizos que lo requieran.
- Profesionalización y seguimiento permanente de los recursos humanos para mejora continua de todas las interacciones con los clientes, ayudando así a reforzar la imagen positiva de la marca en usuarios que ya han hecho uso del servicio y, en consecuencia, promoviendo la retención de clientes y la llegada de nuevos por medio del boca a boca.
- Optimización y continua mejora del software para pulir cualquier aspereza y ofrecer funcionalidad y experiencia de uso mínimamente a la altura de lo mejor que pueda ofrecer la competencia.
- Continuo análisis y optimización de los recursos económicos, que permitan minimizar los costos del emprendimiento para así ofrecer precios competitivos que permitan ganar mercado.

Finalmente, el equipo considera que los resultados obtenidos, considerando lo acotado de los recursos humanos, temporales y económicos, resultan preliminarmente satisfactorios y se alinean con las expectativas y los objetivos propuestos en un inicio. Queda pendiente una evaluación formal de los mismos, una vez se recolecten datos suficientes, mediante las métricas propuestas en la sección correspondiente.

9. Referencias bibliográficas

- [1] Impulso Negocios, *La pandemia aumentó la cantidad de mudanzas en Latinoamérica*, impulsonegocios.com, [Online; accedido el 27 de marzo de 2022]
- [2] iProUP, *Las apps de movilidad, en modo "todos contra todos"*, iproup.com, [Online; accedido el 27 de marzo de 2022]
- [3] Misiones Online, *Coronavirus: durante la cuarentena en Argentina, las compras online crecieron un 52%*, misionesonline.net, [Online; accedido el 27 de marzo de 2022]
- [4] infoNegocios, *Rappi ya tiene el liderazgo de la categoría en CABA*, infonegocios.info, [Online; accedido el 27 de marzo de 2022]
- [5] La Nación, *PedidosYa: cuáles son los planes de expansión del unicornio uruguayo y la polémica por el modelo laboral*, lanacion.com.ar, [Online; accedido el 27 de marzo de 2022]
- [6] Fletalo Sitio Oficial, fletalo.com.ar
- [7] Kinto Sitio Oficial, kinto-mobility.com.ar
- [8] Awto Sitio Oficial, awto.com
- [9] MyKeego Sitio Oficial, mykeego.com
- [10] Verga Hnos Sitio Oficial, vergahnos.com.ar
- [11] ASP.NET Official Site, dotnet.microsoft.com
- [12] Altwater, A., *What is N-Tier Architecture?*, stackify.com, [Online; accedido el 14 de abril de 2022]
- [13] Panyushkin, D., *Generic Repository Pattern in Entity Framework*, codingsight.com, [Online; accedido el 14 de abril de 2022]
- [14] Flutter Official Site, flutter.dev
- [15] Dart Official Site, dart.dev
- [16] React Native Official Site, reactnative.dev
- [17] Gallagher, N., *React Native for Web*, necolas.github.io, [Online; accedido el 27 de marzo de 2022]
- [18] Suri, S., *Architect your Flutter project using BLOC pattern*, medium.com, [Online; accedido el 27 de marzo de 2022]
- [19] Ribeiro, W., *Why use RxDart and how we can use with BLoC Pattern in Flutter?*, medium.com, [Online; accedido el 27 de marzo de 2022]
- [20] RxDart Official Site, pub.dev/packages/rxdart
- [21] Retrofit Official Site, pub.dev/packages/retrofit

- [22] JsonSerializerizable Official Site, pub.dev/packages/json_serializable
- [23] SmarterASP.NET Official Site, smarterasp.net
- [24] Microsoft, *ASP.NET Hosting*, dotnet.microsoft.com, [Online; accedido el 27 de marzo de 2022]