

Proyecto Final: Aplicación Móvil para Conocer la Ubicación de los Colectivos y Estimar Tiempos de Espera

Universidad Nacional del Centro de la Provincia de
Buenos Aires
Facultad de Ciencias Exactas
Ingeniería en Sistemas



Nicolás Miccio Palermo - nicolasmicciopalermo@gmail.com

Directores: Antonela Tommasel - Marcelo Armentano

Capítulo 1: Introducción.....	4
1.1 Motivación.....	4
1.2 Objetivos.....	5
1.3 Organización del Informe.....	5
Capítulo 2: Análisis y Marco Teórico.....	7
2.1 Análisis de Apps Existentes.....	7
2.2 APIs de Geolocalización.....	10
2.3 Lenguajes de Programación y Frameworks.....	15
2.4 Base de Datos.....	23
2.5 Cálculo y Actualización de Rutas y Ubicaciones.....	29
2.6 Técnicas de Predicción de Tiempos de Traslado.....	31
1. Regresión Lineal.....	31
2. Modelos ARIMA (AutoRegressive Integrated Moving Average).....	32
3. Random Forest.....	33
4. Redes Neuronales de Memoria a Largo Plazo (LSTM).....	34
5. Gated Recurrent Units (GRU).....	34
6. Modelo HMM (Hidden Markov Model).....	35
7. Redes Neuronales Recurrentes (RNN).....	36
8. K-Nearest Neighbors (K-NN).....	37
9. Redes Bayesianas.....	37
Resumen.....	38
Capítulo 3: Análisis y Procesamiento de los Datos.....	42
3.1 Cobertura Temporal.....	43
3.2 Distribución por Línea y Unidad.....	44
3.3 Nulos y Duplicados.....	46
3.4 Distribución de Velocidades.....	47
3.5 Filtrado de Datos por Recorrido.....	48
Capítulo 4: Diseño e Implementación de los Modelos de Predicción.....	51
4.1 Modelos de Regresión Lineal.....	52
4.2 Modelo ARIMA.....	56
4.3 Modelo LSTM.....	63
4.4 Modelo GRU.....	68
4.5 Comparación de Resultados.....	71
Capítulo 5: Diseño e Implementación de la Aplicación Móvil.....	85
5.1 Requerimientos Funcionales y No Funcionales.....	86
5.1.1 Requerimientos Funcionales.....	86
5.1.2 Requerimientos No Funcionales.....	87
5.1.3 Priorización de Requerimientos.....	88
5.2 Decisiones Tecnológicas.....	88
5.3 Procesamiento de Paradas y Recorridos.....	91
5.3.1 Procesamiento y Ordenamiento de Paradas.....	92

5.3.2 Corrección Manual de Superposiciones en los Recorridos.....	92
5.3.3 Almacenamiento en la Base de Datos.....	93
5.4 Experiencia e Interfaz de Usuario.....	97
5.4.1 Diseño de la Experiencia de Usuario (UX).....	97
5.4.2 Diseño de la Interfaz de Usuario (UI).....	98
5.4.3 Interfaces de la Aplicación.....	99
5.5 Implementación de la Aplicación.....	100
5.5.1 Arquitectura de la Aplicación.....	100
5.5.2 Desarrollo del Servidor.....	102
5.5.3 Organización del Código.....	104
5.5.4 Desarrollo del Cliente.....	106
Capítulo 6: Conclusión.....	116

Capítulo 1: Introducción

En la ciudad de Tandil, el transporte público de colectivos juega un papel fundamental siendo, por el momento, el único medio de transporte disponible para todos los ciudadanos. Este sistema abarca a usuarios de todos los sectores sociales y de todas las edades, convirtiéndose en un pilar esencial para la movilidad dentro de la ciudad. Para ello cuenta con diversas líneas de colectivos y unidades de las mismas que recorren diferentes puntos de la ciudad, permitiendo una conectividad rápida y eficiente, contemplando que los distintos recorridos cubran la mayor parte de la ciudad. Por ello, es indispensable que el servicio de colectivos funcione de manera óptima, garantizando una experiencia satisfactoria para todos los usuarios.

La tecnología se ha vuelto una aliada esencial en la mejora de la experiencia de transporte público, ofreciendo herramientas que simplifiquen y hagan más eficiente su uso. En consecuencia con esto, el presente trabajo se enfoca en proporcionar al usuario una herramienta más para ayudarlo en su día a día con el uso del transporte público.

1.1 Motivación

Actualmente la ciudad de Tandil no dispone de una aplicación móvil relacionada con el transporte público, a pesar de la reciente incorporación del sistema SUBE, el mismo no cuenta por el momento con un mecanismo de rastreo de colectivos, en contraparte con su antecesor SUMO que sí ofrecía una aplicación, aunque la misma poseía ciertas limitaciones. Si bien era posible visualizar la ubicación de las unidades en tiempo real y los recorridos completos de cada línea, no había forma de calcular rutas óptimas o vislumbrar las demoras asociadas y tiempos estimados de traslado. Esto refleja una necesidad creciente por parte de los usuarios de tener una herramienta que mejore su experiencia brindando mayor cantidad y calidad de información para facilitar el uso del transporte y ofrecer una experiencia más amena. Teniendo en cuenta todo esto, y observando que no hay soluciones que logren consolidar una conexión eficiente entre el usuario y el sistema de colectivos, el proyecto que aquí se presenta busca desarrollar una nueva aplicación móvil, llamada BusNow, que integre las características presentes y faltantes mencionadas con anterioridad. Esta propuesta pretende mejorar la experiencia del servicio de transporte público basándose en las distintas aplicaciones presentes en el mercado e incorporando las funcionalidades que realmente faciliten el uso cotidiano del servicio para las personas de Tandil.

1.2 Objetivos

El principal objetivo de este proyecto es definir un modelo basado en técnicas de *Machine Learning* que permita estimar los tiempos de espera y predecir el tiempo de desplazamiento entre dos puntos específicos utilizando el sistema de colectivos de Tandil. A partir de este modelo será posible recomendar las rutas más eficientes basadas en un par de coordenadas de origen y destino, considerando una variedad de parámetros de entrada. Esto permitirá generar estimaciones precisas, incluso teniendo en cuenta diversos factores que afectan al tráfico y al comportamiento de las unidades como pueden ser las horas o días particulares. El sistema propuesto no solo tomará en cuenta elementos del transporte público, como la línea y el recorrido, sino también variables externas como la hora del día, la proximidad al centro de la ciudad, el día de la semana, entre otros factores relevantes.

Dado que los colectivos son utilizados por una gran parte de la población diariamente, se pondrá especial énfasis en utilizar datos actualizados y fiables, con el fin de generar un servicio que inspire confianza a los usuarios. A través de una interfaz sencilla y amigable, se ofrecerá una visión clara sobre el estado del sistema de transporte para que el usuario pueda mantenerse lo más informado posible.

En cuanto a la arquitectura del sistema, se buscará desarrollar una solución accesible para la mayor cantidad de dispositivos móviles posibles, independientemente de sus características técnicas. Asimismo, la interfaz de usuario será diseñada de forma simple e intuitiva, garantizando que la experiencia de uso sea agradable y atractiva para los usuarios, con el objetivo de fomentar su adopción por parte de nuevos usuarios.

1.3 Organización del Informe

Este informe está estructurado para detallar los diversos aspectos del trabajo desarrollado. A continuación, se ofrece una breve descripción de los capítulos que componen este documento:

- **Capítulo 2:** Se presenta la revisión bibliográfica, el análisis de herramientas y conceptos relacionados con los sistemas de transporte público, así como los fundamentos teóricos utilizados en la resolución del problema.
- **Capítulo 3:** Se hace un análisis exhaustivo de los datos utilizados para los modelos de predicción, comenzando por su origen, siguiendo por su interpretación y finalizando con el filtrado correspondiente para depurar la información presente en el dataset.

- **Capítulo 4:** Este capítulo aborda el desarrollo y evaluación de los diferentes modelos empleados para la predicción de los tiempos de traslado de los colectivos. Para cada uno se detallarán todas las métricas utilizadas para medir, interpretar y calificar el rendimiento de los mismos, y de esta forma fundamentar las decisiones tomadas.
- **Capítulo 5:** Aquí se desarrolla el diseño y la implementación de la aplicación móvil. Se comenzará detallando los requerimientos funcionales y no funcionales, para seguir con las decisiones tecnológicas y el procesamiento de los datos utilizados. Se continuará con el diseño de la interfaz gráfica, para finalizar con los aspectos más técnicos de la implementación de la app.
- **Capítulo 6:** Se incluyen las conclusiones del trabajo, señalando tanto los aspectos positivos como los aspectos que pueden ser mejorados. Además, se realizará una evaluación personal del trabajo desarrollado teniendo en cuenta debilidades y fortalezas en la educación recibida a lo largo de la carrera, conocimientos, herramientas y otras habilidades adquiridas.

Capítulo 2: Análisis y Marco Teórico

En este capítulo se realizará un análisis detallado de las decisiones adoptadas durante el proceso de desarrollo de la aplicación propuesta. Se abordarán las dificultades encontradas a lo largo del proyecto y las soluciones implementadas para superarlas.

Se comenzará con un panorama de las aplicaciones existentes actualmente que ofrecen funcionalidades similares a las que se han integrado en esta propuesta. Se detallarán sus características principales, así como las diferencias clave que distinguen la herramienta desarrollada en este trabajo de las opciones ya disponibles. Estas aplicaciones no solo están orientadas al ámbito local de la ciudad de Tandil, sino que son utilizadas en diversas regiones nacionales e internacionales, y se encuentran accesibles en múltiples plataformas.

En secciones posteriores, se presentará el análisis y la selección de las tecnologías utilizadas en el desarrollo de la aplicación, cubriendo aspectos clave como las herramientas empleadas para gestionar las funcionalidades de geolocalización, así como los lenguajes de programación y frameworks elegidos para la implementación del sistema. Además, se discutirá la elección de la base de datos y la integración de un servidor que permita manejar tareas complejas o que impliquen riesgos para la seguridad de los datos.

Por último, se hará un relevamiento bibliográfico buscando comprender las distintas técnicas utilizadas en la estimación de tiempos de traslado en sistemas de transporte. Este análisis permitirá identificar enfoques tradicionales y modernos, evaluando sus ventajas y limitaciones en distintos escenarios, siendo de suma importancia para seleccionar los métodos más adecuados para su posterior aplicación en el contexto específico del proyecto.

2.1 Análisis de Apps Existentes

En la actualidad, existen diversas aplicaciones móviles dedicadas a la movilidad urbana y al transporte público, que ofrecen una variedad de funcionalidades para mejorar la experiencia de los usuarios. A continuación, se describen algunas de las aplicaciones más utilizadas, que cuentan con características que sirvieron como referencia para el desarrollo de la herramienta propuesta.

Google Maps es una de las aplicaciones más conocidas y utilizadas en el mundo. Su principal funcionalidad es la visualización del mapa, donde los usuarios pueden observar su ubicación actual y explorar lugares cercanos. A través de la sección "Explorar", los usuarios pueden descubrir sitios de interés, y con la opción "Guardados", pueden marcar y acceder a ubicaciones previamente seleccionadas.

La interfaz de búsqueda es intuitiva y fácil de usar, lo que hace que muchos usuarios estén familiarizados con su funcionamiento. Cuando se selecciona un destino, Google Maps ofrece una estimación del tiempo de llegada basado en el medio de transporte elegido, lo cual es útil para planificar el viaje. La simplicidad y la familiaridad de la interfaz de Google Maps lo convierten en un referente en cuanto a la navegación y la planificación de rutas.

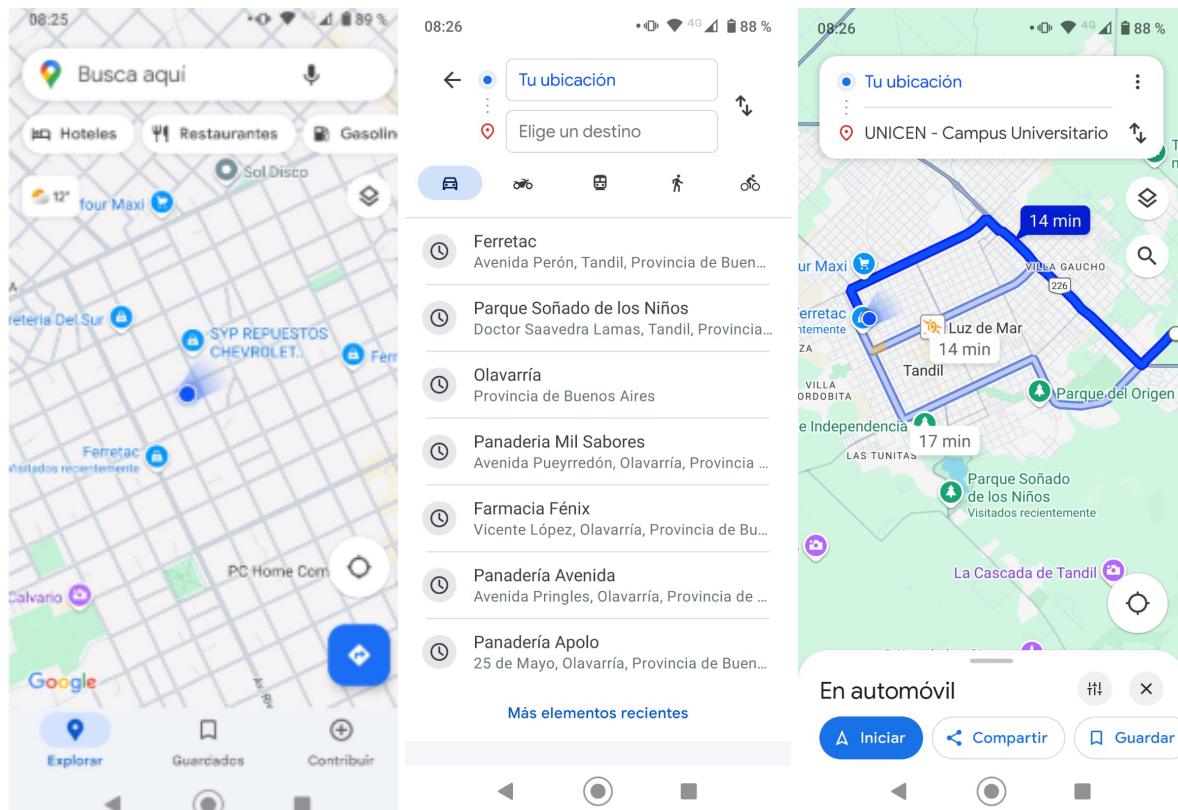


Figura 1 Principales Interfaces de la Aplicación de Google Maps.

Moovit es otra aplicación destacada que proporciona información en tiempo real sobre el transporte público, incluyendo colectivos, trenes y subtes. Su interfaz se asemeja a la de Google Maps, pero incorpora funcionalidades adicionales, como la posibilidad de que los usuarios reporten incidencias en tiempo real, lo que enriquece la experiencia al permitir a la comunidad colaborar para mejorar la información. Además de sugerir rutas alternativas, Moovit ofrece detalles sobre los recorridos a pie y en transporte público, lo que facilita la planificación del viaje. Un aspecto destacable de Moovit es la inclusión de las paradas de colectivos, lo cual ayuda a los usuarios a saber exactamente dónde bajarse, lo que resulta muy útil para optimizar el tiempo de desplazamiento.

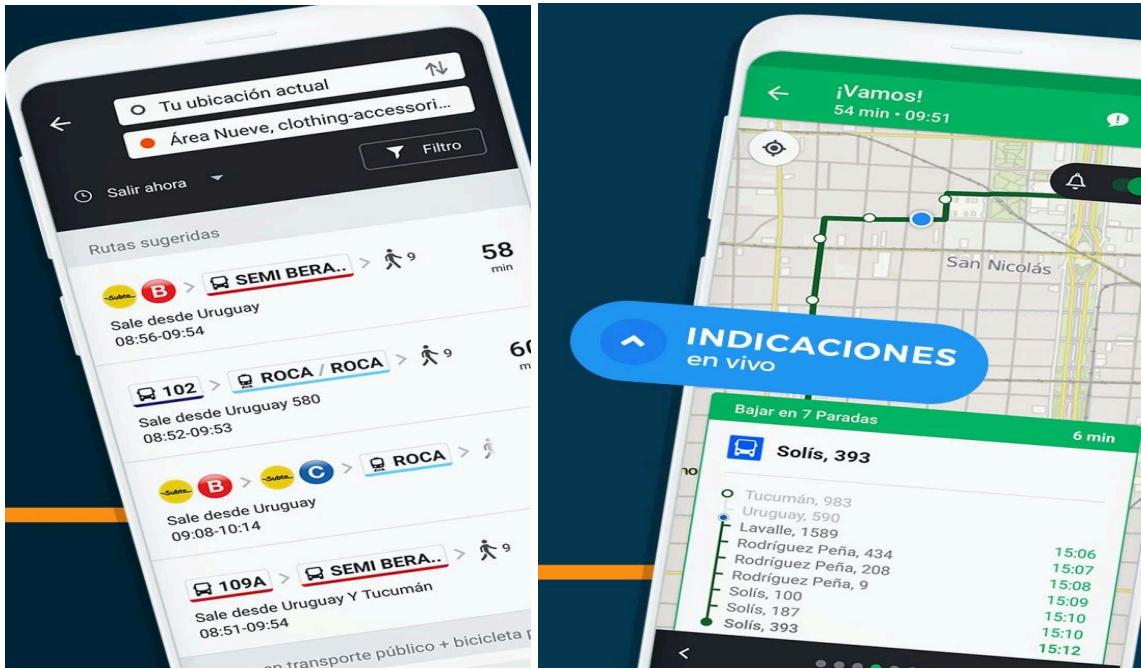


Figura 2 Elección y Seguimiento de Recorridos en Moovit.

Por otro lado, **Waze** está principalmente orientada a la navegación para automóviles, y se basa en la información proporcionada por los usuarios para generar alertas sobre tráfico, accidentes y rutas más rápidas. Aunque Waze es más conocida por su enfoque en la conducción, también ofrece funciones como navegación por GPS, actualizaciones en tiempo real sobre el tráfico, y alertas de seguridad (como información sobre accidentes o controles policiales). Aunque no es una herramienta dedicada al transporte público, su sistema de notificaciones y alertas resulta de interés para la implementación de funciones similares en una aplicación para colectivos.

Otras aplicaciones como **Uber** y **PedidosYa** también brindan al usuario estimaciones precisas sobre el tiempo de espera y la ubicación de los conductores en tiempo real. Estas aplicaciones permiten un seguimiento constante de los vehículos y envían notificaciones al usuario sobre su ubicación, lo cual es útil para una aplicación de colectivos. La posibilidad de integrar un sistema similar, que permita a los usuarios seguir el recorrido de los colectivos y recibir notificaciones sobre su llegada, podría mejorar significativamente la experiencia del usuario.

En cuanto a la aplicación **Transit**, ofrece una funcionalidad similar a la de las aplicaciones mencionadas anteriormente, pero con un añadido interesante: notifica al usuario cuándo debe salir de su ubicación o cuándo debe apurarse para no perder el transporte. Esta funcionalidad resulta particularmente útil en una aplicación que estime los tiempos de espera y ayude a los usuarios a optimizar su tiempo, asegurando que lleguen a las paradas de colectivo en el momento adecuado.



Figura 3 Seguimiento de Recorridos y Notificaciones en Transit.

En general, muchas aplicaciones de transporte permiten al usuario guardar lugares de interés o visualizar rutas previamente elegidas, lo que requiere una cuenta de usuario. Además, existen muchas aplicaciones similares a las ya mencionadas, en diferentes partes del mundo, que ofrecen funcionalidades casi idénticas. En este sentido, la incorporación de un apartado en la aplicación que muestre los recorridos de los colectivos mediante imágenes y permita filtrarlos según la línea sería una característica adicional que enriquecería la experiencia del usuario, permitiendo una visualización clara y fácil de las rutas.

2.2 APIs de Geolocalización

Un Sistema de Información Geográfica (GIS, por sus siglas en inglés) es una tecnología que permite capturar, almacenar, analizar y visualizar datos espaciales para representar el mundo real en mapas digitales. Su principal función es combinar información geográfica con datos tabulares, lo que posibilita un análisis profundo de diferentes variables en función de su ubicación geoespacial. Gracias a su capacidad para manejar grandes volúmenes de datos y su versatilidad, los GIS han encontrado aplicación en una amplia variedad de campos, desempeñando un papel crucial en la planificación, gestión y toma de decisiones en múltiples sectores.

Entre sus principales usos se encuentra la cartografía y la navegación, permitiendo la creación de mapas digitales interactivos que facilitan la localización de puntos de interés, la planificación de rutas y el análisis del terreno. Un claro ejemplo de ello son aplicaciones como Google Maps o Waze, que utilizan GIS para calcular rutas

óptimas y proporcionar información en tiempo real sobre el tráfico y otros factores relevantes. En el ámbito del transporte, los GIS juegan un papel fundamental en la gestión de flotas y logística, optimizando las rutas de distribución y reduciendo costos operativos. Además, son herramientas clave en la planificación urbana y el medio ambiente, ya que permiten analizar el uso del suelo, evaluar el impacto ambiental de proyectos de infraestructura y monitorear fenómenos como la deforestación o el cambio climático.

Con la evolución de la tecnología móvil, los GIS han sido integrados en dispositivos como teléfonos inteligentes y tabletas, lo que ha permitido su uso en tiempo real por parte de profesionales y ciudadanos en general. Una de las aplicaciones más comunes es la navegación y localización mediante mapas digitales. El monitoreo y seguimiento en tiempo real es otro aspecto clave, usándose en aplicaciones de logística y transporte, como Uber o PedidosYa, dependen de estos sistemas para rastrear vehículos y entregas en tiempo real, proporcionando a los usuarios estimaciones de tiempo de llegada y optimizando la asignación de recursos.

Los GISs han revolucionado la manera en que se gestionan y analizan los datos espaciales, proporcionando herramientas poderosas para una gran variedad de industrias. Su integración con dispositivos móviles ha expandido aún más sus posibilidades, facilitando la toma de decisiones basada en información georreferenciada en tiempo real. Gracias a su continua evolución, estos sistemas seguirán desempeñando un papel clave en la mejora de la eficiencia y la precisión en diversos campos, desde la navegación y el transporte hasta la planificación urbana, la gestión ambiental y la seguridad.

Teniendo todo esto en cuenta, una de las principales funcionalidades de la aplicación es la geolocalización en tiempo real, por lo que a continuación se analizan las principales APIs de mapas y geolocalización disponibles para la integración con la app. A continuación, se presentan algunas de las opciones más destacadas y su adecuación para el proyecto.

1. Google Maps¹

Google Maps es la API más utilizada y conocida para integrar mapas y geolocalización en aplicaciones móviles, siendo la opción más confiable y accesible para la mayoría de los desarrolladores. Una de las principales ventajas de Google Maps es que ofrece una amplia gama de funcionalidades listas para usar, lo que permite a los desarrolladores integrar rápidamente mapas interactivos, rutas, geocodificación, Street View, vistas en 3D y otras características avanzadas sin necesidad de crear soluciones desde cero. Su documentación es muy extensa y está bien organizada, con ejemplos claros que facilitan la implementación en aplicaciones móviles. Además, Google Maps cuenta con SDKs específicos tanto

¹ <https://developers.google.com/maps/documentation?hl=es-419>

para Android como para iOS, lo que optimiza la integración y el rendimiento en cada plataforma, permitiendo una experiencia más fluida para el usuario final. La cobertura global de Google Maps es otra de sus grandes ventajas, ya que ofrece mapas detallados de prácticamente cualquier región del mundo, lo que lo convierte en la opción más confiable para aplicaciones que dependen de una alta precisión geográfica. A su vez, Google Maps proporciona información en tiempo real sobre tráfico, condiciones de la carretera, estimación de tiempo de viaje y mucho más, lo cual es fundamental para aplicaciones de movilidad, transporte o logística. Su API también está diseñada para manejar grandes volúmenes de tráfico y usuarios, lo que lo convierte en una opción robusta para aplicaciones con grandes bases de usuarios. Sin embargo, Google Maps también presenta algunas desventajas. Aunque la plataforma es gratuita en pequeña escala, puede volverse costosa a medida que aumentan las vistas, las solicitudes de geocodificación o el uso de funcionalidades avanzadas. Para aplicaciones con una base de usuarios considerable o que requieren un uso intensivo de la API, los costos pueden aumentar rápidamente, lo que puede hacer que esta solución sea menos atractiva a largo plazo, especialmente para desarrolladores con presupuestos limitados. Además, como la plataforma depende completamente de Google, los desarrolladores están sujetos a sus términos y condiciones, así como a posibles cambios en las políticas o precios, lo que introduce un nivel de incertidumbre en cuanto a costos futuros. Aunque Google Maps ofrece una gran cantidad de funcionalidades listas para usar, presenta algunas limitaciones en términos de personalización del diseño de los mapas. Si se requiere un nivel muy alto de personalización en la apariencia o el estilo del mapa, otras alternativas como Mapbox pueden ser más adecuadas.

2. OpenStreetMap (OSM) con Leaflet²

OpenStreetMap es una alternativa libre y colaborativa a Google Maps, desarrollada por una comunidad global de usuarios que contribuyen a mantener los mapas actualizados y detallados. Leaflet es una biblioteca JavaScript que se utiliza para manejar los mapas y marcadores generados a partir de OpenStreetMap, ofreciendo una solución ligera y fácil de integrar en aplicaciones móviles y web. Una de las principales ventajas de OpenStreetMap con Leaflet es que se trata de una opción completamente gratuita y open-source, lo que significa que no hay restricciones de pago ni de uso, lo que la hace ideal para aplicaciones que buscan minimizar los costos de implementación y mantenimiento. Además, los desarrolladores tienen total control sobre el estilo del mapa, lo que les permite personalizarlo según sus necesidades, sin las limitaciones que pueden presentar otras plataformas comerciales. OpenStreetMap también facilita la integración con datos de geolocalización, ofreciendo funcionalidades de ubicación y visualización de puntos de interés de manera sencilla. Sin embargo, tiene algunas desventajas importantes. Aunque el proyecto está en constante crecimiento, la precisión de los mapas y la

² <https://leafletjs.com>

cantidad de detalles pueden ser inferiores en algunas regiones en comparación con Google Maps, especialmente en áreas rurales o menos desarrolladas. Aunque Tandil está bien mapeado, este puede no ser el caso en otras zonas. OpenStreetMap no cuenta con un SDK oficial para Android o iOS, por lo que se deben usar bibliotecas o herramientas adicionales para integrar la funcionalidad de mapas y geolocalización en las aplicaciones móviles, lo que puede incrementar la complejidad del proyecto. Esto implica una mayor carga en cuanto a la configuración y mantenimiento, lo que puede ser un inconveniente para equipos que buscan una solución más sencilla y rápida.

3. Mapbox³

Mapbox es una plataforma avanzada para la creación de mapas interactivos y personalizados que proporciona una API muy flexible, ideal para aplicaciones móviles que requieren una integración sofisticada de mapas con soporte para GPS y geolocalización en tiempo real. Una de las principales ventajas de Mapbox es la posibilidad de personalizar los mapas de manera avanzada, lo que permite crear visualizaciones únicas y adaptadas a las necesidades específicas de cada aplicación, sin las restricciones de diseño que pueden tener otras alternativas como Google Maps. Además, Mapbox ofrece un rendimiento superior y un menor costo en comparación con Google Maps para proyectos pequeños o de mediana escala, ya que proporciona 50,000 vistas gratuitas por mes, lo que puede cubrir una gran cantidad de usuarios en etapas iniciales del desarrollo. También permite la integración de funciones para el manejo de ubicaciones en tiempo real, lo cual es crucial para aplicaciones que dependen de la geolocalización y la actualización constante de la ubicación de los usuarios. La plataforma ofrece SDKs oficiales para Android e iOS, lo que simplifica la integración de mapas y geolocalización en las aplicaciones móviles, además de proporcionar un buen soporte para la gestión de mapas vectoriales, lo que permite un renderizado rápido y eficiente. Sin embargo, las desventajas de Mapbox están relacionadas con su modelo de precios, ya que aunque es más asequible que Google Maps en las etapas iniciales, puede volverse costoso a medida que aumenta la base de usuarios o el volumen de vistas. Esto puede ser un inconveniente a largo plazo si la aplicación crece rápidamente o si se requiere una gran cantidad de vistas o interacciones con los mapas. A pesar de ser más económico en términos de tarifas iniciales, los costos pueden escalar rápidamente, lo que requiere una planificación adecuada para prever los gastos asociados a la expansión.

4. MapLibre⁴ (anteriormente Mapbox GL JS open-source)

MapLibre es una bifurcación de código abierto de Mapbox GL JS, lo que significa que ofrece muchas de las características avanzadas de Mapbox sin la necesidad de

³ <https://docs.mapbox.com/api/maps>

⁴ <https://maplibre.org/maplibre-gl-js/docs/>

pagar licencias o tarifas. Es una excelente opción para aplicaciones que requieren un motor de renderizado de mapas interactivos y no desean depender de plataformas comerciales. Entre las ventajas de MapLibre se encuentra su completa apertura como software open-source, lo que permite a los desarrolladores tener acceso al código fuente y modificarlo según sus necesidades. Esto proporciona una gran flexibilidad para personalizar la visualización de los mapas y ajustarlos a requisitos específicos del proyecto. Además, al ser completamente libre de costos, es una opción atractiva para proyectos con presupuestos limitados, ya que no hay cargos asociados con el uso de la API o las vistas del mapa. La capacidad de personalizar el estilo y comportamiento del mapa sin restricciones comerciales le otorga una gran ventaja en términos de flexibilidad y control. Sin embargo, MapLibre también presenta algunas desventajas. Requiere un mayor nivel de personalización y habilidad técnica para implementar la solución de manera eficaz, lo que podría ser un reto para desarrolladores sin experiencia en programación avanzada o que busquen una solución más rápida y directa. Además, aunque MapLibre tiene una comunidad activa, no cuenta con el soporte comercial y oficial que ofrece Mapbox, lo que puede significar que los desarrolladores deban resolver por sí mismos los problemas técnicos que surjan durante la implementación. Esta falta de soporte profesional directo puede ser un obstáculo para equipos que buscan una solución más sencilla o que requieren asistencia rápida en caso de problemas técnicos.

5. Here Maps⁵

Here Maps es una plataforma potente que ofrece una API robusta para la integración de mapas, geocodificación y navegación en tiempo real. Una de sus principales ventajas es que proporciona funcionalidades avanzadas de geolocalización y mapas detallados, que incluyen actualizaciones en tiempo real sobre tráfico, condiciones de la carretera y rutas, lo que la convierte en una opción atractiva para aplicaciones que requieren datos precisos de localización y navegación. También tiene un costo más accesible en comparación con Google Maps, lo que la convierte en una alternativa interesante para proyectos que necesitan una solución económica sin sacrificar funcionalidades avanzadas. Además, Here Maps ofrece un SDK oficial tanto para Android como para iOS, lo que facilita la integración de mapas y geolocalización en las aplicaciones móviles. Este SDK incluye características de navegación avanzada, como indicaciones paso a paso y análisis en tiempo real del tráfico, lo que es útil para aplicaciones que se centran en la movilidad o en la optimización de rutas. Sin embargo, Here Maps también presenta algunas desventajas. A pesar de ser una alternativa potente, puede resultar más compleja de manejar que opciones más populares como Google Maps o Mapbox, especialmente para desarrolladores sin mucha experiencia en el uso de sus herramientas. La documentación y el soporte pueden no ser tan amplios o accesibles como los que ofrece Google Maps, lo que puede generar dificultades adicionales en el proceso de integración y desarrollo. Además, aunque Here Maps

⁵ <https://www.here.com/docs/>

es más asequible que Google Maps, sus costos pueden aumentar dependiendo del volumen de uso y las funcionalidades específicas que se necesiten, lo que requiere una planificación cuidadosa del presupuesto a largo plazo.

La decisión depende en gran medida de los objetivos y el presupuesto del proyecto. Si la prioridad es la facilidad de uso, la integración directa y el acceso a una amplia variedad de funcionalidades listas para usar, Google Maps es la opción más atractiva, especialmente para aplicaciones que requieren mapas de alta calidad y características como el tráfico en tiempo real, Street View y las rutas. Sin embargo, si se busca una solución más personalizada, económica o open-source, alternativas como Mapbox, Here Maps o OpenStreetMap pueden ofrecer ventajas significativas en cuanto a flexibilidad y costos a largo plazo. La decisión también dependerá del enfoque multiplataforma, ya que Flutter podría requerir más esfuerzo en la integración de Google Maps en comparación con Kotlin, que ofrece soporte nativo más directo. En resumen, Google Maps es la mejor opción si se busca una solución confiable y ampliamente soportada, pero no es la única opción viable dependiendo de las necesidades y los recursos del proyecto.

2.3 Lenguajes de Programación y Frameworks

La elección de un lenguaje de programación para desarrollar una aplicación es una decisión fundamental que puede afectar diversos aspectos del proyecto, desde el rendimiento y la eficiencia hasta la facilidad de mantenimiento y escalabilidad. Cada lenguaje tiene características que lo hacen más adecuado para ciertos tipos de aplicaciones, y la selección debe basarse en una combinación de factores técnicos, organizativos y estratégicos.

Uno de los principales motivos para elegir un lenguaje sobre otro es el rendimiento y la eficiencia. Algunos lenguajes son conocidos por su capacidad de manejar operaciones de bajo nivel con gran rapidez y optimización de recursos. Esto los hace ideales para aplicaciones que requieren un alto rendimiento, como los videojuegos, sistemas embebidos o software que necesita manejar grandes volúmenes de datos en tiempo real. Por otro lado, lenguajes interpretados como Python suelen ser más lentos en comparación, pero su facilidad de uso y flexibilidad pueden hacerlos una mejor opción para aplicaciones donde la velocidad no es el factor más crítico.

La facilidad de desarrollo y la productividad son otro aspecto clave al momento de decidir. Lenguajes como Python o Ruby permiten escribir código de manera concisa y clara, lo que facilita el desarrollo rápido y reduce el tiempo de implementación. Comparados con lenguajes como Java, que requieren una mayor cantidad de líneas de código para lograr la misma funcionalidad, estos lenguajes dinámicos son

preferidos en entornos donde la rapidez de desarrollo es crucial, como en startups o proyectos con plazos ajustados.

El ecosistema y las librerías disponibles también juegan un papel fundamental. Un lenguaje con una comunidad activa y una amplia variedad de librerías facilita la implementación de funcionalidades avanzadas sin necesidad de desarrollar todo desde cero. Por ejemplo, Python ha ganado popularidad en el campo de la inteligencia artificial y el análisis de datos gracias a bibliotecas como TensorFlow, Pandas y Scikit-learn, lo que lo convierte en la mejor opción para proyectos relacionados con estas áreas.

Otro factor importante es la compatibilidad y portabilidad del lenguaje. Algunos lenguajes están diseñados para funcionar mejor en ciertos entornos. Java y Kotlin, por ejemplo, son los más adecuados para el desarrollo de aplicaciones móviles en Android, mientras que Swift es el estándar para iOS. Sin embargo, existen soluciones multiplataforma como Flutter, basado en Dart, o React Native, basado en JavaScript, que permiten desarrollar aplicaciones para ambas plataformas con un solo código, reduciendo el tiempo y los costos de desarrollo.

La seguridad es otro criterio relevante. Algunos lenguajes están diseñados con características que minimizan riesgos comunes en el desarrollo de software. Rust, por ejemplo, previene errores de manejo de memoria que pueden causar vulnerabilidades en aplicaciones escritas en C o C++. Java, por su parte, cuenta con una gestión de memoria automática que reduce la posibilidad de errores como desbordamientos de búfer o fugas de memoria.

Finalmente, la comunidad y el soporte disponible pueden marcar una gran diferencia en la elección de un lenguaje. Lenguajes con una comunidad activa, como JavaScript, Python o Java, cuentan con abundante documentación, foros de discusión y recursos educativos que facilitan la resolución de problemas y el aprendizaje continuo. En cambio, lenguajes menos populares pueden presentar mayores dificultades en este sentido, lo que podría ralentizar el desarrollo o dificultar la resolución de errores.

Por lo tanto, la elección de un lenguaje de programación debe basarse en una evaluación cuidadosa de las necesidades del proyecto y los recursos disponibles. No existe un lenguaje universalmente superior, sino que cada uno tiene ventajas y desventajas que deben ser analizadas en función del contexto. Considerando todo esto se buscó tomar una decisión informada que optimice el desarrollo y el mantenimiento de la aplicación en el tiempo. A continuación, se detallan los lenguajes y frameworks más relevantes para el desarrollo de aplicaciones móviles y cómo se ajustan a las necesidades del proyecto.

1. Kotlin

Kotlin ha ganado una gran popularidad en los últimos años como lenguaje de programación para el desarrollo de aplicaciones móviles, especialmente en el ecosistema Android. Este lenguaje, desarrollado por JetBrains, combina la concisión de la sintaxis con la interoperabilidad con Java, lo que facilita la migración de proyectos existentes.

Entre sus principales ventajas se encuentran su seguridad, su facilidad de aprendizaje y su compatibilidad con las últimas características de Android. Kotlin es oficialmente soportado por Google para el desarrollo de Android, lo que garantiza actualizaciones regulares y buena integración con nuevas características de Android. Las herramientas de desarrollo en Kotlin, como Android Studio, están bien establecidas y son robustas, lo que facilita la creación y depuración de aplicaciones.

Sin embargo, su principal desventaja radica en la necesidad de mantenerse al día con las actualizaciones del lenguaje que suele ir variando pero sigue ofreciendo retrocompatibilidad, por lo que no termina siendo un problema tan grave.

2. Swift

Swift es el lenguaje de programación desarrollado por Apple para la creación de aplicaciones iOS, macOS, watchOS y tvOS. Introducido en 2014, Swift ha ido ganando terreno rápidamente gracias a su sintaxis clara y concisa, su seguridad mejorada y su alto rendimiento.

Una de las principales ventajas de Swift es su capacidad para evitar errores comunes de programación gracias a características como el manejo de opcionales y la inferencia de tipos. Swift sigue evolucionando rápidamente, y Apple invierte fuertemente en mejorar el lenguaje, lo que hace que sea una apuesta a largo plazo para los desarrolladores de iOS.

Su principal desventaja radica en su limitada interoperabilidad con otros lenguajes como Objective-C, lo que puede dificultar la integración de código heredado.

3. Java

Java ha sido durante mucho tiempo uno de los lenguajes de programación más utilizados en el desarrollo de aplicaciones móviles, especialmente en el ecosistema Android. A pesar de su larga trayectoria, Java sigue siendo una opción sólida para el desarrollo móvil gracias a su portabilidad, su robusta biblioteca estándar y su amplia comunidad de desarrolladores.

Entre sus ventajas se encuentran su madurez, su soporte multiplataforma y su amplia gama de herramientas y frameworks disponibles. Java es altamente compatible con versiones anteriores, lo que permite a los desarrolladores mantener

aplicaciones más antiguas sin problemas de compatibilidad al actualizar el SDK de Android. Además es el único lenguaje de la lista en el que se tuvo experiencia previa durante la carrera de Ingeniería en Sistemas.

Java también presenta algunas desventajas, como su verbosidad y su rendimiento comparativamente inferior en ciertos casos (aunque es competitivo para muchas aplicaciones que no requieren un alto rendimiento gráfico).

4. React Native

React Native es un framework de desarrollo móvil creado por Facebook que permite construir aplicaciones móviles multiplataforma utilizando JavaScript y React.

Una de las principales ventajas de React Native es su capacidad para compartir código entre plataformas, lo que permite a los desarrolladores crear aplicaciones para iOS y Android con un solo código base. Además, React Native ofrece un rendimiento cercano al de las aplicaciones nativas gracias a su arquitectura basada en componentes. También tiene una comunidad muy activa y un amplio ecosistema de bibliotecas y componentes, lo que facilita encontrar soluciones o integraciones ya hechas.

Su principal desventaja radica en su dependencia de bibliotecas de terceros (aunque permite la integración con código nativo, lo que puede mejorar el rendimiento en ciertas áreas críticas) y en su curva de aprendizaje para aquellos que no están familiarizados con JavaScript o React.

5. Flutter

Flutter es un framework de desarrollo móvil creado por Google que permite construir aplicaciones móviles para iOS y Android desde un solo código base. Utilizando el lenguaje de programación Dart, Flutter ofrece un rendimiento excepcional gracias a su motor de renderizado personalizado y su enfoque en la creación de interfaces de usuario fluidas y atractivas.

Sus ventajas son su rapidez de desarrollo, su recarga en caliente para la visualización instantánea de cambios y su amplia colección de widgets personalizables. El rendimiento de Flutter es uno de sus mayores puntos fuertes, ya que utiliza su propio motor gráfico (Skia), lo que le da ventaja sobre otros frameworks en términos de animaciones fluidas y tiempos de respuesta. Además está creciendo rápidamente en popularidad, lo que significa que el soporte y las bibliotecas disponibles seguirán mejorando con el tiempo.

Su principal desventaja radica en su relativa juventud y en la falta de algunas características avanzadas presentes en lenguajes específicos de las plataformas, lo que resulta en que ciertas funcionalidades pueden requerir ser escritas en código nativo para cada una.

6. Python (con Kivy o BeeWare)

Aunque no es tan común como otros lenguajes de programación para aplicaciones móviles, Python se puede utilizar con frameworks como Kivy o BeeWare para el desarrollo de aplicaciones móviles. También permite construir aplicaciones para iOS y Android.

Es excelente para desarrolladores de Python que desean construir aplicaciones móviles sin aprender un nuevo lenguaje, ideal para aplicaciones gráficas o juegos que necesiten interfaces de usuario personalizadas y también es multiplataforma. El ecosistema de Python ofrece una gran cantidad de bibliotecas que pueden extender la funcionalidad de las aplicaciones de manera creativa.

Por contraparte, posee un peor rendimiento en comparación con los lenguajes nativos, lo que puede afectar la experiencia del usuario en aplicaciones exigentes y tiene una comunidad más pequeña y menos recursos en comparación con otros frameworks móviles como Flutter o React Native, lo que significa menos estabilidad y soporte. Python no está diseñado específicamente para móviles, por lo que las soluciones como Kivy o BeeWare no se integran tan bien con las APIs nativas de cada plataforma como otros lenguajes móviles.

7. Ruby (con RubyMotion)

RubyMotion permite utilizar Ruby para el desarrollo de aplicaciones móviles tanto en iOS como en Android. Es especialmente útil para aquellos familiarizados con Ruby.

Ruby es un lenguaje popular y conocido por su simplicidad, genera código nativo, lo que significa que las aplicaciones tienen buen rendimiento en comparación con otras herramientas de desarrollo multiplataforma y además RubyMotion tiene buena integración con las bibliotecas nativas de cada plataforma, permitiendo acceso completo a las API de iOS y Android.

Sin embargo, RubyMotion tiene una comunidad más pequeña y menos recursos en comparación con otras herramientas de desarrollo móvil como Flutter o React Native, y aunque genera código nativo, el soporte y la documentación disponibles son limitados, lo que puede dificultar el desarrollo de aplicaciones más complejas. Además puede ser costoso, ya que RubyMotion requiere una licencia de pago para acceder a todas sus características.

8. Objective-C

Objective-C es el lenguaje de programación original utilizado para el desarrollo de aplicaciones para iOS y macOS antes de la introducción de Swift. A pesar de que Swift ha reemplazado a Objective-C como el lenguaje principal para iOS, muchas aplicaciones heredadas todavía utilizan Objective-C, y el soporte de Apple para el lenguaje sigue siendo fuerte.

La principal ventaja es la interoperabilidad con C y C++, lo que lo hace ideal para trabajar con código legado y bibliotecas antiguas, cuenta con una extensa base de código existente, lo cual es útil si se está manteniendo o expandiendo aplicaciones antiguas y es estable, maduro y bien soportado en el ecosistema de Apple.

Entre sus desventajas se encuentra la sintaxis más compleja y verbosea en comparación con Swift, lo que puede dificultar el aprendizaje para nuevos desarrolladores y ha sido reemplazado por el mismo como el lenguaje preferido para el desarrollo en iOS.

Las aplicaciones que fueron mencionadas en la Sección "Análisis de Apps Existentes", (Google Maps, Moovit, Waze, Uber, PedidosYa, Transit) es muy probable que estén programadas utilizando una combinación de lenguajes de programación y tecnologías según las plataformas en las que operan (iOS, Android, y web). La información sobre los lenguajes de programación utilizados para desarrollar aplicaciones como Google Maps, Moovit, Waze, Uber, PedidosYa, y Transit se basa en prácticas comunes de desarrollo de aplicaciones móviles y en detalles proporcionados por las propias compañías a través de publicaciones técnicas, artículos de ingeniería, conferencias de desarrolladores, y documentación pública. A menos que una aplicación sea desarrollada con tecnologías multiplataforma (como React Native o Flutter), los lenguajes nativos son los estándares. Por lo tanto, la información sugiere que estas aplicaciones utilizan Java o Kotlin para Android y Swift u Objective-C para iOS.

Dado que la aplicación se basa en la geolocalización y el uso del GPS, es esencial elegir un lenguaje o framework que ofrezca un buen soporte para las funciones de ubicación en tiempo real, ya que esto impacta directamente en la precisión y el rendimiento de la app. A continuación se detalla cómo manejan el soporte de GPS y geolocalización los lenguajes y frameworks mencionados:

1. Kotlin

Kotlin es un lenguaje de programación moderno y versátil que cuenta con soporte completo para todas las APIs nativas de Android, incluyendo las de Google Play Services para geolocalización. Gracias a esto, Kotlin se convierte en una excelente opción para gestionar el GPS en dispositivos móviles, permitiendo obtener la ubicación actual del usuario con gran precisión y acceder a actualizaciones periódicas de la posición. Además, su integración con estas APIs permite ajustar la precisión del servicio de geolocalización en función de las necesidades de la aplicación, desde una precisión alta utilizando GPS hasta modos de bajo consumo energético aprovechando redes Wi-Fi y torres celulares. Otra de las ventajas que ofrece Kotlin es su integración con Google Maps y otras librerías de mapas, lo que facilita la implementación de funcionalidades avanzadas como el seguimiento en tiempo real de la ubicación del usuario, la navegación asistida y la geocodificación.

inversa, permitiendo así una experiencia más completa y personalizada dentro de la aplicación.

2. Swift

Swift, el lenguaje de programación desarrollado por Apple, proporciona un soporte eficiente para GPS a través de las APIs nativas de Core Location y MapKit. Estas herramientas permiten acceder con gran precisión a la ubicación del dispositivo, facilitando la configuración de actualizaciones en segundo plano y permitiendo el control del nivel de precisión requerido, desde una ubicación aproximada a nivel de ciudad hasta coordenadas exactas. Esto resulta especialmente útil en aplicaciones que requieren información en tiempo real sobre la posición del usuario. Además, Swift ofrece una integración fluida con Apple Maps, lo que permite a los desarrolladores mostrar la ubicación del usuario en un mapa, trazar rutas y calcular estimaciones de tiempo de llegada (ETA) con un alto grado de precisión y eficiencia.

3. Java

Java, al igual que Kotlin, tiene acceso completo a las APIs de Android y Google Play Services, lo que le permite ofrecer todas las funcionalidades avanzadas de geolocalización necesarias para el desarrollo de aplicaciones móviles. Al ser un lenguaje maduro y ampliamente utilizado, cuenta con una gran cantidad de documentación, ejemplos y bibliotecas disponibles que facilitan la implementación de sistemas de GPS de alta precisión. Además, Java permite una integración completa con Google Maps y las APIs de localización de Google, lo que posibilita el desarrollo de aplicaciones con funciones avanzadas de mapeo y navegación, garantizando así una experiencia de usuario óptima y eficiente.

4. React Native

React Native, a pesar de no ser un lenguaje nativo, proporciona una integración adecuada con las APIs de geolocalización mediante el uso del módulo react-native-geolocation-service. Este módulo permite acceder a las funcionalidades de GPS tanto en dispositivos iOS como Android, facilitando el desarrollo de aplicaciones híbridas que requieren el uso de datos de ubicación. Además, existen paquetes disponibles para integrar mapas de Google y Apple en las aplicaciones, lo que amplía las posibilidades de implementación. Sin embargo, aunque React Native es eficiente para la mayoría de las aplicaciones que requieren GPS, puede presentar ciertas limitaciones en proyectos que demandan funcionalidades avanzadas o personalizadas, como la actualización de ubicación en segundo plano o la obtención de coordenadas con alta precisión de manera continua. En estos casos, puede ser necesario recurrir a la escritura de código nativo en Kotlin o Swift para optimizar el rendimiento y la funcionalidad de la aplicación.

5. Flutter

Flutter cuenta con un buen soporte para geolocalización mediante el uso del plugin geolocator. Este complemento proporciona acceso a las funcionalidades de GPS en dispositivos Android e iOS, permitiendo obtener la ubicación actual del usuario y recibir actualizaciones periódicas de su posición. Para la integración de mapas dentro de las aplicaciones desarrolladas con Flutter, se puede utilizar el paquete `google_maps_flutter`, el cual facilita la inclusión de Google Maps en la aplicación y la implementación de diversas funcionalidades de navegación y mapeo. Asimismo, Flutter cuenta con soporte para otras APIs de geolocalización y mapas, lo que amplía las opciones disponibles para los desarrolladores a la hora de diseñar aplicaciones basadas en la ubicación.

6. Objective-C

Objective-C, al igual que Swift, ofrece acceso completo a las APIs de Core Location y MapKit, lo que permite a los desarrolladores implementar con facilidad funcionalidades avanzadas de GPS en aplicaciones móviles. Sin embargo, a pesar de seguir siendo un lenguaje totalmente funcional y compatible con el ecosistema de Apple, Objective-C ha sido desplazado por Swift como el lenguaje preferido para el desarrollo en iOS. Apple ha priorizado el soporte y desarrollo de Swift, lo que lo convierte en una opción más recomendable a largo plazo en comparación con su antecesor, especialmente en lo que respecta a nuevas funcionalidades y actualizaciones de las APIs de geolocalización.

7. Python (con Kivy o BeeWare)

Python, mediante frameworks como Kivy y BeeWare, ofrece cierto nivel de soporte para funcionalidades de GPS en aplicaciones móviles. Sin embargo, dado que estos frameworks no son nativos, su integración con los servicios de geolocalización es limitada y menos eficiente en comparación con los lenguajes específicos para desarrollo móvil. Aunque es posible acceder a la ubicación del usuario mediante el uso de bibliotecas de terceros, la implementación de estas funcionalidades suele ser más compleja y puede presentar limitaciones en términos de rendimiento y precisión. Dado que Python no está específicamente diseñado para el desarrollo de aplicaciones móviles, los proyectos que dependen del GPS o requieren actualizaciones en tiempo real pueden no alcanzar el nivel de eficiencia deseado al utilizar estos frameworks.

8. Ruby (con RubyMotion)

Ruby, a través de RubyMotion, permite acceder a las APIs nativas tanto de Android como de iOS, lo que hace posible la implementación de funciones de GPS dentro de las aplicaciones desarrolladas con este lenguaje. Sin embargo, aunque técnicamente es viable, RubyMotion ha perdido popularidad en los últimos años, lo

que ha generado una disminución en la disponibilidad de documentación y soporte. Esto puede representar un desafío para los desarrolladores, especialmente cuando se requieren funcionalidades avanzadas como la geolocalización en segundo plano o la integración con servicios de mapas. Dada esta situación, el uso de RubyMotion para aplicaciones que dependen en gran medida del GPS no es una opción ampliamente recomendada en el contexto del desarrollo móvil moderno.

Para el desarrollo de una aplicación basada en geolocalización y uso intensivo de GPS, las mejores opciones dependen del enfoque nativo o multiplataforma. Para Android, Kotlin ofrece un soporte óptimo para APIs de GPS y Google Maps, mientras que en iOS, Swift es ideal con acceso completo a Core Location y MapKit. Si se busca un enfoque multiplataforma, Flutter es una excelente opción por su rendimiento cercano al nativo y buen soporte de GPS, aunque React Native también es viable, especialmente para proyectos más simples. Cabe destacar que optar por tecnologías nativas garantiza un mejor rendimiento y acceso a funcionalidades avanzadas. Teniendo en cuenta todo el análisis hecho, las dos opciones más tentadoras fueron trabajar con Kotlin y programar solo en Android, o hacerlo multiplataforma con Flutter y Dart, ya que ambas ofrecen la experiencia de incorporar nuevos conocimientos y herramientas populares en el desarrollo móvil.

2.4 Base de Datos

Las bases de datos son sistemas diseñados para almacenar, organizar y gestionar grandes volúmenes de información de manera estructurada. Su principal propósito es permitir el acceso eficiente a los datos, facilitando su recuperación, modificación y eliminación de manera segura y confiable. Estos sistemas son fundamentales en la informática y en el desarrollo de software, ya que permiten a las aplicaciones manejar información de forma persistente, evitando la pérdida de datos entre sesiones de uso.

La utilidad de las bases de datos radica en su capacidad para almacenar información de manera ordenada, permitiendo que los usuarios y sistemas puedan acceder a ella mediante consultas estructuradas. Gracias a esto, se pueden obtener respuestas rápidas y precisas sin necesidad de recorrer grandes volúmenes de información manualmente. Además, las bases de datos garantizan la integridad de los datos, evitando inconsistencias y duplicaciones mediante mecanismos de control. Otro aspecto clave es la seguridad, ya que permiten definir permisos de acceso, asegurando que solo usuarios autorizados puedan modificar o consultar determinados datos. Asimismo, estos sistemas están diseñados para manejar múltiples accesos simultáneos, lo que posibilita que varios usuarios interactúen con la información al mismo tiempo sin comprometer su integridad.

En el contexto de las aplicaciones móviles, las bases de datos desempeñan un papel crucial al gestionar la información que utilizan los usuarios y la propia

aplicación. Uno de sus usos principales es el almacenamiento de datos personales, como credenciales de usuario, preferencias y configuraciones personalizadas, lo que permite mejorar la experiencia de uso al recordar información relevante entre sesiones. También resultan esenciales para el almacenamiento de datos sin conexión a internet, ya que muchas aplicaciones necesitan acceder a cierta información incluso cuando no hay conectividad. Para esto, se utilizan bases de datos locales, como SQLite en Android o Core Data en iOS, que permiten guardar datos en el dispositivo y sincronizarlos con un servidor cuando la conexión esté disponible.

Otro uso fundamental de las bases de datos en aplicaciones móviles es la sincronización con servidores remotos. Muchas apps dependen de información actualizada en tiempo real, como redes sociales, aplicaciones de mensajería o sistemas de geolocalización. En estos casos, se utilizan bases de datos en la nube, como Firebase, MongoDB Atlas o MySQL en servidores, que permiten a los dispositivos acceder a la información de manera centralizada y mantenerla actualizada para todos los usuarios. Esto es especialmente útil en aplicaciones colaborativas, donde múltiples usuarios pueden modificar los mismos datos y los cambios deben reflejarse en todos los dispositivos en tiempo real.

Además, las bases de datos permiten optimizar la velocidad de respuesta de una aplicación móvil. Al almacenar y gestionar información localmente o en servidores bien estructurados, se reduce la necesidad de procesar datos repetitivos, mejorando la eficiencia y el rendimiento general de la app. También juegan un papel clave en la administración de contenido dinámico, como en aplicaciones de noticias, redes sociales o sistemas de transporte, donde la información cambia constantemente y debe estar disponible para los usuarios de forma inmediata.

En este proyecto, la base de datos cumple una función esencial en la gestión de la ubicación de los colectivos reportada por los usuarios. A través de este sistema, se puede almacenar y procesar la información sobre las rutas disponibles, los tiempos estimados de llegada y los reportes en tiempo real, permitiendo a los usuarios tomar decisiones informadas sobre su transporte. Sin un sistema de base de datos eficiente, la aplicación no podría gestionar adecuadamente la gran cantidad de datos que se generan y necesitan actualizarse constantemente, lo que afectaría la precisión y utilidad del servicio ofrecido.

Para analizar las bases de datos que podrían ser más adecuadas para el proyecto, se deben considerar las necesidades específicas de la aplicación en términos de almacenamiento de datos, sincronización en tiempo real y escalabilidad. Existen múltiples opciones que van desde bases de datos locales como SQLite, hasta soluciones en la nube como Firebase Realtime Database, cada una con sus características y ventajas. A continuación, se detallan las opciones principales.

1. SQLite

SQLite es una base de datos relacional que se ejecuta directamente en el dispositivo, lo que significa que no depende de servidores externos para almacenar datos. Al ser una base de datos local, permite un acceso rápido y eficiente a los datos sin necesidad de conexión a internet. Utiliza SQL, lo que permite realizar consultas complejas que incluyen joins y subconsultas, lo que la hace adecuada para aplicaciones que gestionan datos relacionales. Sin embargo, SQLite presenta limitaciones importantes cuando se trata de sincronizar datos entre dispositivos. Dado que no está diseñada para la sincronización en tiempo real, si la aplicación requiere compartir datos entre usuarios o dispositivos de manera instantánea, SQLite necesitará ser complementada con soluciones adicionales. Además, aunque SQLite es eficiente para manejar volúmenes pequeños a medianos de datos, no es la opción más adecuada cuando la aplicación requiere escalabilidad y capacidad para manejar grandes volúmenes de datos distribuidos.

2. Firebase Realtime Database

Firebase Realtime Database es una solución basada en la nube que permite la sincronización de datos en tiempo real. Esta base de datos NoSQL almacena los datos en formato JSON, lo que la hace adecuada para aplicaciones que necesitan actualizar datos al instante entre múltiples usuarios o dispositivos. Firebase se integra fácilmente con Android e iOS a través de sus SDKs bien documentados, y permite gestionar no solo el almacenamiento de datos, sino también otras funcionalidades como la autenticación de usuarios, notificaciones push y análisis. Esta opción es especialmente útil para aplicaciones de seguimiento en vivo, como las que requieren rastrear la ubicación de colectivos o usuarios en tiempo real. La principal desventaja de Firebase es su modelo de precios, que si bien incluye un plan gratuito, puede volverse costoso a medida que la aplicación escala en términos de usuarios activos o cantidad de datos generados. Sin embargo, para proyectos pequeños o en etapas iniciales, el plan gratuito puede ser suficiente, y su integración es más sencilla que la de otras soluciones más complejas. Además, su enfoque en la sincronización en tiempo real hace que sea una de las mejores opciones cuando se requiere una actualización instantánea de los datos en todos los dispositivos conectados.

3. Room

Room es una capa de abstracción sobre SQLite diseñada específicamente para Android, que facilita la interacción con bases de datos SQLite en Kotlin. Room reduce la complejidad de manejar SQL directamente y ofrece soporte nativo para LiveData y Coroutines, lo que facilita la gestión de bases de datos en tiempo real dentro de la aplicación. Sin embargo, al igual que SQLite, Room es una solución local que no proporciona funcionalidad de sincronización en tiempo real entre dispositivos. Por lo tanto, si la aplicación requiere compartir datos en tiempo real

entre múltiples usuarios, Room no será suficiente por sí sola. Sin embargo, si el proyecto está centrado exclusivamente en la gestión de datos locales en un dispositivo Android, Room puede ofrecer una solución eficiente y sencilla para manejar bases de datos relacionales.

4. MongoDB Atlas Device SDKs

MongoDB Atlas Device SDKs, antes conocido como Realm, es una base de datos NoSQL orientada a documentos que permite la sincronización de datos en tiempo real entre dispositivos y la nube. Al ser parte del ecosistema de MongoDB Atlas, esta opción ofrece una solución escalable que puede manejar grandes volúmenes de datos y usuarios. MongoDB Atlas Device SDKs es ideal para aplicaciones que necesitan mantener datos sincronizados en tiempo real, como las que siguen la ubicación de los usuarios o permiten la interacción en tiempo real entre dispositivos. Además, su esquema flexible basado en BSON/JSON permite manejar datos no relacionales de manera eficiente. Aunque el uso local de MongoDB Atlas Device SDKs es gratuito, el uso de la infraestructura en la nube de MongoDB Atlas genera costos que pueden aumentar a medida que la aplicación crece. Además, la configuración de MongoDB Atlas es más compleja que la de Firebase, especialmente si se requieren características adicionales como notificaciones push, que deben implementarse a través de servicios adicionales, como Firebase Cloud Messaging. Sin embargo, su escalabilidad y flexibilidad lo convierten en una opción adecuada para aplicaciones que requieren tanto sincronización en tiempo real como capacidad para manejar grandes volúmenes de datos.

5. Couchbase Lite

Couchbase Lite es una base de datos NoSQL orientada a documentos diseñada para dispositivos móviles. Esta base de datos permite la sincronización automática entre dispositivos utilizando Couchbase Sync Gateway, lo que la convierte en una opción adecuada para aplicaciones que requieren la sincronización distribuida de datos. Couchbase Lite es una opción poderosa para aplicaciones que operan tanto online como offline, permitiendo que los datos se actualicen automáticamente entre dispositivos cuando se restablece la conexión. Sin embargo, su configuración puede ser más compleja en comparación con Firebase, y su implementación de sincronización distribuida puede ser excesiva si la aplicación no requiere este nivel de complejidad. Esta base de datos es más adecuada para aplicaciones que necesitan manejar datos entre muchos dispositivos distribuidos, pero puede ser innecesaria para aplicaciones de menor escala que no requieren sincronización en tiempo real.

6. Supabase

Supabase es una alternativa de código abierto a Firebase, basada en PostgreSQL. Ofrece características similares a Firebase, como la sincronización en tiempo real y la gestión de usuarios, pero con la diferencia de que utiliza SQL. Si el proyecto requiere un esquema de base de datos relacional, Supabase puede ser una excelente opción, además de ser de código abierto, lo que permite mayor control sobre la infraestructura. Aunque es una opción interesante, su comunidad y ecosistema son más pequeños en comparación con Firebase, lo que puede resultar en menos recursos disponibles y soporte limitado. Sin embargo, si se busca una solución basada en SQL sin depender de ecosistemas cerrados, Supabase es una opción viable.

7. ObjectBox

ObjectBox es una base de datos orientada a objetos que se distingue por su alta velocidad y eficiencia, especialmente en aplicaciones móviles que manejan grandes volúmenes de datos. ObjectBox está optimizada para trabajar localmente en dispositivos, lo que la convierte en una opción excelente cuando el rendimiento y la velocidad son prioritarios. Sin embargo, la sincronización en tiempo real solo está disponible como parte de su servicio de pago, lo que podría limitar su utilidad en aplicaciones que requieren una actualización constante de los datos entre múltiples dispositivos sin un costo adicional.

La elección de la base de datos más adecuada depende de varios factores clave, como el tipo de datos a manejar, la necesidad de sincronización en tiempo real, el control sobre la infraestructura y la escalabilidad de la aplicación. Es importante también considerar si la base de datos debe ser relacional o no relacional, así como si los datos deben almacenarse de forma local o en la nube.

Las bases de datos relacionales como SQLite y Room son útiles cuando se necesita un modelo estructurado para gestionar datos mediante tablas y relaciones. Son ideales para aplicaciones que manejan datos de manera organizada y necesitan mantener relaciones entre diferentes entidades. Además, estas bases de datos permiten realizar consultas complejas utilizando SQL, lo que ofrece gran flexibilidad en el manejo de los datos.

En contraste, las bases de datos orientadas a objetos como ObjectBox y MongoDB Atlas Device SDKs permiten almacenar datos en un formato que se alinean más estrechamente con la estructura de objetos utilizada en el código de la aplicación. Esto las convierte en una excelente opción para aplicaciones que manejan datos complejos o que requieren una mayor flexibilidad en su esquema. La principal ventaja de este tipo de bases de datos es que son más intuitivas en términos de cómo se integran con el código de la aplicación, ya que los objetos en la aplicación se almacenan directamente sin necesidad de realizar un mapeo entre objetos y

tablas. Sin embargo, las bases orientadas a objetos pueden no ser tan eficientes cuando se trata de realizar consultas complejas o mantener la normalización de los datos, ya que no ofrecen la misma robustez que las bases de datos relacionales en términos de integridad referencial. También, muchas de estas soluciones requieren configuraciones adicionales si se desea usar infraestructura en la nube, lo que podría añadir complejidad al proyecto.

En cuanto a si la base de datos debe ser local o en la nube, las bases de datos locales como SQLite y Room ofrecen grandes ventajas en términos de rendimiento, ya que los datos se almacenan directamente en el dispositivo del usuario, lo que permite un acceso rápido y sin dependencia de conexión a internet. Además, ofrecen un alto control sobre los datos, ya que todo el almacenamiento se maneja localmente, lo que puede ser crucial para aplicaciones que operan en entornos sin conexión o en zonas con mala conectividad. Sin embargo, estas bases de datos locales tienen la limitación de que no permiten una sincronización en tiempo real entre usuarios o dispositivos, lo que es esencial en una aplicación que gestione datos compartidos como la ubicación en vivo de colectivos y usuarios.

Por el contrario, las bases de datos en la nube como Firebase Realtime Database, MongoDB Atlas Device SDKs y Supabase permiten la sincronización en tiempo real entre dispositivos y usuarios. Estas soluciones son ideales para aplicaciones que necesitan actualizar los datos instantáneamente entre varios usuarios, lo cual es fundamental para el seguimiento en vivo de la ubicación de colectivos y la interacción entre usuarios en tiempo real. Además, las soluciones en la nube permiten la escalabilidad, ya que pueden manejar grandes volúmenes de datos y usuarios sin que sea necesario gestionar infraestructura adicional. No obstante, su principal desventaja es la dependencia de una conexión a internet, lo que podría afectar la experiencia de usuario en zonas con poca cobertura. También, los costos asociados con las soluciones en la nube pueden crecer rápidamente a medida que la aplicación se expande, especialmente si la base de usuarios o el volumen de datos es alto.

Por todo lo anteriormente mencionado, se consideró que Firebase Realtime Database es la mejor opción para este proyecto debido a su capacidad para manejar sincronización en tiempo real y su escalabilidad. Su integración tanto en Kotlin como en Flutter es sencilla, y su ecosistema ofrece ventajas adicionales como autenticación, notificaciones push y servicios de análisis que pueden mejorar la experiencia del usuario y agilizar el desarrollo. A pesar de que MongoDB Atlas Device SDKs y Supabase ofrecen capacidades similares, Firebase tiene la ventaja de ser un servicio completamente integrado y muy bien soportado dentro del ecosistema de Google, lo que facilita la implementación de características adicionales sin necesidad de integraciones externas. Además, su plan gratuito es suficiente para cubrir las necesidades de este proyecto en sus etapas iniciales, lo que la convierte en una opción económica y accesible.

2.5 Cálculo y Actualización de Rutas y Ubicaciones

En el desarrollo de una aplicación para el seguimiento de colectivos, es fundamental determinar dónde se realizará el cálculo de las rutas y la actualización de las ubicaciones, ya sea en el dispositivo del usuario o en un servidor centralizado. Esta decisión influye significativamente en el rendimiento de la aplicación, la eficiencia en el uso de los recursos y la experiencia del usuario. Por lo tanto, en esta sección se busca analizar las ventajas y desventajas de realizar estas tareas en el cliente o en el servidor.

En cuanto al cálculo de rutas, se analizará en primer lugar la opción de ejecutarse en el cliente, es decir, directamente en el dispositivo del usuario. En este modelo, la aplicación recibe las ubicaciones en tiempo real de los colectivos y utiliza un algoritmo local para calcular la mejor ruta disponible hasta el destino del usuario. Esta estrategia tiene varias ventajas. En primer lugar, reduce la carga de procesamiento en el servidor, lo que puede significar una reducción en los costos operativos y una distribución más equitativa de los recursos. Además, dado que el cálculo se realiza localmente, los resultados pueden obtenerse más rápidamente, sin depender de la latencia de la red para comunicarse con un servidor externo.

No obstante, también presenta algunas desventajas importantes. El procesamiento de rutas en el cliente puede consumir una cantidad considerable de recursos del dispositivo, lo que podría agotar la batería y la memoria, especialmente en modelos más antiguos o de menor capacidad. Además, distintos usuarios podrían obtener resultados ligeramente diferentes debido a las diferencias en la capacidad de procesamiento de sus dispositivos. Por otro lado, la información sobre el tráfico y las condiciones viales puede no estar tan actualizada como cuando el cálculo se realiza en un servidor con acceso a bases de datos en tiempo real, aunque esta diferencia pueda ser mínima.

Alternativamente, el cálculo de rutas puede delegarse a un servidor centralizado. En este caso, el servidor recibe las solicitudes de ubicación tanto del usuario como del colectivo, calcula la mejor ruta utilizando servicios de mapas y algoritmos de enrute especializados y posteriormente envía los resultados al cliente. Este enfoque ofrece varias ventajas. En primer lugar, se garantiza la consistencia en los resultados, ya que todos los usuarios recibirán la misma información basada en datos y algoritmos uniformes. Además, el cliente no necesita realizar cálculos complejos, lo que es especialmente beneficioso para dispositivos con recursos limitados. Un beneficio adicional es el acceso a datos centralizados y actualizados, lo que permite un mejor análisis del tráfico y un cálculo de rutas más preciso.

Sin embargo, también existen desventajas en este enfoque. La carga en el servidor puede aumentar significativamente a medida que crece el número de usuarios activos, lo que podría requerir mayor infraestructura y generar costos adicionales. Además, la latencia en la red podría afectar los tiempos de respuesta, lo que podría impactar negativamente en la experiencia del usuario en momentos de alta demanda.

De manera similar, la actualización de las ubicaciones de los colectivos puede abordarse mediante una estrategia centralizada en el servidor o una distribución entre los clientes. En el primer caso, la información sobre la ubicación de los colectivos se almacena y distribuye desde el servidor, lo que garantiza la consistencia de los datos para todos los usuarios. Este modelo permite una mejor gestión de la sincronización de la información, reduciendo la sobrecarga en los dispositivos de los usuarios y permitiendo una administración eficiente del sistema. Además, que el acceso a la base de datos sea mediante el servidor permite mantener cierto control y seguridad sobre los datos dispuestos en la misma, lo que también resulta muy importante no solo en cuanto a la seguridad de los datos de los colectivos sino también los personales de cada usuario. Este enfoque brinda mayor seguridad ya que se minimiza las interacciones de los dispositivos móviles particulares con la base de datos.

No obstante, centralizar la actualización de ubicaciones en el servidor también presenta desventajas. Al igual que para el cálculo de rutas, a medida que aumenta el número de usuarios conectados, la carga en el servidor también se incrementa, lo que puede derivar en mayores costos operativos y en tiempos de respuesta más lentos durante períodos de alta demanda. Obviamente esto no impacta de la misma forma que en el caso anterior ya que hacer una consulta a la base de datos no resulta igual de costoso que realizar muchos cálculos para encontrar la mejor ruta, por lo que la tolerancia a la sobrecarga del servidor es mucho mayor.

Por otro lado, la actualización distribuida de ubicaciones permite que cada usuario contribuya con información sobre la ubicación del colectivo directamente en la base de datos compartida. Este enfoque tiene ventajas, como la reducción de la dependencia del servidor y la posibilidad de involucrar activamente a los usuarios en la recopilación de datos en tiempo real. Sin embargo, también implica ciertos riesgos, como la generación de un gran número de conexiones simultáneas a la base de datos, lo que podría afectar el rendimiento general de la aplicación si no se maneja adecuadamente, y la anteriormente mencionada vulnerabilidad en los datos.

En conclusión, la elección entre realizar los cálculos y actualizaciones en el cliente o en el servidor depende de un equilibrio entre el rendimiento, la precisión y la carga de trabajo distribuida. Ambas estrategias tienen ventajas y desventajas, y la solución óptima dependerá del caso de uso específico y de los recursos disponibles para el desarrollo y mantenimiento de la aplicación.

2.6 Técnicas de Predicción de Tiempos de Traslado

El cálculo de los tiempos de traslado en sistemas de transporte público es un área que ha recibido mucha atención en la literatura, debido a la importancia que tiene para mejorar la eficiencia del transporte urbano. Existen diversas técnicas y enfoques que se pueden aplicar para estimar estos tiempos, y cada uno de ellos tiene sus propias características, ventajas y limitaciones. La elección de una técnica u otra dependerá principalmente de los datos disponibles, el contexto específico del sistema de transporte en cuestión, y los recursos computacionales disponibles para procesar los datos. A continuación, se describe un análisis detallado de algunas de las técnicas más comunes utilizadas en la estimación de tiempos de traslado.

1. Regresión Lineal

La regresión lineal es un modelo estadístico ampliamente utilizado que tiene como objetivo encontrar una relación matemática entre una o más variables independientes (como la velocidad, la distancia recorrida o las condiciones del tráfico) y el tiempo de traslado. Este modelo parte de la premisa de que existe una relación lineal entre las variables involucradas.

En términos de ventajas, la regresión lineal es muy fácil de implementar y entender, lo que la hace accesible incluso para aquellos con poca experiencia en análisis estadístico. Además, no requiere de mucha potencia computacional, lo que la convierte en una opción adecuada cuando se dispone de recursos limitados. Sin embargo, este modelo tiene limitaciones importantes. No es capaz de capturar relaciones no lineales entre las variables, lo que significa que podría fallar al modelar escenarios más complejos, como el comportamiento dinámico del tráfico o los efectos de condiciones climáticas adversas. Esto puede reducir la precisión de las estimaciones de tiempo en contextos urbanos donde estos factores varían constantemente.

Los autores Cardona et al. (2015), analizaron la aplicabilidad de modelos de regresión lineal para estimar los tiempos de llegada de los buses en un sistema de transporte masivo en la ciudad de Pereira, Colombia.

Para ello, realizaron simulaciones del corredor del BRT MEGABUS, utilizando el software Transmodeler® para modelar la infraestructura del sistema y el lenguaje de programación R para analizar los datos. Se establecieron diferentes escenarios de prueba, considerando variaciones en las condiciones del tráfico y los planes de señalización, con el fin de evaluar la validez del modelo bajo distintas circunstancias.

En el análisis de los resultados, los autores encontraron que los modelos de regresión lineal no cumplían con los supuestos fundamentales de normalidad, independencia y homocedasticidad en la mayoría de los casos. A pesar de estos inconvenientes, se identificó que en un corredor específico, sin estaciones de parada ni intersecciones semaforizadas, sí fue posible construir un modelo de regresión lineal válido. Como conclusión, los autores destacaron que los modelos de regresión lineal clásica no son adecuados para estimar tiempos de viaje en sistemas de transporte masivo debido a las violaciones de los supuestos fundamentales.

2. Modelos ARIMA (AutoRegressive Integrated Moving Average)

Los modelos ARIMA son herramientas estadísticas utilizadas para el análisis y la predicción de series temporales. Estos modelos se aplican a datos que muestran patrones a lo largo del tiempo, como los históricos de tiempos de viaje en sistemas de transporte. ARIMA es especialmente útil para predecir valores futuros basándose en la tendencia, la estacionalidad y otros patrones de los datos pasados.

Los modelos ARIMA son muy efectivos para modelar datos que varían con el tiempo y exhiben patrones cíclicos, como los cambios en el tráfico durante las horas pico. Estos modelos también pueden incorporar estacionalidades, lo que les permite adaptarse a la repetición de ciertos patrones en el tiempo. Sin embargo, el ajuste de modelos ARIMA puede ser complejo, especialmente si los datos presentan fluctuaciones abruptas o no siguen patrones estables. Además, no son tan efectivos cuando los datos históricos son inconsistentes o cuando factores externos impredecibles (como accidentes o condiciones climáticas) afectan el tiempo de traslado de manera significativa.

Los autores Suwardo et al. (2010), investigaron la aplicación de modelos ARIMA para predecir los tiempos de viaje de colectivos en el corredor Ipoh-Lumut, en Malasia. La motivación principal del estudio fue la necesidad de mejorar la precisión en la predicción de tiempos de viaje, considerando que los métodos de regresión tradicionales requieren información detallada sobre los factores que afectan el tránsito, como retrasos en intersecciones, paradas de colectivos e incidentes en la vía, los cuales no siempre están disponibles con exactitud. A diferencia de estos enfoques, el modelo ARIMA permite realizar estimaciones confiables basándose únicamente en datos históricos de tiempos de viaje, sin necesidad de información adicional sobre las condiciones del tráfico en tiempo real.

Para la recolección de datos, los investigadores realizaron un registro a bordo durante un año, desde enero hasta diciembre de 2007, registrando los tiempos de llegada y salida de los colectivos en distintos puntos del recorrido. Se tomaron en cuenta factores como la ubicación de las paradas, la frecuencia de los servicios y las variaciones en la velocidad de los colectivos a lo largo del trayecto.

Los modelos ARIMA fueron aplicados en ambas direcciones del recorrido, es decir, desde Ipoh a Lumut y en el sentido inverso. Se probaron distintas configuraciones del modelo, evaluando diferentes órdenes para los componentes autorregresivos y de media móvil. El rendimiento de cada modelo se comparó utilizando métricas de error, en particular el error absoluto medio relativo (MARE) y el error absoluto medio porcentual de predicción (MAPPE), que permiten medir la precisión de las estimaciones en relación con los tiempos de viaje observados.

El estudio concluyó que dos modelos puntuales, fueron los más adecuados para la predicción de tiempos de viaje, lo que evidencia que es viable aplicar modelos ARIMA para este tipo de problemáticas. Además, los autores sugirieron que la integración de datos en tiempo real sobre las condiciones del tráfico podría mejorar aún más la precisión de las estimaciones.

3. Random Forest

El algoritmo Random Forest es un modelo de aprendizaje supervisado que utiliza múltiples árboles de decisión para realizar predicciones. En el caso de los tiempos de traslado, cada árbol puede modelar una relación entre variables como la velocidad de un colectivo, las condiciones del tráfico y el tiempo estimado de viaje.

Este enfoque es muy robusto y tiene la ventaja de manejar bien relaciones no lineales y complejas entre variables. Además, puede procesar grandes volúmenes de datos sin perder precisión, lo que lo hace adecuado para escenarios con grandes cantidades de información. Sin embargo, requiere una mayor capacidad computacional en comparación con los modelos estadísticos más simples. Además, puede resultar más difícil de interpretar, ya que el modelo final es una "caja negra" donde no siempre es claro cómo se llegó a una decisión específica.

El autor Li (2017), aplicó el modelo Random Forest para predecir el tiempo de llegada de los colectivos utilizando datos históricos de ubicación GPS, condiciones de tráfico, datos meteorológicos, y la hora del día. Este enfoque se implementó en Beijing, donde el sistema de transporte público es altamente utilizado y la precisión en la predicción de tiempos de llegada puede mejorar la experiencia del usuario.

Para entrenar el modelo utilizó datos de colectivos para realizar la predicción de la llegada en una parada específica. Los resultados mostraron que el modelo Random Forest presentó un rendimiento superior al de modelos tradicionales, logrando un error absoluto medio de predicción (MAPE) de 20.43%. Además, al aumentar el número de árboles (hasta 800 árboles), el modelo mostró una mejora en la precisión, lo que indica la capacidad del Random Forest para manejar grandes volúmenes de datos de manera eficiente.

El autor concluyó que el Random Forest es una técnica robusta y confiable para la predicción de tiempos de llegada de colectivos en sistemas de transporte público urbanos, especialmente en escenarios con datos heterogéneos y dinámicos, como los que se presentan en las ciudades.

4. Redes Neuronales de Memoria a Largo Plazo (LSTM)

Las redes neuronales LSTM son una variante de las redes neuronales artificiales (ANN) que están diseñadas para manejar secuencias de datos y series temporales. Son especialmente útiles en el contexto de los tiempos de viaje, ya que pueden ajustarse en tiempo real según la información que va llegando.

Las LSTM son muy eficaces para predecir tiempos de traslado basándose en datos secuenciales o temporales. Pueden ajustar sus predicciones continuamente a medida que llegan nuevos datos, lo que las hace altamente dinámicas. Sin embargo, son modelos complejos que requieren una gran cantidad de datos para entrenarse adecuadamente y necesitan una considerable potencia computacional. Además, su proceso de entrenamiento puede ser largo y costoso.

Los autores Ge et al. (2024), utilizaron diferentes variaciones del modelo LSTM para la predicción de horarios de colectivos en entornos urbanos. En primer lugar, implementaron un modelo LSTM estándar, optimizado mediante el Bayesian Optimization Algorithm (BOA), resultando en la variante LSTM-BOA, cuyo objetivo era capturar patrones temporales en los datos y mejorar la precisión de la predicción ajustando hiperparámetros clave. Además, se evaluó una versión Bidireccional LSTM (BiLSTM) optimizada con el Seagull Optimization Algorithm (SOA), que permitía aprender tanto de la información pasada como de la futura dentro de la serie temporal.

Los resultados arrojaron que los modelos LSTM poseen la capacidad para extraer características temporales, y sus predicciones son bastante buenas, pero pueden ser mejoradas aún más en combinación con otros modelos, aumentando la estabilidad y generalización de las predicciones.

5. Gated Recurrent Units (GRU)

Las GRU son una variante de las redes neuronales recurrentes (RNN) que, al igual que las LSTM, son útiles para manejar secuencias de datos y dependencias a largo plazo. Sin embargo, las GRU son más simples y rápidas, ya que utilizan menos puertas de control, lo que las hace menos costosas computacionalmente.

Aunque las GRU son más eficientes que las LSTM en términos de recursos computacionales, pueden no tener el mismo nivel de control sobre el flujo de información. Esto puede hacer que pierdan precisión cuando se enfrentan a secuencias muy complejas o a relaciones a largo plazo en los datos.

Los autores Zhao et al. (2018), proponen un modelo de predicción del tiempo de viaje que emplea un modelo GRU junto con un enfoque de fusión de datos para mejorar la precisión en la predicción de los tiempos de viaje en carreteras. Este modelo integra datos provenientes de diversas fuentes, como sensores de tráfico de microondas y sistemas de peaje electrónicos (ETC), para estimar con mayor precisión el tiempo necesario para atravesar un tramo de carretera.

El modelo GRU fue seleccionado debido a su capacidad para manejar datos secuenciales, mitigando problemas comunes en redes neuronales tradicionales, como la desaparición o explosión de gradientes. Los autores destacan que el uso del GRU es adecuado en situaciones con datos de flujo de tráfico, ya que puede captar las correlaciones temporales y espaciales que afectan el flujo de vehículos.

Los resultados obtenidos en el estudio mostraron que la combinación de datos de sensores con técnicas de fusión, como la fusión de datos de velocidad y volumen de vehículos, mejora significativamente la precisión de las predicciones de tiempo de viaje en comparación con otros métodos tradicionales. Además, al incorporar estos datos en el modelo GRU, se logró predecir con éxito los tiempos de viaje en condiciones variables de tráfico y climatológicas, lo cual es esencial para una planificación eficiente en sistemas de gestión de tráfico.

6. Modelo HMM (Hidden Markov Model)

El Modelo de Markov Oculto (HMM) es un enfoque probabilístico utilizado para modelar sistemas en los cuales el estado real no es observable directamente, pero puede inferirse a partir de variables observadas. En el caso de la predicción de tiempos de llegada en el transporte público, los estados ocultos podrían ser condiciones de tráfico o el comportamiento de los vehículos, mientras que las observaciones serían datos de sensores, como la ubicación GPS o el tiempo de llegada a paradas anteriores.

Los HMM son muy útiles cuando se debe manejar la incertidumbre, ya que permiten integrar diferentes tipos de datos y hacer predicciones en tiempo real, adaptándose a las condiciones cambiantes. Sin embargo, su implementación puede ser compleja, ya que requieren una gran cantidad de datos históricos y son sensibles a datos incorrectos. Además, los HMM asumen que el próximo estado depende sólo del estado actual, lo que puede ser una limitación si existen influencias de largo plazo en las rutas, como eventos en la ciudad o cambios de largo alcance en el comportamiento del tráfico.

Los autores Chen et al. (2016), utilizaron el modelo de Hidden Markov Model (HMM) para abordar el problema de la predicción del tráfico en entornos urbanos. Su estudio partió del reconocimiento de que el tráfico vehicular exhibe patrones temporales y espaciales que pueden ser modelados mediante un enfoque probabilístico. A través del uso de datos históricos de flujo vehicular y velocidades

en distintas intersecciones, los investigadores entrenaron un HMM para identificar estados de congestión y sus transiciones a lo largo del tiempo.

El modelo propuesto clasificó el tráfico en distintos estados, como fluido, moderado y congestionado, determinando probabilidades de transición entre ellos con base en la dinámica del tráfico registrada en días y horarios previos. Para evaluar el rendimiento del modelo, se utilizaron métricas como el error cuadrático medio (MSE) y la precisión de predicción en comparación con enfoques tradicionales basados en promedios históricos y modelos autoregresivos. Los resultados mostraron que el HMM ofrecía una mejora del 15-20% en la precisión de la predicción respecto a los métodos convencionales, especialmente en escenarios con patrones de tráfico recurrentes, como horas pico y días laborales.

Uno de los hallazgos clave del estudio fue que el modelo no solo lograba predecir las condiciones del tráfico con alta precisión a corto plazo, sino que también permitía detectar patrones de congestión emergentes, proporcionando una base sólida para la implementación de estrategias de control del tráfico y planificación urbana. Sin embargo, los autores señalaron que la precisión del modelo se veía afectada en eventos atípicos, como accidentes o desvíos inesperados, sugiriendo que la combinación del HMM con modelos híbridos o datos en tiempo real podría mejorar su desempeño en entornos dinámicos.

Este estudio refuerza la utilidad de los modelos probabilísticos en la predicción del tráfico y sugiere que los HMM pueden ser una herramienta eficaz para optimizar la movilidad en las ciudades, proporcionando información clave para la toma de decisiones en sistemas inteligentes de transporte.

7. Redes Neuronales Recurrentes (RNN)

Las RNN son ampliamente utilizadas en aplicaciones como Google Maps y Moovit para predecir tiempos de llegada, ya que pueden aprovechar patrones secuenciales en los datos de tráfico y condiciones meteorológicas. Aunque las RNN son más flexibles que otros modelos, requieren grandes cantidades de datos para entrenarse adecuadamente y pueden ser difíciles de interpretar.

Los autores Xiao et al. (2023), utilizaron una variante de Redes Neuronales Recurrentes (RNN) combinada con redes convolucionales espaciales para modelar patrones de tráfico en redes de transporte urbano. Implementaron un modelo basado en una Red Neuronal Dinámica Espacio-Temporal (STDN) que incorpora una estructura de memoria para capturar dependencias a largo plazo en las series temporales del tráfico.

El modelo propuesto se enfocó en la predicción del flujo vehicular en distintas ubicaciones de una ciudad, utilizando como datos históricos registros de tráfico, información geoespacial y condiciones climáticas. Los resultados obtenidos

mostraron que STDN supera en precisión a modelos tradicionales como ARIMA y Redes Neuronales Convencionales, logrando una reducción significativa en el error medio absoluto (MAE) y el error cuadrático medio (RMSE), lo que demuestra su capacidad para capturar tanto la variabilidad temporal como la influencia espacial en la predicción del tráfico.

8. K-Nearest Neighbors (K-NN)

El algoritmo K-NN es utilizado en aplicaciones como Beeline en Singapur, donde se busca patrones en los tiempos de llegada basándose en rutas y condiciones anteriores. Aunque es fácil de implementar y funciona bien con conjuntos de datos pequeños, su rendimiento disminuye cuando se trabaja con grandes volúmenes de datos.

Los autores Liu y Wang (2021), exploraron el uso del algoritmo K-Nearest Neighbors (KNN) para predecir los patrones de tráfico en entornos urbanos. El estudio resalta la efectividad de KNN, un modelo de aprendizaje automático simple pero poderoso, para predecir las condiciones de tráfico futuras a partir de datos históricos.

El artículo aborda cómo KNN puede ayudar a gestionar y predecir el flujo de tráfico, lo cual es fundamental para mejorar la movilidad urbana y reducir la congestión. Al analizar los datos de sensores de tráfico, el modelo KNN clasifica y predice estados de tráfico, lo que puede ser utilizado para sistemas de gestión de tráfico en tiempo real. Los autores discuten la aplicación de KNN para predecir diversos parámetros, como la densidad y la velocidad del tráfico, ofreciendo ideas sobre su potencial para optimizar la gestión y planificación del tráfico en las ciudades.

Los resultados del estudio muestran que el modelo KNN para la predicción del flujo de tráfico urbano tiene un tiempo promedio de predicción de 1.3 segundos y una precisión promedio de 91.1%. Este modelo es eficaz para predecir en tiempo real el flujo de tráfico, lo que resulta valioso para la gestión del tráfico urbano y la mejora de la eficiencia en los sistemas de transporte

9. Redes Bayesianas

Las redes bayesianas son útiles para modelar la probabilidad de eventos, como la congestión del tráfico, y pueden actualizar las predicciones a medida que llega nueva información. Sin embargo, su implementación es compleja y requiere un manejo cuidadoso de las probabilidades.

Los autores Vangeneugden et al. (2024), aplicaron un modelo de redes bayesianas para predecir los retrasos de los trenes de pasajeros utilizando datos de otros trenes que circulan en la misma red ferroviaria. El modelo fue diseñado para mejorar la eficiencia de la gestión del transporte ferroviario, aprovechando las relaciones entre

variables como la ocupación de las vías, los tiempos de llegada y los posibles factores externos que podrían influir en el retraso de los trenes.

El enfoque basado en redes bayesianas es adecuado para este tipo de problemas, ya que permite manejar la incertidumbre y las relaciones de dependencia entre diferentes factores, lo que resulta crucial en un sistema tan complejo como el ferroviario. En el estudio, se utilizó un conjunto de datos históricos de retrasos en los trenes, que permitió a las redes bayesianas aprender patrones y hacer predicciones sobre la probabilidad de retrasos en tiempo real. Los resultados obtenidos demostraron que este modelo tiene una alta precisión para predecir los retrasos y que puede actualizar sus predicciones de manera dinámica conforme se reciben nuevos datos.

En cuanto a las métricas de desempeño, el modelo alcanzó una alta precisión en las predicciones de los retrasos, con un índice de aciertos del 92% en comparación con métodos tradicionales basados en regresión. Esta cifra subraya la efectividad del modelo para manejar la variabilidad inherente al tráfico ferroviario. Por otro lado, el tiempo de procesamiento para realizar las predicciones en tiempo real se mantuvo bajo, lo que facilita la implementación del sistema en entornos operativos donde la rapidez es esencial para la toma de decisiones.

Los resultados del estudio muestran que las redes bayesianas son una herramienta potente para la predicción de retrasos en el transporte ferroviario, y su aplicación podría extenderse a otros tipos de transporte, como el colectivo urbano, donde las variables de tráfico, condiciones meteorológicas y otros factores también afectan la puntualidad del servicio. Este enfoque, que incorpora tanto la flexibilidad en la modelización como la capacidad de actualización dinámica, ofrece un gran potencial para mejorar la eficiencia en la gestión de sistemas de transporte en tiempo real.

Resumen

Cada uno de estos enfoques tiene sus ventajas y limitaciones, dependiendo de la complejidad de los datos y las condiciones del entorno. A continuación, a modo de resumen, se presentará la Tabla 1 con una comparación de las técnicas mencionadas, destacando sus características principales, las aplicaciones y sus respectivas fortalezas y debilidades en función de los diferentes escenarios.

Técnica	Papers	Predicciones	Ventajas	Desventajas
Regresión Lineal	Cardona et al. (2015)	Estimar los tiempos de llegada de los buses en un sistema de transporte masivo en la ciudad de Pereira, Colombia	-Fácil de implementar y entender -Requiere poca potencia computacional -Buen rendimiento en escenarios sencillos o donde la relación entre las variables es lineal	-Puede no capturar bien relaciones no lineales -No maneja bien la complejidad del tráfico urbano o cambios dinámicos como accidentes o condiciones climáticas
ARIMA	Suwardo et al. (2010)	Tiempos de viaje de colectivos en el corredor Ipoh-Lumut, en Malasia	-Bien adaptado a los datos que varían con el tiempo y que presentan patrones cíclicos -Puede incorporar estacionalidad es (patrones repetidos en el tiempo)	-Puede ser difícil de ajustar si los datos tienen muchas fluctuaciones abruptas o no siguen patrones estables -No es efectivo con datos inconsistentes o con factores externos que afectan el tiempo de traslado
Random Forest	Li (2017)	Tiempo de llegada de los colectivos utilizando datos históricos de ubicación GPS, condiciones de tráfico, datos meteorológicos, y la hora del día	-Es robusto y maneja bien las variables con relaciones no lineales y complejas -Puede manejar grandes volúmenes de datos sin perder precisión	-Requiere mayor capacidad computacional -Es más difícil de interpretar debido a su naturaleza de "caja negra"

LSTM	Ge et al. (2024)	Horarios de colectivos en entornos urbanos	-Excelente para secuencias de datos y series temporales, lo que permite adaptarse en tiempo real a nuevos datos -Muy preciso en la predicción de tiempos de viaje	-Requiere gran cantidad de datos y potencia computacional para entrenarse
GRU	Zhao et al. (2018)	Tiempos de viaje en carreteras	-Buen rendimiento en datos secuenciales -Estructura más simple	-Puede perder precisión en secuencias muy complejas -Menor control sobre el flujo de información que LSTM
HMM	Chen et al. (2016)	Tráfico vehicular en entornos urbanos	-Manejo de incertidumbre -Pueden integrar diferentes tipos de datos y hacer predicciones en tiempo real	-Requieren grandes cantidades de datos históricos -Complejos de implementar y mantener -Sensibilidad a datos incorrectos y factores externos a largo plazo
RNN	Xiao et al. (2023)	Tráfico en redes de transporte urbano	-Más flexibles que los HMM para capturar relaciones complejas y no lineales entre variables temporales	-Requieren grandes cantidades de datos y poder computacional -Pueden ser difíciles de interpretar

K-NN	Liu y Wang (2021)	Patrones de tráfico en entornos urbanos	-Fácil de implementar -Funciona bien con conjuntos de datos pequeños	-No escala bien con grandes cantidades de datos -Puede ser lento en tiempo real
Redes Bayesianas	Vangeneugden et al. (2024)	Retrasos de los trenes de pasajeros	-Capacidad de incorporar incertidumbre y manejar información incompleta	-Complejidad en la implementación y la actualización de las probabilidades a medida que llegan datos nuevos

Tabla 1 Comparación de las Técnicas de Estimación de Tiempos de Traslado.

Capítulo 3: Análisis y Procesamiento de los Datos

En este capítulo se aborda el análisis detallado de los datos de posicionamiento de los colectivos, con el objetivo de comprender su distribución, detectar patrones de circulación y evaluar posibles irregularidades que puedan influir en la estimación de tiempos de llegada. El volumen y la calidad de los datos son aspectos clave en el desarrollo de sistemas de predicción, ya que cualquier inconsistencia, datos faltantes o valores anómalos pueden afectar la precisión de los cálculos y, en consecuencia, la experiencia del usuario final.

Para este análisis, se dispone de un conjunto de datos conformado por aproximadamente 4 millones de registros correspondientes a seis líneas de colectivos de la ciudad de Tandil (500, 501, 502, 503, 504 y 505). Cada línea cuenta con alrededor de 750 mil registros, distribuidos en múltiples archivos, donde cada fila almacena información clave como el identificador del colectivo, la marca de tiempo de la solicitud y la respuesta, la velocidad registrada y la posición geográfica (latitud y longitud). Cabe destacar que toda esta información fue recopilada durante los meses de octubre y noviembre del año 2017 de la vieja aplicación SUMO que permitía observar las posiciones de los colectivos en tiempo real, donde se estuvieron guardando los datos correspondientes a las distintas unidades de cada línea de colectivo.

El análisis se centra en la validación y limpieza de los datos mediante diversas técnicas de procesamiento. En primer lugar, se analiza la cobertura temporal para verificar que los datos abarcan distintos momentos del día y diferentes días de la semana, incluyendo horarios pico y fines de semana. Luego, se examina la distribución de registros por línea y unidad, identificando posibles desequilibrios en la recolección de datos. Posteriormente, se detectan y manejan valores nulos, duplicados y registros con valores atípicos, con el fin de garantizar la fiabilidad del conjunto de datos y se explora la distribución de velocidades para evaluar si los valores capturados reflejan un comportamiento realista del tránsito de los colectivos. Finalmente se procede a mostrar cómo fueron filtrados los datos para dejar un dataset lo más depurado posible.

El resultado de este análisis permitirá determinar la calidad de los datos disponibles y su idoneidad para la implementación de modelos de predicción de tiempos de llegada, sentando así las bases para el desarrollo de una herramienta confiable y eficiente para los usuarios del transporte público.

3.1 Cobertura Temporal

El análisis de la cobertura temporal es crucial para evaluar la representatividad de los datos recolectados y su idoneidad para la estimación de tiempos de llegada. Para garantizar que el modelo de predicción pueda adaptarse a diferentes momentos del día y días de la semana, es necesario verificar que la información recolectada abarque una distribución equitativa en horarios clave, como las horas pico, los horarios nocturnos y los fines de semana.

El conjunto de datos utilizado se compone de distintos archivos por cada una de las seis líneas de colectivos analizadas (500, 501, 502, 503, 504 y 505), con una cantidad variable de registros dependiendo de la cantidad de unidades en circulación. Cada registro cuenta con el identificador del colectivo, las marcas de tiempo de la solicitud y del registro de posición, la velocidad y la ubicación geográfica. En total, se cuenta con alrededor de 750 mil registros por línea, sumando aproximadamente 4 millones de registros en el dataset completo.

El primer paso en este análisis consistió en verificar que la distribución de datos a lo largo del día refleje el comportamiento real del sistema de transporte. Se observó que la mayoría de los registros se concentran en las horas pico, lo que sugiere que la recopilación de datos sigue un patrón lógico en relación con la demanda del servicio, como se puede observar en la Figura 4.

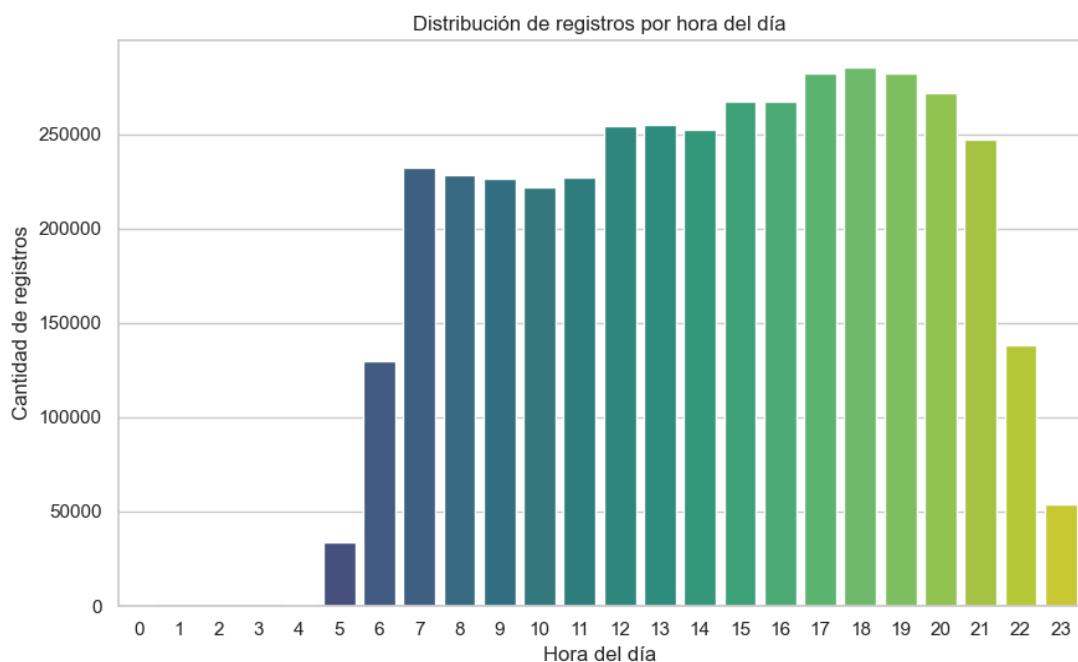


Figura 4 Gráfico de la Cantidad de Registros para cada Hora del Día.

Las distribuciones de hora parecen muy acertadas, correspondientes al rango horario en que trabajan los colectivos. Sin embargo, se identificaron algunas

anomalías en los extremos del día, particularmente alrededor de las 5:00 y las 23:00 horas, lo que podría estar relacionado con retrasos o adelantos en los recorridos de los colectivos.

Asimismo, se analizó la distribución de registros a lo largo de la semana. En términos generales, los datos parecen estar bien equilibrados, con la excepción de un incremento inesperado los días miércoles. Esta variación podría atribuirse a una mayor cantidad de unidades en circulación en esos días particulares debido a que podrían necesitar más al tratarse exactamente del medio de la semana. Durante los fines de semana, la cantidad de registros disminuye, lo cual es esperable debido a la reducción en la frecuencia de los servicios de transporte público en esos días.

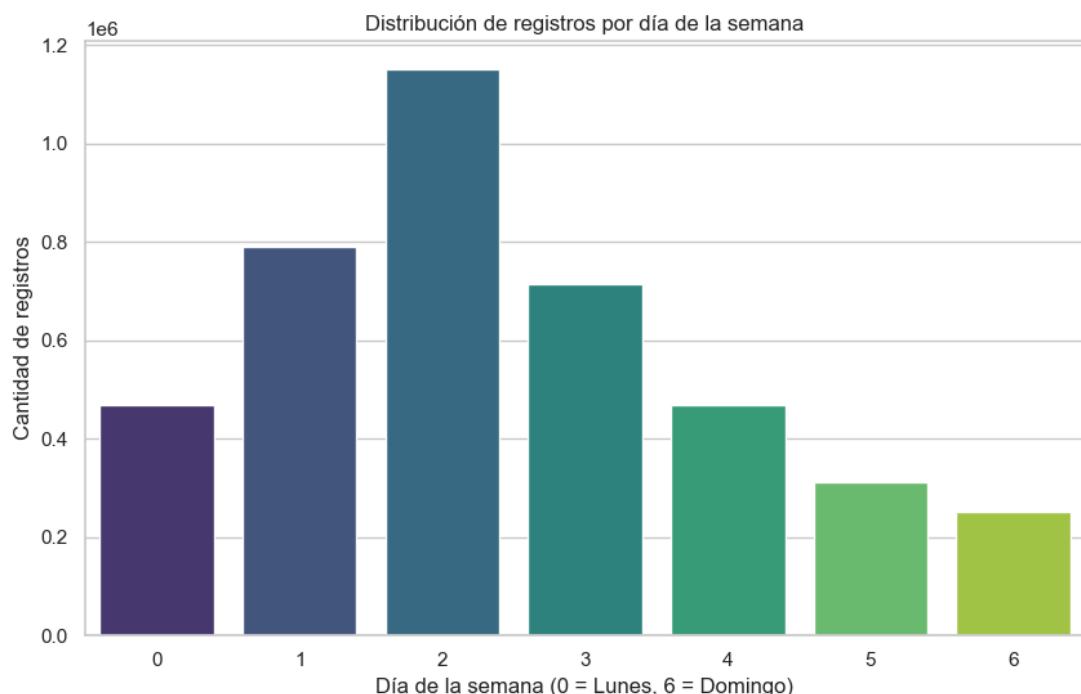


Figura 5 Gráfico de la Cantidad de Registros para cada Día de la Semana.

Este análisis resulta relevante debido a que una distribución inadecuada de los datos temporales podría afectar la precisión del modelo de predicción, especialmente en situaciones donde los patrones de tráfico varían significativamente según la hora y el día.

3.2 Distribución por Línea y Unidad

El análisis de la distribución de datos por línea y por unidad de colectivo permite evaluar si la información recolectada es equitativa entre los distintos servicios de transporte y si refleja de manera adecuada la operación de cada unidad. Una distribución equilibrada es importante para evitar sesgos en el modelo de predicción y garantizar que todas las líneas y colectivos sean representados correctamente.

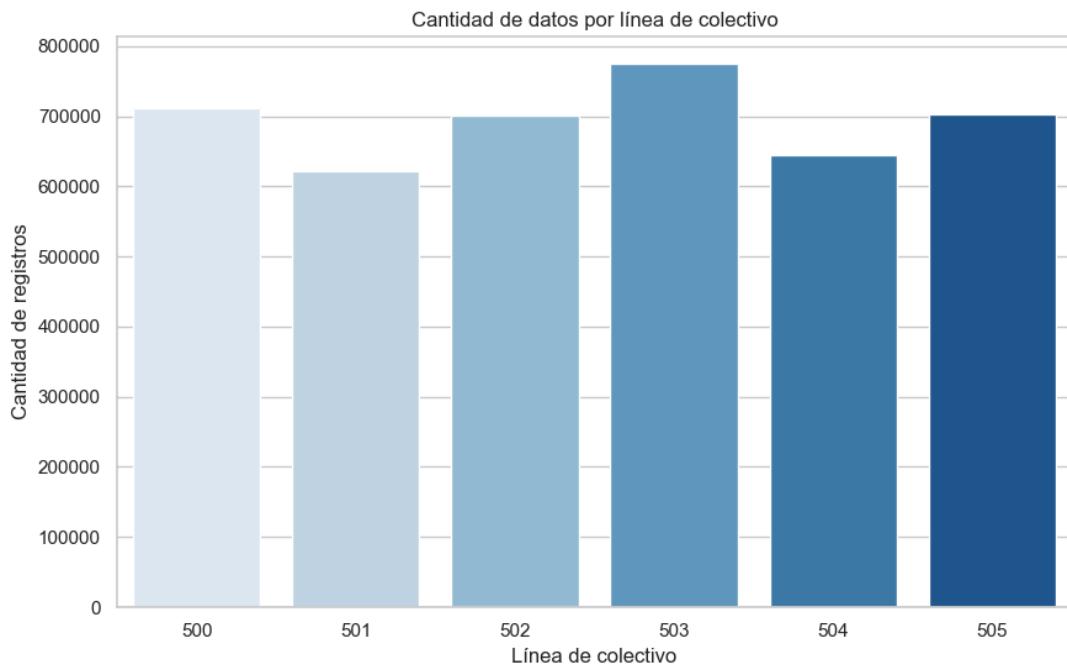


Figura 6 Gráfico de la Cantidad de Registros por Línea de Colectivo.

Como se puede observar en la Figura 6, en términos generales, la cantidad de datos por línea de colectivo se encuentra bien distribuida, sin variaciones significativas entre ellas, siendo la Línea 503 la que más datos posee, lo cual es consecuente considerando su amplio recorrido y alta demanda por parte de la población de Tandil. Esto indica que la recopilación de datos se ha realizado de manera uniforme, permitiendo que el análisis de cada línea sea representativo de su comportamiento real.

Sin embargo, al observar la distribución de datos por unidad dentro de cada línea, se identifican variaciones más pronunciadas. Algunas unidades poseen una mayor cantidad de registros en comparación con otras, lo que puede deberse a diferencias en la frecuencia de circulación.

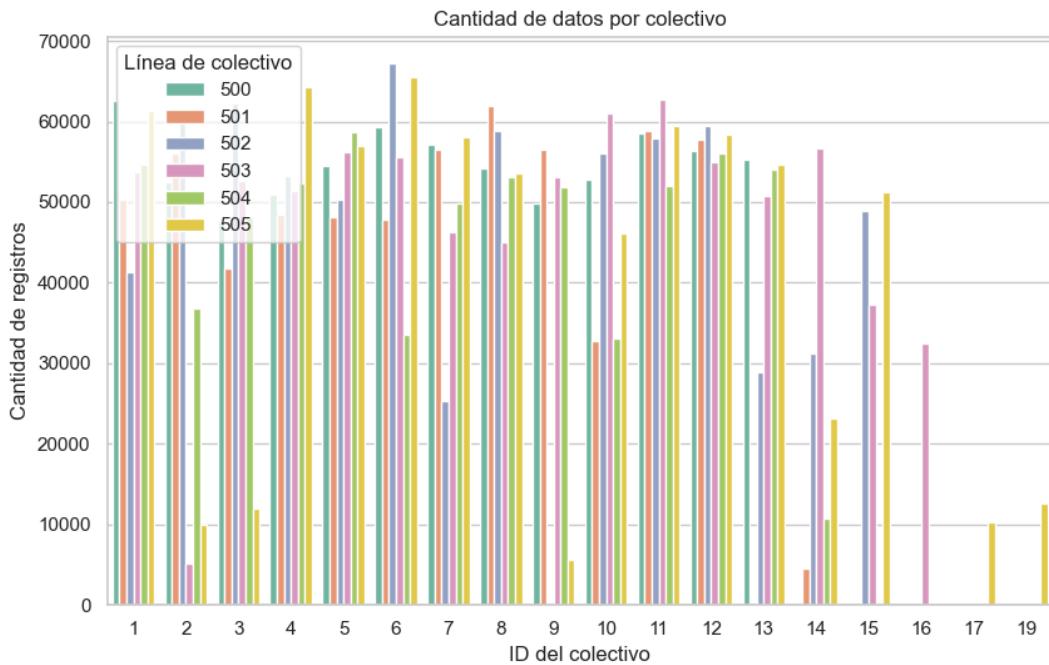


Figura 7 Gráfico de la Cantidad de Registros por Unidad de Colectivo.

A pesar de estas diferencias, lo más relevante para el análisis es la información agregada de cada línea, ya que el comportamiento individual de una unidad específica no impactaría significativamente en el rendimiento global del modelo.

Es importante mencionar que, si bien una distribución más homogénea por unidad sería ideal, la variabilidad observada es esperable en un sistema de transporte público, donde ciertos colectivos pueden operar más tiempo o en rutas más transitadas.

3.3 Nulos y Duplicados

La presencia de valores nulos y duplicados en un dataset puede afectar la calidad del análisis y la precisión de cualquier modelo basado en estos datos. Identificar y manejar estos valores es un paso fundamental en el proceso de limpieza de datos.

En este caso, se verificó que no existen valores nulos en el conjunto de datos. Esto es una ventaja significativa, ya que evita la necesidad de imputaciones o estrategias de manejo de datos faltantes, que podrían introducir sesgos o errores en el análisis.

Por otro lado, se detectó la presencia de registros duplicados, considerando que esta condición se cumple cuando dos datos poseen la misma línea, unidad, marca de tiempo (tanto de la solicitud como de la información particular del colectivo) y coordenadas. En promedio, cada línea de colectivo cuenta con alrededor de 700 mil registros, de los cuales aproximadamente 40 mil son duplicados (alrededor del 5% del total), como puede observarse en la Figura 8.

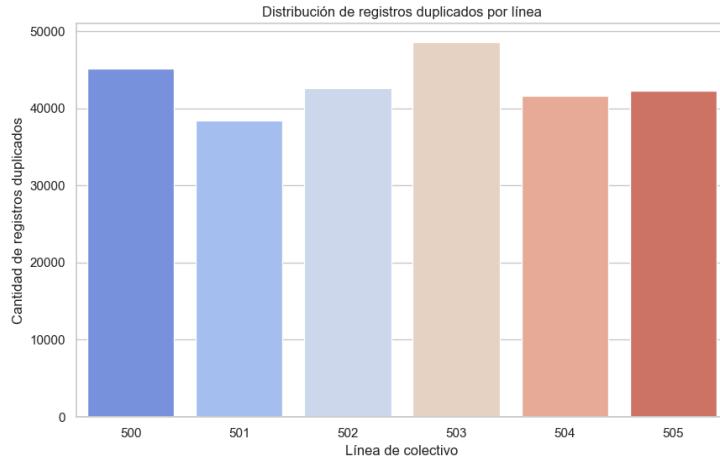


Figura 8 Gráfico de la Cantidad de Registros Duplicados por Línea de Colectivo.

Los registros duplicados pueden generarse por errores en la recopilación de datos, transmisiones repetidas o eventos en los que una misma posición fue capturada varias veces en un corto intervalo de tiempo. Dado que los mismos no aportan nueva información y pueden sesgar el análisis, se considera que su eliminación no afecta significativamente la calidad del dataset. Su depuración permitió mejorar la eficiencia del procesamiento y garantizar que el análisis refleje de manera más precisa el comportamiento real de los colectivos.

3.4 Distribución de Velocidades

El análisis de la distribución de velocidades es clave para evaluar el comportamiento de los colectivos en distintos escenarios y para detectar posibles inconsistencias en los datos. La velocidad de un colectivo puede verse afectada por múltiples factores, como el tipo de vía, la densidad del tráfico, la presencia de semáforos y las paradas establecidas en la ruta.

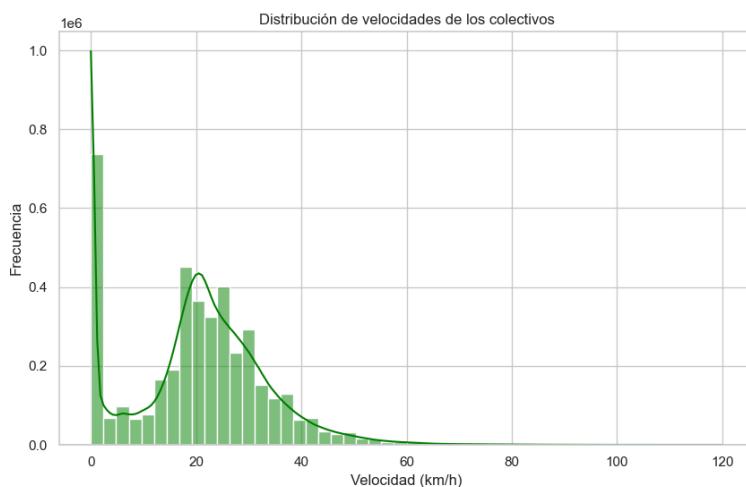


Figura 9 Gráfico de la Distribución de Velocidades de los Colectivos.

Los datos muestran que las velocidades registradas siguen un patrón lógico y esperado en la mayoría de los casos. Sin embargo, la velocidad máxima observada es de aproximadamente 120 km/h, aunque estas mediciones terminan siendo una cantidad mínima de datos, pudiendo deberse a excepciones o errores en las mediciones. Estos valores serán analizados más adelante para corroborar su importancia. Por otro lado, el máximo con una cantidad relevante de valores resulta en 60 km/h, lo que es un valor normal considerando que pueden ser momentos en que los colectivos circulan por avenidas o tramos con menor congestión vehicular. La velocidad media se encuentra en torno a los 25 km/h, lo que es consistente con la operación normal de colectivos en entornos urbanos.

También se detectan múltiples registros donde la velocidad es igual a cero, lo cual es coherente con la existencia de semáforos, paradas y otras interrupciones del tránsito. Estas pausas forman parte del comportamiento habitual de los colectivos y deben ser consideradas en el modelo de predicción para evitar errores en la estimación de tiempos de llegada.

La validez de estos datos de velocidad refuerza la fiabilidad del dataset y su utilidad para el análisis. Además, la distribución de velocidades permite inferir patrones de tránsito y evaluar si ciertos recorridos presentan mayores niveles de congestión, lo que podría ser útil para futuras optimizaciones del servicio de transporte.

3.5 Filtrado de Datos por Recorrido

Durante el análisis de los datos de velocidad, se detectaron valores atípicos que alcanzaban hasta 120 km/h, lo cual no es realista para un colectivo urbano. Al revisar las coordenadas de estos registros, se encontró que muchos de estos puntos se ubicaban fuera del recorrido habitual de las líneas de colectivos, más específicamente correspondían a zonas de ruta.

Para visualizar mejor esta anomalía, se generaron mapas con la distribución de los puntos registrados para cada línea y ahí se pudo observar que estos registros con valores altos de velocidad no eran los únicos erróneos, sino que había muchos más cuyas coordenadas caían fuera de los recorridos. En el caso de la línea 500, se observó que una gran cantidad de datos estaban en ubicaciones que no forman parte del recorrido oficial de la misma, lo cual se puede observar en la Figura 10.



Figura 10 Ubicación de los Registros de la Línea 500 sin Filtrar.

Esto puso en evidencia la necesidad de realizar un proceso de filtrado de datos, descartando aquellos registros que no correspondan al trayecto real de los colectivos.

Tras aplicar el filtrado, se obtuvo una distribución mucho más precisa y alineada con los recorridos reales. Es importante señalar que algunos segmentos del recorrido no cuentan con registros debido a que los datos utilizados corresponden a un período anterior (año 2017), cuando ciertas modificaciones en los trayectos aún no se habían implementado, por ejemplo el tramo por Fuerte Independencia que antes se hacía por la 9 de Julio.

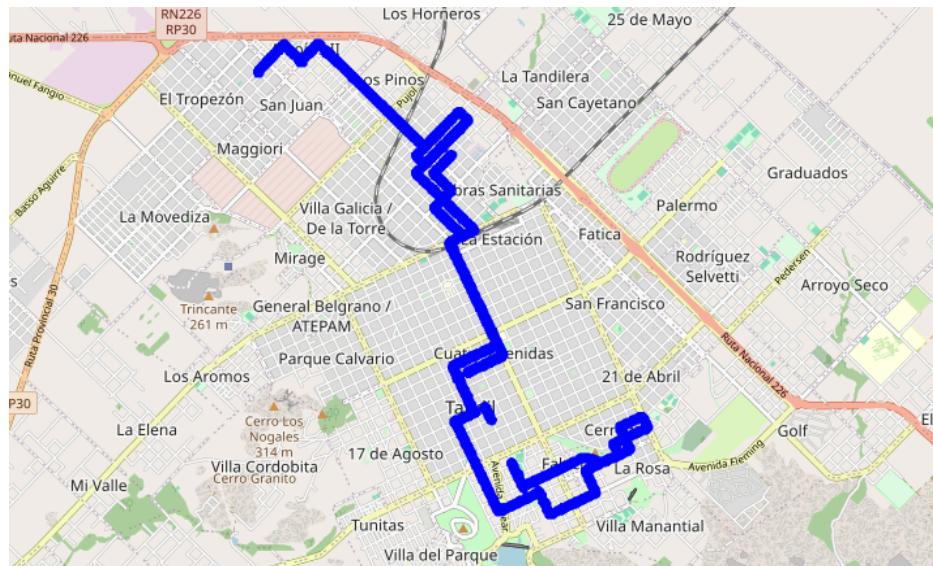


Figura 11 Ubicación de los Registros de la Línea 500 Luego de Filtrar.

Este proceso de filtrado se efectuó para cada una de las líneas de colectivos, lo que resultó computacionalmente costoso. Se evaluaron los 4 millones de registros, comparando cada uno con aproximadamente 500 segmentos de recorrido (definidos a partir de 1,000 puntos de referencia por línea). En total, se realizaron alrededor de 2 mil millones de comparaciones, lo que demandó casi una semana de procesamiento.

Este filtrado fue esencial para garantizar que los datos utilizados en el análisis sean representativos del servicio real y evitar distorsiones en los cálculos de tiempos de espera y estimaciones de llegada.

Capítulo 4: Diseño e Implementación de los Modelos de Predicción

En este capítulo se describe detalladamente el proceso de diseño e implementación de los modelos de predicción utilizados para estimar los tiempos de traslado dentro del sistema de transporte público de la ciudad. El objetivo principal es desarrollar un modelo que permita calcular de manera eficiente y precisa el tiempo necesario para desplazarse entre dos puntos específicos.

Para ello, es fundamental definir claramente los datos disponibles y los que se necesitan obtener, así como evaluar las estrategias más adecuadas para el procesamiento y análisis de dicha información. Se tomaron en cuenta diferentes enfoques y se evaluaron sus ventajas y limitaciones en el contexto de los datos recopilados.

Tras el relevamiento de las diversas técnicas de predicción de tiempos de traslado, se decidió diseñar, implementar y evaluar cuatro tipos de modelos principales, los cuales son Regresión Lineal, ARIMA, LSTM y GRU. Estos modelos fueron seleccionados en función de su adecuación a los datos disponibles y su potencial para ofrecer estimaciones confiables dentro del contexto de esta aplicación. Además, para cada tipo fue necesario procesar, filtrar e interpretar los datos de forma particular, pero esto se detalla más adelante.

En las distintas secciones correspondientes a cada uno de los modelos implementados se mencionan métricas como MAE, MSE, RMSE y R^2 . Las mismas serán utilizadas para evaluar el rendimiento de los distintos modelos (los dos últimos únicamente para la regresión), pero para poder entender qué significan estas métricas y porque un modelo resulta mejor que otro es necesario definirlas.

El Error Absoluto Medio (MAE) mide la diferencia promedio en valor absoluto entre los valores reales y los valores predichos. Es una métrica intuitiva y fácil de interpretar, ya que expresa el error en las mismas unidades que la variable objetivo. Su principal característica es que trata todos los errores de manera uniforme, sin amplificar los más grandes, lo que lo hace menos sensible a valores atípicos en comparación con otras métricas.

El Error Cuadrático Medio (MSE) es el promedio de los errores elevados al cuadrado. A diferencia del MAE, esta métrica penaliza con mayor severidad los errores grandes debido a la cuadratura de las diferencias. Aunque esto puede ser útil en ciertos contextos donde los errores grandes son particularmente perjudiciales, también implica que el MSE es más sensible a valores atípicos, ya que un solo error grande puede aumentar significativamente su valor.

Para mitigar la falta de interpretabilidad del MSE en términos de las unidades originales de la variable objetivo, se emplea la Raíz del Error Cuadrático Medio (RMSE), que se obtiene al calcular la raíz cuadrada del MSE. De esta manera, el RMSE mantiene la ventaja de penalizar más los errores grandes pero, a diferencia del MSE, se expresa en las mismas unidades que la variable a predecir, lo que facilita su interpretación.

Por otro lado, el Coeficiente de Determinación (R^2) mide qué proporción de la variabilidad total de los datos es explicada por el modelo. Su valor oscila generalmente entre 0 y 1, donde un valor cercano a 1 indica que el modelo es capaz de explicar la mayor parte de la variabilidad de la variable dependiente, mientras que un valor cercano a 0 sugiere que el modelo no logra capturar la relación entre las variables. En algunos casos, cuando el modelo tiene un desempeño peor que simplemente predecir el valor promedio de los datos, el R^2 puede ser negativo.

4.1 Modelos de Regresión Lineal

El primer enfoque consistió en la implementación de un modelo de regresión lineal simple, donde la variable de entrada considerada fue la distancia entre los puntos de origen y destino, y la variable de salida el tiempo estimado de viaje. Si bien al momento de entrenar el modelo los datos estaban todos juntos, el testeo se consideró hacerlo en ambos casos, todo junto y por línea de colectivo. Los resultados obtenidos fueron deficientes en todos los casos, con un R^2 menor a 0.1 para todas las líneas testeadas, lo que indica que el modelo no pudo relacionar las variables y evidencia que una sola variable no es suficiente para capturar la complejidad del sistema de transporte público, por lo que rápidamente se abandonó este modelo simple inicial.

Como solución natural a este problema, se consideró la extensión hacia un modelo de regresión lineal múltiple, incorporando variables adicionales como la hora del día, la velocidad, el día de la semana, entre otras. Sin embargo, el principal desafío radicó en la transformación y procesamiento de los datos para obtener una representación adecuada de las variables de entrada y salida.

A partir de los datos procesados, analizados y filtrados anteriormente, se decidió calcular el tiempo de viaje entre pares de puntos geográficos para cada línea y unidad de colectivo dentro de un mismo día y en franjas horarias específicas. Este enfoque generó un incremento exponencial en el tamaño del dataset, lo que dificultó su procesamiento y prueba en un entorno realista. En un rango horario se compararon de a pares todos los puntos correspondientes a una línea, unidad, día y rango horario particular para saber el tiempo entre sí. Esto resultó en más de 8 mil puntos para cada cuarteto pero llevándolo a todas las unidades de todas las líneas en todos los días y horarios posibles durante el periodo que se recolectaron datos son millones y millones de registros.

Por ello se probó con un subconjunto pequeño de los datos, más específicamente unos 10 mil registros que corresponden a una línea, unidad, en un rango horario y día particulares, obteniendo las métricas de las Tabla 2 para cada línea correspondiente. Cabe aclarar que para los resultados se eligió mostrar el dia 15/10/2017, correspondiente al primer dia que se tiene registros, en el horario entre 11 y 13 horas, debido a que cayó domingo y los colectivos se suponen presentarían una regularidad a lo largo del recorrido de sus rutas sin mucha interferencia de factores externos como demasiado tránsito o mucha demanda del servicio.

Línea	Error Absoluto Medio (MAE)	Error Cuadrático Medio (MSE)	Raíz del Error Cuadrático Medio (RMSE)	Coeficiente de Determinación (R^2)
500	342.359351	250552.155258	500.551850	0.718487
501	443.820663	352886.777370	594.042740	0.527775
502	604.023884	649139.717564	805.692073	0.222906
503	177.528587	85371.110123	292.183350	0.880779
504	816.701797	1070092.778977	1034.452888	0.073771
505	443.513887	339034.764539	582.266918	0.689960

Tabla 2 Métricas del Modelo de Regresión Lineal para Tiempo entre Puntos.

Para la línea 500, el MAE de 342.36 segundos indica un error promedio de aproximadamente 5 minutos y 42 segundos en las predicciones. El RMSE de 500.55 segundos refuerza este resultado, sugiriendo que los errores son moderados pero con cierta presencia de valores atípicos. Con un R^2 de 0.7185, el modelo logra explicar aproximadamente el 71.85% de la variabilidad en los datos.

Para la línea 501, el error aumenta con un MAE de 443.82 segundos (7 minutos y 24 segundos), mientras que el RMSE de 594.04 segundos sugiere que los errores pueden ser aún mayores en algunos casos. Con un R^2 de 0.5278, el modelo explica solo el 52.78% de la variabilidad, indicando una menor capacidad predictiva.

En la línea 502, el MAE de 604.02 segundos (10 minutos y 4 segundos) y el RMSE de 805.69 segundos revelan una gran dispersión en los errores de predicción. Además, el R^2 de 0.2229 indica que el modelo apenas explica el 22.29% de la variabilidad, lo que sugiere que otros factores no considerados podrían estar afectando el tiempo entre puntos.

Por otro lado, la línea 503 muestra el mejor desempeño, con un MAE de 177.52 segundos (2 minutos y 57 segundos) y un RMSE de 292.18 segundos. Su R^2 de 0.8808 sugiere que el modelo logra capturar el 88.08% de la variabilidad, lo que indica una alta precisión en las predicciones.

Para la línea 504, el desempeño es el peor entre todas las líneas analizadas, con un MAE de 816.70 segundos (13 minutos y 37 segundos) y un RMSE de 1034.45 segundos. Su R^2 de 0.0738 indica que el modelo prácticamente no explica la variabilidad de los datos, lo que sugiere que la relación entre las variables no es lineal o que faltan factores clave en el análisis.

La línea 504 presenta un comportamiento intermedio, con un MAE de 443.51 segundos (7 minutos y 23 segundos) y un RMSE de 582.26 segundos. Su R^2 de 0.68996 indica que el modelo es relativamente confiable, explicando el 68.99% de la variabilidad.

En conclusión, el modelo tiene un desempeño variable según la línea analizada. Aunque hay valores razonables de R^2 para algunas líneas, los valores relativamente altos de MAE y RMSE sugieren que los errores aún son considerables en términos prácticos, y para algunas líneas los resultados son significativamente malos, como la 502 o la 504. Todo esto sumado a la inviabilidad del tamaño de los datos resultantes sugirió explorar alternativas, siendo la más viable en lugar de predecir directamente los tiempos de viaje, estimar velocidades promedio segmentadas por horario y zona de la ciudad. Con esta información, el cálculo del tiempo podría realizarse utilizando la distancia recorrida en cada zona con la velocidad que se obtiene en ese punto.

El nuevo enfoque de predecir velocidades se decidió implementar como un solo modelo donde la línea y unidad sean variables del mismo, ya que se podría suponer que la velocidad depende más de la zona geográfica que del recorrido en sí. Como consecuencia se utilizaron unos 2.2 millones de datos de entrenamiento y casi 1 millón de testeo divididos de forma aleatoria siguiendo una relación 70/30. Sin embargo, los resultados obtenidos con este segundo enfoque fueron desalentadores, empeorando incluso los obtenidos anteriormente.

Línea	Error Absoluto Medio (MAE)	Error Cuadrático Medio (MSE)	Raíz del Error Cuadrático Medio (RMSE)	Coeficiente de Determinación (R^2)
500	8.827606	125.417621	11.199000	0.101150
501	8.857301	128.901301	11.353470	0.216973
502	10.311423	169.371038	13.014262	0.097564
503	9.917830	156.084707	12.493386	0.194672

504	9.336566	141.159682	11.881064	0.084679
505	8.963347	135.187654	11.627022	0.187193
Promedio	9.013509	134.425592	11.594205	0.237027

Tabla 3 Métricas del Modelo de Regresión Lineal para Velocidad.

Considerando que ahora la salida es en kilómetros por hora, el MAE promedio de 9.01 presente en la Tabla 3 indica que, las predicciones del modelo tienen un error de aproximadamente 9 km/h respecto a los valores reales. Esto significa que, en términos generales, las estimaciones del modelo pueden desviarse de la velocidad real en esa magnitud.

El MSE de 134.43 km^2/h^2 representa el promedio de los errores elevados al cuadrado. Dado que esta métrica amplifica los errores grandes, su interpretación directa no es intuitiva, pero su valor relativamente alto sugiere la presencia de predicciones con errores significativos. Al igual que antes, para mejorar la interpretación, observamos el RMSE de 11.59 km/h, que indica que el error promedio de las predicciones es aproximadamente 11.6 km/h. Como el RMSE es mayor que el MAE, nuevamente esto sugiere que hay errores grandes que están afectando la precisión del modelo.

El R^2 de 0.2370 indica que el modelo solo explica aproximadamente el 23.7% de la variabilidad en los datos de velocidad. Este es un valor bajo, lo que sugiere que el modelo no logra capturar adecuadamente la relación entre las variables de entrada y la velocidad. Un R^2 tan bajo generalmente implica que el modelo no es fiable y que probablemente haya margen de mejora en su diseño.

La línea 500 tiene el menor error absoluto y cuadrático medio, lo que sugiere que el modelo de regresión lineal funciona mejor para predecir su velocidad, aunque tampoco es un resultado bueno al fin y al cabo, solo mejor que el resto. Por otro lado, la línea 502 tiene los errores más altos en MAE, MSE y RMSE, lo que indica que la predicción es menos confiable para esta línea. Además, ninguna de las líneas tiene un R^2 alto, lo que sugiere que la regresión lineal no es el mejor modelo para predecir la velocidad de los colectivos.

En conclusión, el modelo presenta errores relativamente altos en comparación con la escala de la variable de salida, y su bajo R^2 indica que no está explicando bien los datos. Por esta razón, se definió que el enfoque basado en la regresión lineal no era adecuado para el problema planteado, y se decidió explorar alternativas más sofisticadas para mejorar la precisión de las predicciones.

4.2 Modelo ARIMA

ARIMA es un modelo ampliamente utilizado en series temporales debido a su capacidad para capturar tendencias y patrones recurrentes en los datos, cuya efectividad depende de la estructura y regularidad de la serie temporal utilizada. Para poder utilizar este modelo para predecir el tiempo entre dos puntos del recorrido, los datos deben estar en una forma que represente una serie temporal. Esto significa que deben ser ordenados cronológicamente y reflejar una métrica (en este caso, el tiempo entre dos puntos) a intervalos regulares o predecibles. En lo que respecta a la regularidad, si los mismos son irregulares, es decir, con puntos dispersos a lo largo del día y sin un patrón claro de paradas, las predicciones pueden no ser tan efectivas debido a que el modelo no está viendo un comportamiento "predecible" en la secuencia temporal.

Considerando este aspecto, se procesó el conjunto de datos utilizando un patrón temporal regular, seleccionando únicamente los registros con intervalos de un minuto. A partir de estos registros, se construyeron las series temporales para cada línea de colectivo, unidad y fecha particular. La estrategia consistió en generar un modelo por línea de colectivo, donde la posición registrada cada un minuto serviría como base para estimar la ubicación en los siguientes n pasos (minutos).

Debido a que el modelo ARIMA solo predice una variable lo que se tuvo que hacer es ajustar dos modelos simultáneos con las mismas series temporales, uno que prediga latitud y otro longitud. La particularidad de este tipo de modelos es que poseen ciertos parámetros p , d y q los cuales son el número de retardos en el componente autorregresivo (AR), el número de diferencias requeridas para estacionalizar la serie y el número de retardos en el componente de media móvil (MA) respectivamente. Por lo tanto entrenar el modelo implica ajustar estos parámetros a los que más se ajusten a la serie, y para ello se utilizó la librería pmdarima y el módulo auto_arima el cual busca los más convenientes. Luego de probar modelos para cada línea se observaron que las predicciones de los siguientes 10 pasos eran muy malas, no cayendo ni siquiera en lugares dentro del recorrido respectivo. En las figuras 12 a 17 presentadas a continuación se observan en color rojo las 10 predicciones hechas para cada serie temporal. Cabe aclarar que una serie temporal son los registros de un día particular, pero estos malos resultados se repiten para cualquier día que se prueben los modelos.

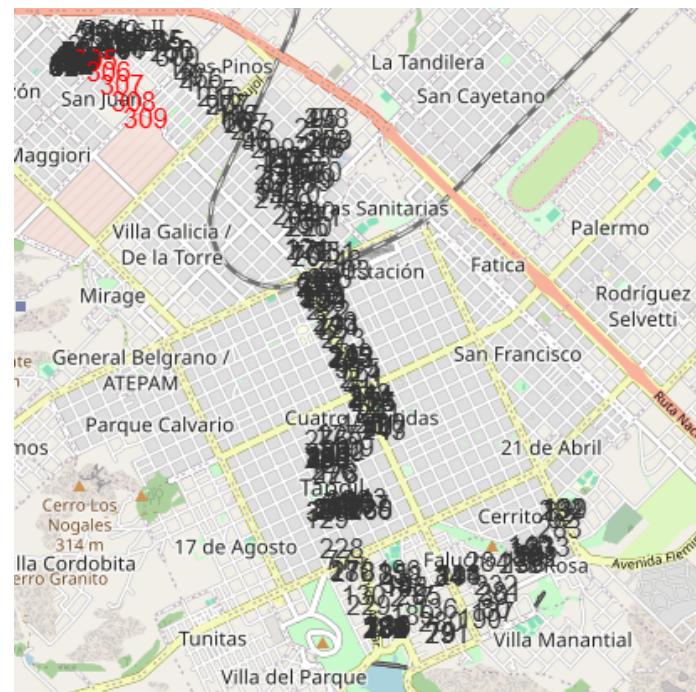


Figura 12 Predicciones del Modelo ARIMA Línea 500.

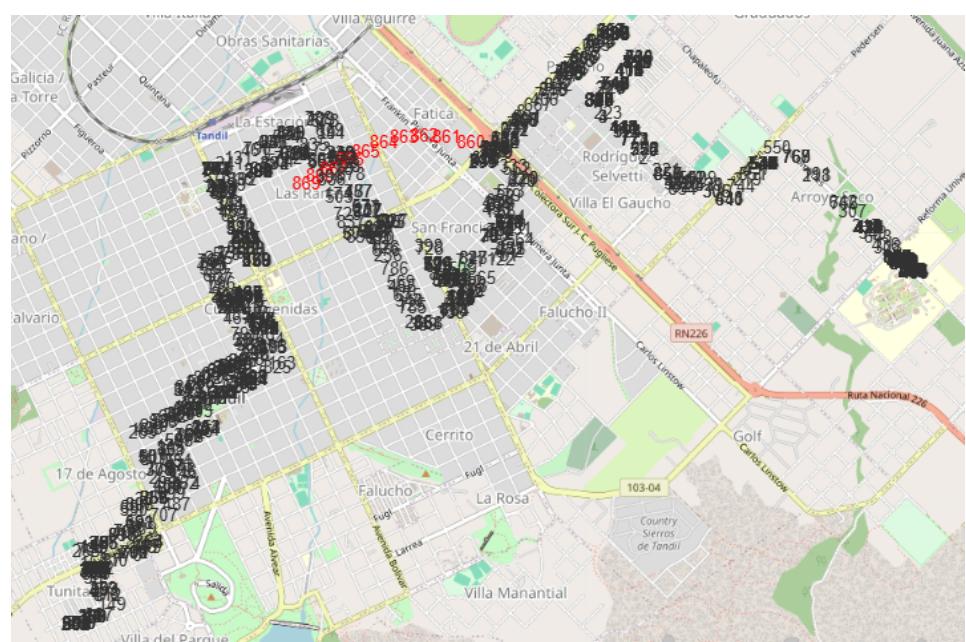


Figura 13 Predicciones del Modelo ARIMA Línea 501.

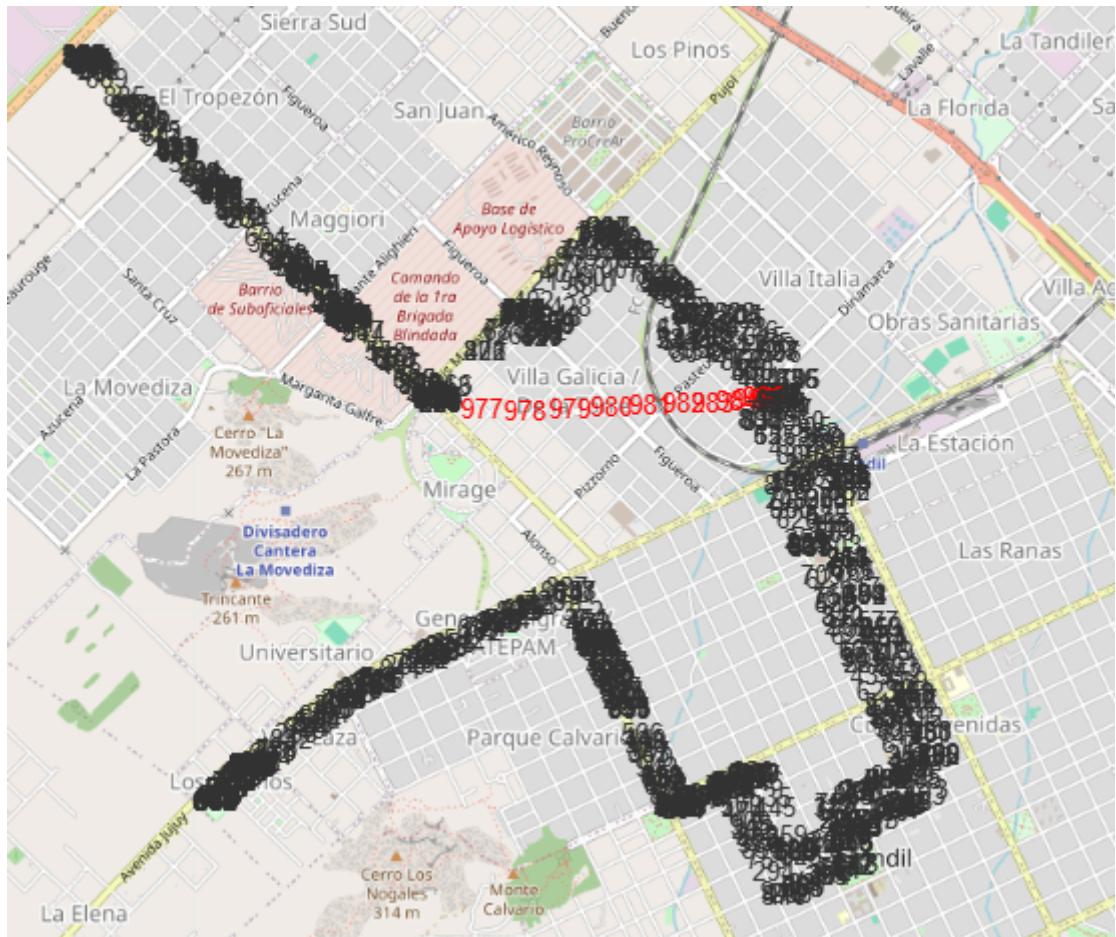


Figura 14 Predicciones del Modelo ARIMA Línea 502.

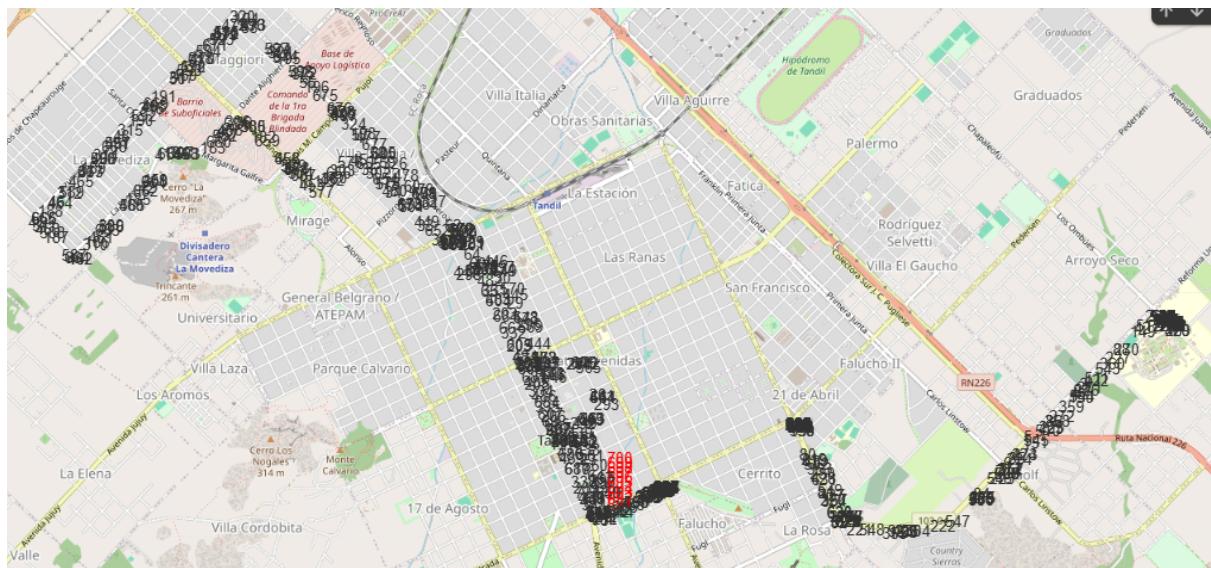


Figura 15 Predicciones del Modelo ARIMA Línea 503.

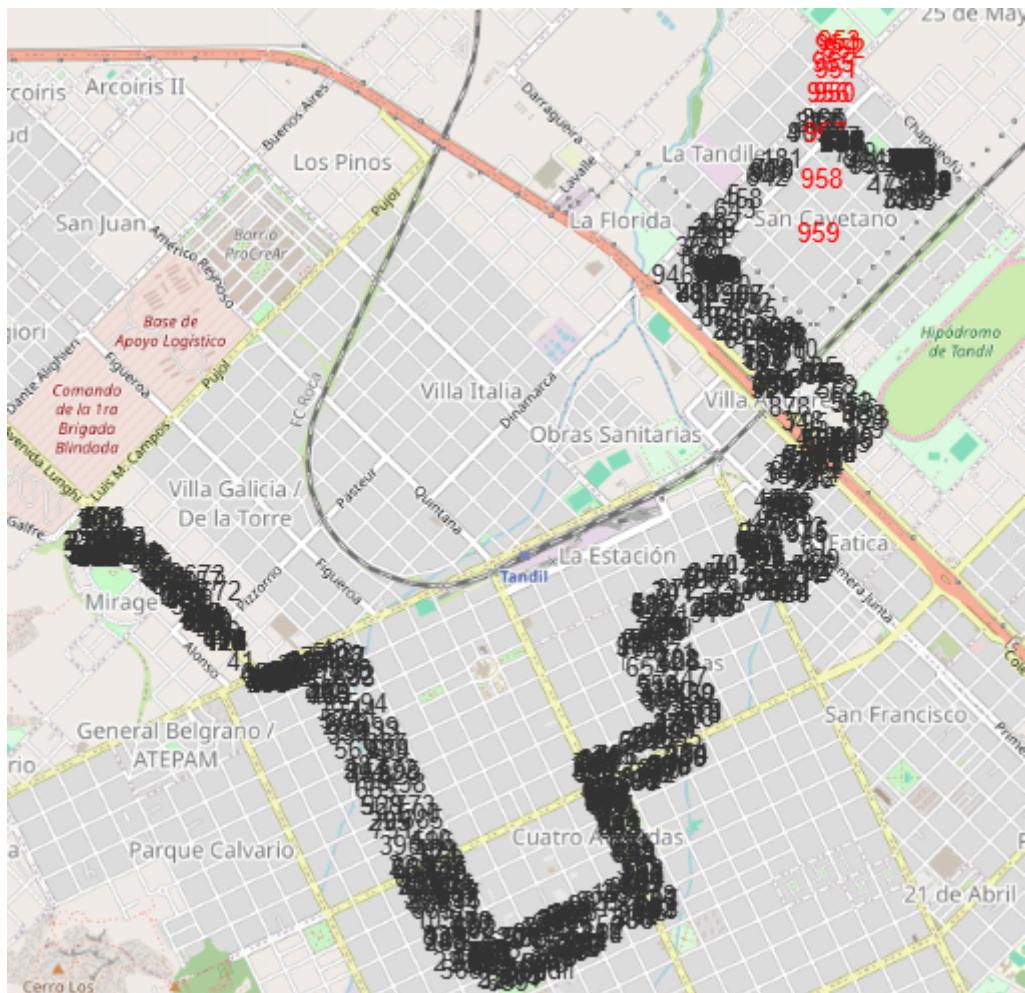


Figura 16 Predicciones del Modelo ARIMA Línea 504.

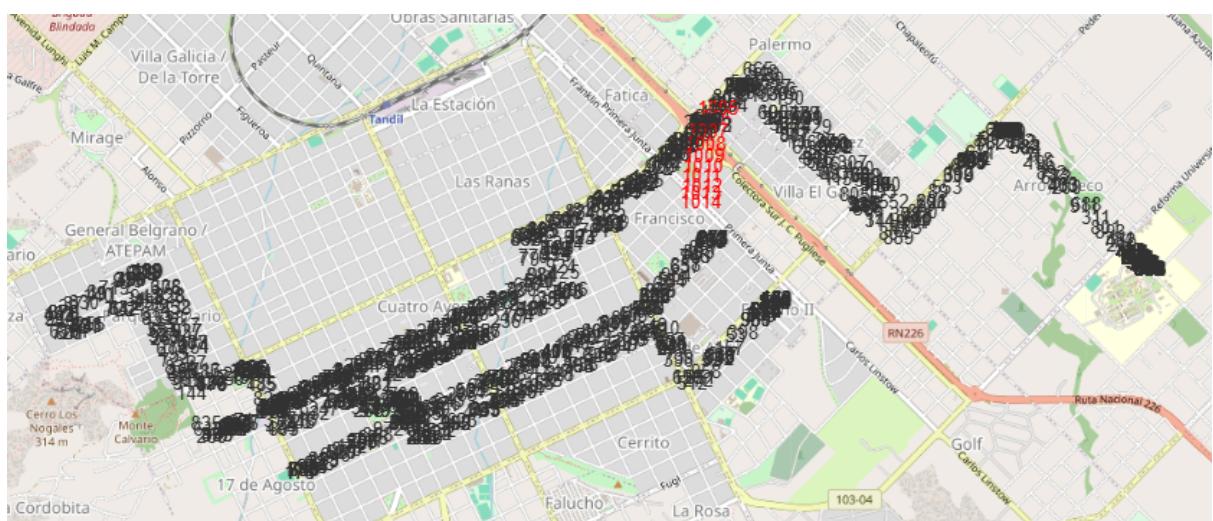


Figura 17 Predicciones del Modelo ARIMA Línea 505.

Este comportamiento podría deberse a la naturaleza del modelo ARIMA y a las características intrínsecas de los datos, más que a la cantidad o calidad de estos.

Con 900 elementos por serie, el volumen de datos es suficiente para entrenar el modelo, y la regularidad del intervalo de un minuto favorece la estabilidad de la predicción. No obstante, si las variaciones entre coordenadas son pequeñas, el modelo podría encontrar dificultades para identificar patrones claros. ARIMA es más efectivo en datos con tendencias bien definidas, y si las variaciones en las coordenadas no se ajustan bien a un modelo basado en diferencias, es posible que esto amplificara pequeñas desviaciones en lugar de capturar un patrón significativo.

Además, ARIMA es un modelo lineal, por lo que si los datos contienen componentes no lineales, el modelo no puede representarlos adecuadamente. En recorridos con curvas pronunciadas o trayectorias zigzagueantes, las coordenadas espaciales no siguen un patrón lineal predecible, y el modelo intenta ajustarse incorrectamente. Otro posible problema puede ser el modelado de la latitud y longitud por separado, lo que ignora la relación geográfica entre ambas y podía generar predicciones inconsistentes, donde una coordenada predicha fuera razonable, pero la otra no se correspondiera con su ubicación real.

Ante estos inconvenientes, se exploraron alternativas sin obtener mejoras significativas en los resultados. En primer lugar, se intentó predecir la distancia recorrida entre puntos consecutivos en lugar de las coordenadas geográficas. Sin embargo, este método presentaba el inconveniente de asumir trayectorias en línea recta entre puntos, cuando en realidad el recorrido podía incluir giros y desvíos, afectando la precisión de la estimación. Se observó que los primeros pasos de la predicción tenían un error relativamente bajo, pero este aumentaba progresivamente a medida que se predecían más pasos, volviendo el modelo ineficaz para estimaciones a largo plazo. En las Figuras 18 a 23 se observan las predicciones de distancia recorrida comparadas con el valor real y los errores acumulados luego de 20 minutos.

```

1 minutos despues= Prediccion: 298.58838763487256, Valor Real: 41.17645282115639, Error: 257.4119348137162, Error Acumulado: -257.4119348137162
2 minutos despues= Prediccion: 277.44644539562034, Valor Real: 167.71655256217022, Error: 109.72989283345012, Error Acumulado: -367.1418276471663
3 minutos despues= Prediccion: 256.67253304206815, Valor Real: 187.90332915263113, Error: 68.76920388943702, Error Acumulado: -435.91103153660333
4 minutos despues= Prediccion: 237.83778171698643, Valor Real: 426.9516877970922, Error: -189.1139059901058, Error Acumulado: -246.79712554649745
5 minutos despues= Prediccion: 221.91325126168647, Valor Real: 387.4880824657336, Error: -165.57483120404714, Error Acumulado: -81.22229434245014
6 minutos despues= Prediccion: 209.3610994208148, Valor Real: 284.63140033983154, Error: -75.27030091901673, Error Acumulado: -5.9519934234333505
7 minutos despues= Prediccion: 200.24103796149333, Valor Real: 425.68343481948085, Error: -225.44239685798752, Error Acumulado: 219.4904034345543
8 minutos despues= Prediccion: 194.31917455500263, Valor Real: 193.88324845773425, Error: 0.435926972683785, Error Acumulado: 219.0544773372858
9 minutos despues= Prediccion: 191.16979037154792, Valor Real: 495.1693737355947, Error: -303.99958336404677, Error Acumulado: 523.054060701322
10 minutos despues= Prediccion: 190.26384178740958, Valor Real: 174.05906883571143, Error: 16.204772951698146, Error Acumulado: 506.84928774996342
11 minutos despues= Prediccion: 191.0407993287859, Valor Real: 430.825914808413, Error: -239.7851154796271, Error Acumulado: 746.6344032292614
12 minutos despues= Prediccion: 192.96273705275536, Valor Real: 771.6673471985695, Error: -578.7046101458052, Error Acumulado: 1325.3390133750663
13 minutos despues= Prediccion: 195.55132369892402, Valor Real: 365.99566730916244, Error: -170.44428361023841, Error Acumulado: 1495.7832969853043
14 minutos despues= Prediccion: 198.4095616850733, Valor Real: 258.928302525917, Error: -60.51874084084369, Error Acumulado: 1556.3020378261485
15 minutos despues= Prediccion: 201.23082849297427, Valor Real: 251.35252567464966, Error: -50.12169718167539, Error Acumulado: 1606.4237350078242
16 minutos despues= Prediccion: 203.7980773989102, Valor Real: 327.7347869379992, Error: -123.93670953908901, Error Acumulado: 1730.3604445469132
17 minutos despues= Prediccion: 205.97604114172753, Valor Real: 94.22117550419044, Error: 111.7548656375371, Error Acumulado: 1618.6055789093766
18 minutos despues= Prediccion: 207.69904324491498, Valor Real: 0.0, Error: 207.69904324491498, Error Acumulado: 1410.9065356644614
19 minutos despues= Prediccion: 208.95664026829073, Valor Real: 92.65777594723619, Error: 116.29886432105454, Error Acumulado: 1294.6076713434072
20 minutos despues= Prediccion: 209.77886527851115, Valor Real: 49.79384224411586, Error: 159.98502303439528, Error Acumulado: 1134.6226483090122

```

Figura 18 Predicciones del Modelo ARIMA para Distancias Encadenadas Línea 500.

```

1 minutos despues= Prediccion: 264.3123481986345, Valor Real: 188.9755359436143, Error: 75.33681225502019, Error Acumulado: -75.33681225502019
2 minutos despues= Prediccion: 262.1831720805118, Valor Real: 335.64572846975904, Error: -73.46255638924725, Error Acumulado: -1.874255865772966
3 minutos despues= Prediccion: 257.079940351793, Valor Real: 315.4056660706514, Error: -58.32572571887209, Error Acumulado: 56.45146985309967
4 minutos despues= Prediccion: 252.5262681942517, Valor Real: 0.0, Error: 252.5262681942517, Error Acumulado: -196.07479834632613
5 minutos despues= Prediccion: 248.46297435478607, Valor Real: 225.3964802177763, Error: 23.066494137009784, Error Acumulado: -219.14129248333575
6 minutos despues= Prediccion: 244.8372506936423, Valor Real: 280.56391283634906, Error: -35.72666214270677, Error Acumulado: -183.41463034062895
7 minutos despues= Prediccion: 241.6017952159452, Valor Real: 319.35331059772824, Error: -77.75133467613372, Error Acumulado: -105.6632956449508
8 minutos despues= Prediccion: 238.71510316766083, Valor Real: 514.1893793832827, Error: -275.4742762156219, Error Acumulado: 169.8109805511267
9 minutos despues= Prediccion: 236.1391135270522, Valor Real: 204.82054074651526, Error: 31.318572780509953, Error Acumulado: 138.49240777061586
10 minutos despues= Prediccion: 233.84052845097017, Valor Real: 676.7869213756948, Error: -442.9463986847246, Error Acumulado: 581.4388064553418
11 minutos despues= Prediccion: 231.78947464682548, Valor Real: 234.31538349243095, Error: -2.5259088456054712, Error Acumulado: 583.9647153009473
12 minutos despues= Prediccion: 229.9529583320662, Valor Real: 350.360186438114, Error: -120.4012281060476, Error Acumulado: 704.365838111552
13 minutos despues= Prediccion: 228.32620630476487, Valor Real: 258.7713854835103, Error: -30.45179178745434, Error Acumulado: 734.8110172992971
14 minutos despues= Prediccion: 226.86898180404376, Valor Real: 57.35633276301649, Error: 169.512649490120726, Error Acumulado: 565.2983682492695
15 minutos despues= Prediccion: 225.56868368289415, Valor Real: 423.1153881698335, Error: -197.5467448693938, Error Acumulado: 762.8450727362992
16 minutos despues= Prediccion: 224.40841276854394, Valor Real: 534.0668337879995, Error: -309.6584201201945555, Error Acumulado: 1072.5034937556647
17 minutos despues= Prediccion: 223.37308973547147, Valor Real: 251.9950654135749, Error: -28.621966678103433, Error Acumulado: 1101.1254604337682
18 minutos despues= Prediccion: 222.44925912871125, Valor Real: 622.3036236460773, Error: -399.8543645119661, Error Acumulado: 1500.979249457335
19 minutos despues= Prediccion: 221.624914491606396, Valor Real: 435.700880247826, Error: -214.0759733317865, Error Acumulado: 1715.055798478912
20 minutos despues= Prediccion: 220.8893423252837, Valor Real: 283.4980240499881, Error: -62.60868172470438, Error Acumulado: 1777.6644802036162

```

Figura 19 Predicciones del Modelo ARIMA para Distancias Encadenadas Línea 501.

```

1 minutos despues= Prediccion: 236.71515688303, Valor Real: 15.54982843028497, Error: 221.16532845274503, Error Acumulado: -221.16532845274503
2 minutos despues= Prediccion: 238.2446770305186, Valor Real: 0.0, Error: 238.2446770305186, Error Acumulado: -459.41000548326366
3 minutos despues= Prediccion: 239.34482354224147, Valor Real: 47.32148199633653, Error: 192.02334154590494, Error Acumulado: -651.4333470291687
4 minutos despues= Prediccion: 240.1361320773028, Valor Real: 275.76543853567074, Error: -35.62936212637685, Error Acumulado: -615.8040405708007
5 minutos despues= Prediccion: 240.70530099343227, Valor Real: 187.2691511207946, Error: 53.436149872637685, Error Acumulado: -669.2401904434385
6 minutos despues= Prediccion: 241.1146903333384, Valor Real: 293.7558671748113, Error: -52.641176871477484, Error Acumulado: -616.599013571961
7 minutos despues= Prediccion: 241.40915404215048, Valor Real: 370.5443979819945, Error: -129.13519575604897, Error Acumulado: -487.4638178159121
8 minutos despues= Prediccion: 241.62095462262658, Valor Real: 454.59416481436216, Error: -212.97321019173557, Error Acumulado: -274.4906076241776
9 minutos despues= Prediccion: 241.77329761083388, Valor Real: 193.912328074847898, Error: 47.861043336343896, Error Acumulado: -322.351650987520777
10 minutos despues= Prediccion: 241.8828742040791, Valor Real: 848.3951101154994, Error: -606.5122359114114, Error Acumulado: 284.16065849238994
11 minutos despues= Prediccion: 241.961689973209976, Valor Real: 577.1909359953918, Error: -335.229246022182, Error Acumulado: 619.3898309460728
12 minutos despues= Prediccion: 242.01838023323156, Valor Real: 421.18996345198615, Error: -179.1711832196458, Error Acumulado: 798.5610141657371
13 minutos despues= Prediccion: 242.0591516152085, Valor Real: 311.25636462114295, Error: -69.197208469509759, Error Acumulado: 867.758222634795
14 minutos despues= Prediccion: 242.08848527596982, Valor Real: 413.57049536368964, Error: -171.4820108871983, Error Acumulado: 1039.2402327225145
15 minutos despues= Prediccion: 242.1095809895637, Valor Real: 20.601160170105498, Error: 221.508428288509, Error Acumulado: 817.7318118936637
16 minutos despues= Prediccion: 242.124754637626, Valor Real: 432.8561015783094, Error: -198.73085552020493, Error Acumulado: 1088.4626674138681
17 minutos despues= Prediccion: 242.135085950206, Valor Real: 418.300830650963865, Error: -176.165186159451217, Error Acumulado: 1184.62782936838
18 minutos despues= Prediccion: 242.14351885950206, Valor Real: 276.33260191015927, Error: -34.189083059063866, Error Acumulado: 1218.816912419019
19 minutos despues= Prediccion: 242.149165321242255, Valor Real: 452.1015213539984, Error: -209.95235604157585, Error Acumulado: 1428.7692684605954
20 minutos despues= Prediccion: 242.15322666818426, Valor Real: 121.23775779236578, Error: 120.91546887581848, Error Acumulado: 1387.8537995847773

```

Figura 20 Predicciones del Modelo ARIMA para Distancias Encadenadas Línea 502.

```

1 minutos despues= Prediccion: 106.23189145871918, Valor Real: 0.0, Error: 106.23189145871918, Error Acumulado: -106.23189145871918
2 minutos despues= Prediccion: 119.79980831081451, Valor Real: 0.0, Error: 119.79980831081451, Error Acumulado: -226.0316997695337
3 minutos despues= Prediccion: 139.24481075076872, Valor Real: 0.0, Error: 139.24481075076872, Error Acumulado: -365.2765105203024
4 minutos despues= Prediccion: 154.21669060809973, Valor Real: 0.0, Error: 154.21669060809973, Error Acumulado: -519.4932011284021
5 minutos despues= Prediccion: 165.7444367139207, Valor Real: 573.4350283356363, Error: -407.6905846632424, Error Acumulado: -111.80261646515794
6 minutos despues= Prediccion: 174.620355475911, Valor Real: 260.41705156903146, Error: -85.79669574140436, Error Acumulado: -26.005920743717525
7 minutos despues= Prediccion: 181.45445582375004, Valor Real: 308.412250542742471, Error: -126.95779482872405, Error Acumulado: 100.9518408500647
8 minutos despues= Prediccion: 186.7164414723505, Valor Real: 316.4623387004959, Error: -129.74589722814537, Error Acumulado: 230.6977131315186
9 minutos despues= Prediccion: 190.76796156108332, Valor Real: 417.1515788077116, Error: -226.3836173016827, Error Acumulado: 457.08138861476004
10 minutos despues= Prediccion: 193.88741707532194, Valor Real: 305.4139771674342, Error: -111.526506959211227, Error Acumulado: 568.6078947068725
11 minutos despues= Prediccion: 196.2893695902306, Valor Real: 514.6774546659425, Error: -318.3880857056494, Error Acumulado: 886.995979782522
12 minutos despues= Prediccion: 198.13873608467617, Valor Real: 494.7429488104519, Error: -296.6041227257757, Error Acumulado: 1183.6001925982976
13 minutos despues= Prediccion: 199.56267486028594, Valor Real: 390.40944190137634, Error: -190.8467670410904, Error Acumulado: 1374.4469595493879
14 minutos despues= Prediccion: 200.65905181312043, Valor Real: 303.401230587428203, Error: -102.39880167116493, Error Acumulado: 1476.8450412205493
15 minutos despues= Prediccion: 201.503217424274107, Valor Real: 361.10792221693885, Error: -159.6047842479446778, Error Acumulado: 1636.4497460150174
16 minutos despues= Prediccion: 202.15139191099073, Valor Real: 190.54083587046884, Error: 11.61235604521888, Error Acumulado: 1624.8373899744956
17 minutos despues= Prediccion: 202.653364646133068, Valor Real: 175.57031651312022, Error: 27.083329948210462, Error Acumulado: 1597.754060026285
18 minutos despues= Prediccion: 203.03897659563546, Valor Real: 427.51964184525497, Error: -224.4806652496195, Error Acumulado: 1822.2347252759046
19 minutos despues= Prediccion: 203.33566550035715, Valor Real: 185.0601890577585, Error: 18.27547644259866, Error Acumulado: 1803.9592488333055
20 minutos despues= Prediccion: 203.56410417760662, Valor Real: 159.84312743950983, Error: 43.720976738096795, Error Acumulado: 1760.2382720952087

```

Figura 21 Predicciones del Modelo ARIMA para Distancias Encadenadas Línea 503.

```

1 minutos despues= Prediccion: 303.3795286171584, Valor Real: 65.74481276649598, Error: 237.63471585066245, Error Acumulado: -237.63471585066245
2 minutos despues= Prediccion: 283.6980027313119, Valor Real: 764.857057579605, Error: -481.1590550282926, Error Acumulado: 243.5243917763017
3 minutos despues= Prediccion: 267.20965259976276, Valor Real: 541.9328504721502, Error: -274.72319787238746, Error Acumulado: 518.2475370500175
4 minutos despues= Prediccion: 254.066081223163, Valor Real: 295.39767507990945, Error: -41.33699496776782, Error Acumulado: 559.5845319576954
5 minutos despues= Prediccion: 243.57476287180066, Valor Real: 360.00400768702394, Error: -116.42949851252329, Error Acumulado: 676.0137767729186
6 minutos despues= Prediccion: 235.2125534111392, Valor Real: 148.39158475763607, Error: 86.82096865410313, Error Acumulado: 589.19280811188155
7 minutos despues= Prediccion: 228.54393825230116, Valor Real: 363.80534867432596, Error: -135.2614103950248, Error Acumulado: 724.4542185138403
8 minutos despues= Prediccion: 223.22591454745523, Valor Real: 364.6876530785363, Error: -141.4617385180107, Error Acumulado: 865.915957049212
9 minutos despues= Prediccion: 218.9849476522789, Valor Real: 264.2049272100507, Error: -45.2199759577181, Error Acumulado: 911.1359366026932
10 minutos despues= Prediccion: 215.60290176377625, Valor Real: 199.13427564386454, Error: 16.468626119920714, Error Acumulado: 894.6673104827723
11 minutos despues= Prediccion: 212.9058200296973, Valor Real: 395.2849989268284, Error: -182.3782789057557, Error Acumulado: 1077.0455893903109
12 minutos despues= Prediccion: 210.75497695672834, Valor Real: 370.310043496580893, Error: -159.5558665398526, Error Acumulado: 1236.600655930184
13 minutos despues= Prediccion: 209.03974311588522, Valor Real: 543.9449006574552, Error: -334.90515694157, Error Acumulado: 1571.5958128717537
14 minutos despues= Prediccion: 207.67189476841108, Valor Real: 215.7296840804463, Error: -8.057789312035254, Error Acumulado: 1579.563602183789
15 minutos despues= Prediccion: 206.58107609568742, Valor Real: 362.09883208554443, Error: -155.50975598986201, Error Acumulado: 1735.0733581736508
16 minutos despues= Prediccion: 205.7111804953365, Valor Real: 0.0, Error: 205.7111804953365, Error Acumulado: 1529.361277641174
17 minutos despues= Prediccion: 205.01746428264767, Valor Real: 492.19873134515836, Error: -287.1812670625107, Error Acumulado: 1816.543448266278
18 minutos despues= Prediccion: 204.4642461527449, Valor Real: 531.8389026347711, Error: -327.3746558820262, Error Acumulado: 2143.9181007086545
19 minutos despues= Prediccion: 204.02307102737058, Valor Real: 226.94708597510526, Error: -22.90218505164468, Error Acumulado: 2166.842191260299
20 minutos despues= Prediccion: 203.67124689014778, Valor Real: 864.534661768455, Error: -660.8634148783071, Error Acumulado: 2827.705534138606

```

Figura 22 Predicciones del Modelo ARIMA para Distancias Encadenadas Línea 504.

```

1 minutos despues= Prediccion: 239.98349844794484, Valor Real: 427.47129964854844, Error: -187.4878012006036, Error Acumulado: 187.4878012006036
2 minutos despues= Prediccion: 234.1336573591469, Valor Real: 0.0, Error: 234.1336573591469, Error Acumulado: -46.645856158543324
3 minutos despues= Prediccion: 245.65049423169475, Valor Real: 694.856344228464, Error: -449.2058499887693, Error Acumulado: 402.55999383022595
4 minutos despues= Prediccion: 247.9658710140539, Valor Real: 349.1935884033355, Error: -101.22771738928162, Error Acumulado: 503.7877112195076
5 minutos despues= Prediccion: 251.37325907728743, Valor Real: 337.53716913505855, Error: -86.16391005777112, Error Acumulado: 589.9516212772787
6 minutos despues= Prediccion: 253.07611990630434, Valor Real: 187.83053454485276, Error: 65.24558446145159, Error Acumulado: 524.7060368158272
7 minutos despues= Prediccion: 255.8028336780268, Valor Real: 275.58139343949006, Error: -19.77855976146938, Error Acumulado: 544.4845965772968
8 minutos despues= Prediccion: 257.8985676346074, Valor Real: 351.93182625529437, Error: -94.03326049183363, Error Acumulado: 638.5178570691303
9 minutos despues= Prediccion: 259.73231563477486, Valor Real: 411.94685530620796, Error: -152.2145396714331, Error Acumulado: 790.7323967405632
10 minutos despues= Prediccion: 261.21886957313774, Valor Real: 497.18584345350445, Error: -235.9669738803667, Error Acumulado: 1026.6993706209296
11 minutos despues= Prediccion: 262.5940023193715, Valor Real: 385.123189454821, Error: -122.52918713544949, Error Acumulado: 1149.228557756379
12 minutos despues= Prediccion: 263.80084760162543, Valor Real: 266.45924559869854, Error: -2.65839970731093, Error Acumulado: 1151.8869557534522
13 minutos despues= Prediccion: 264.8555843994901, Valor Real: 312.00730489657684, Error: -47.15172049708673, Error Acumulado: 1199.038676250539
14 minutos despues= Prediccion: 265.7664906623392, Valor Real: 463.111032791654, Error: -197.34454261682617, Error Acumulado: 1396.383218867365
15 minutos despues= Prediccion: 266.5659325503299, Valor Real: 0.0, Error: 266.5659325503299, Error Acumulado: 1129.8172863170353
16 minutos despues= Prediccion: 267.26693572169995, Valor Real: 1162.9091294351872, Error: -895.6421937134872, Error Acumulado: 2025.459480030523
17 minutos despues= Prediccion: 267.8805299508594, Valor Real: 156.26989097881716, Error: 111.6186380004224, Error Acumulado: 1913.8488410504806
18 minutos despues= Prediccion: 268.416202110377, Valor Real: 601.6920964836072, Error: -333.2758943732302, Error Acumulado: 2247.124735423711
19 minutos despues= Prediccion: 268.88456804670056, Valor Real: 386.87265315372144, Error: -117.9880510702888, Error Acumulado: 2365.1128205307323
20 minutos despues= Prediccion: 269.29434772541583, Valor Real: 200.84561782775143, Error: 68.4487298976644, Error Acumulado: 2296.6640906330676

```

Figura 23 Predicciones del Modelo ARIMA para Distancias Encadenadas Línea 505.

Se observa que, aunque los errores iniciales eran pequeños, después de 20 minutos la predicción presentaba una desviación de más de 1 kilómetro en todos los modelos (algunos más de 2), lo que evidencia una pérdida significativa de precisión.

En conclusión, aunque ARIMA puede ofrecer buenos resultados en condiciones ideales, las características de los datos espaciales analizados no se ajustaban bien a su estructura matemática. Dado que el modelo presentó problemas incluso en conjuntos de datos reducidos y controlados, era poco probable que lograra mejoras en conjuntos más grandes. Esto indicaba la necesidad de explorar modelos alternativos más adecuados para el tipo de datos en cuestión.

4.3 Modelo LSTM

El modelo de redes neuronales de memoria a largo plazo (LSTM) es ampliamente utilizado para el análisis de secuencias temporales, lo que lo hace adecuado para el seguimiento de colectivos. Este tipo de modelo es capaz de aprender las relaciones temporales entre eventos pasados, como los tiempos de paso por determinadas coordenadas geográficas, y realizar predicciones sobre posiciones futuras.

En la implementación inicial, se emplearon capas LSTM para modelar las dependencias temporales en los datos de ubicación. Se proporcionaron al modelo datos históricos que incluyen posiciones geográficas y tiempos de paso, sin incorporar información adicional como condiciones de tráfico u otras variables contextuales. El objetivo fue evaluar la capacidad del modelo para predecir las coordenadas futuras basándose únicamente en la secuencia de posiciones previas.

Para la evaluación inicial, se utilizaron tres conjuntos de datos diferenciados por el intervalo de muestreo: cada cinco minutos, cada dos minutos y cada un minuto. Cada conjunto de datos posee una cantidad diferente de registros los cuales fueron procesados utilizando una ventana deslizante de 10 pasos previos para estimar la posición del colectivo en el siguiente paso. Es importante destacar que cada paso en la predicción equivale al intervalo de muestreo utilizado en la construcción del conjunto de datos.

Los resultados obtenidos de predecir la siguiente posición de un colectivo particular dada las últimas diez posiciones fueron los siguientes en términos de métricas como MSE y MAE. Para cada conjunto de datos, el 70% se utilizaron para entrenar el modelo y el 30% restante como testeo, pero al ser datos secuenciales donde el orden es fundamental, no es posible separarlos de forma aleatoria sino que la división se hizo con un split en la cantidad deseada siendo los primeros para entrenamiento y los últimos para testeo.

Línea	Intervalos	Error Absoluto Medio (MAE)	Error Cuadrático Medio (MSE)
500	Cada 1 minuto	0.000817	2.239951 e-6
	Cada 2 minutos	0.001247	4.666387 e-6
	Cada 5 minutos	0.002482	1.490602 e-5
501	Cada 1 minuto	0.000933	2.739497 e-6
	Cada 2 minutos	0.001399	5.459574 e-6
	Cada 5 minutos	0.002753	1.714023 e-5
502	Cada 1 minuto	0.001021	2.989869 e-6
	Cada 2 minutos	0.001541	6.650852 e-6
	Cada 5 minutos	0.003369	2.634859 e-5
503	Cada 1 minuto	0.001118	5.934820 e-6
	Cada 2 minutos	0.001785	1.187022 e-5
	Cada 5 minutos	0.003622	3.737278 e-5
504	Cada 1 minuto	0.000986	3.115848 e-6
	Cada 2 minutos	0.001410	6.026867 e-6
	Cada 5 minutos	0.002982	1.919876 e-5
505	Cada 1 minuto	0.001848	1.176009 e-5
	Cada 2 minutos	0.0028471	2.755869 e-5
	Cada 5 minutos	0.005137	7.845086 e-5

Tabla 4 Métricas del Modelo LSTM Simple para cada Conjunto de Datos.

El análisis comparativo de las métricas obtenidas para cada línea de colectivo muestra que, en general, el error en las predicciones aumenta a medida que el intervalo de medición se amplía. Esto indica que cuanto mayor es el tiempo entre registros, mayor es la acumulación de desviaciones en los tiempos estimados, lo que es esperable en sistemas de transporte sujetos a variaciones en la circulación.

Las líneas 500, 501, 502 y 504 presentan un comportamiento estable, con errores relativamente bajos en todas las mediciones. Esto sugiere que sus tiempos entre puntos son más regulares y que el modelo logra capturar con mayor precisión sus patrones de desplazamiento. En contraste, la línea 503 muestra un aumento significativo en los errores, especialmente en el MSE, lo que indica una mayor variabilidad en los tiempos de recorrido.

Por otro lado, la línea 505 es la que presenta los errores más elevados, reflejando un comportamiento menos predecible. La alta variabilidad en sus tiempos podría deberse a factores como congestión vehicular, demoras en paradas o irregularidades en la frecuencia del servicio.

A pesar de que los valores obtenidos permitieron una comparación inicial entre modelos, su interpretación numérica puede resultar poco intuitiva. Por ello, se realizó una visualización en un mapa (Figura 24), donde se representaron en azul los valores reales y en rojo las predicciones obtenidas, concentrándose únicamente en el conjunto con intervalos de un minuto para la línea 500 a modo de exemplificación.

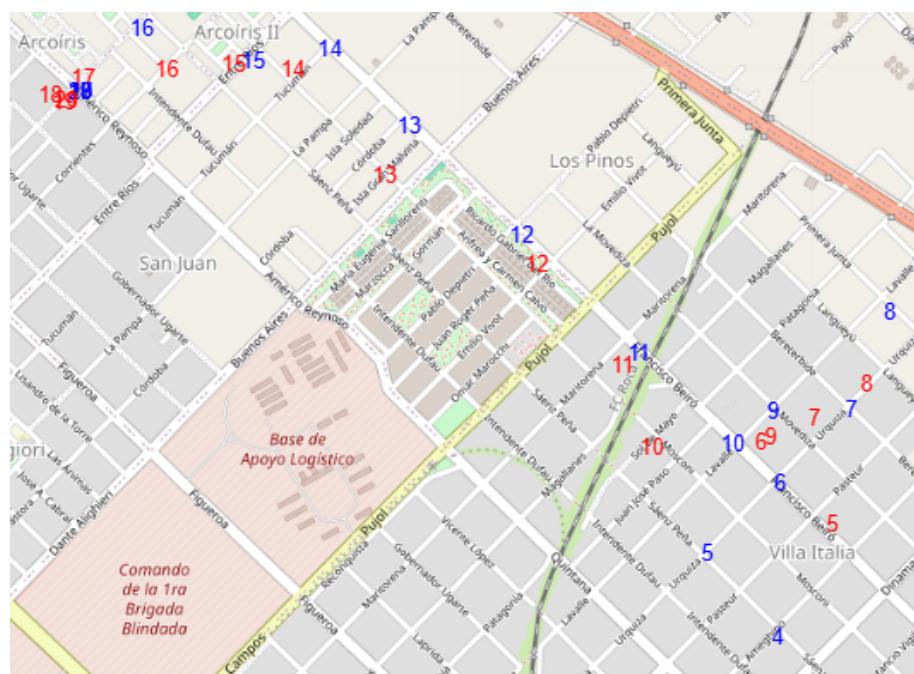


Figura 24 Predicciones del Modelo LSTM Simple con Intervalos de 1 Minuto para la Línea 500.

Si bien las predicciones iniciales mostraron resultados prometedores, su utilidad práctica es limitada cuando se analizan intervalos de solo un minuto. Para evaluar el rendimiento en predicciones más prolongadas, se implementó un enfoque de predicción recursiva: tras generar una primera predicción, ésta se incorpora a la ventana de entrada y se utiliza para predecir el siguiente valor, repitiendo este proceso durante 20 pasos. En la visualización resultante, se representaron en negro los valores de la ventana base, en azul los valores reales de los siguientes 20 pasos y en rojo las predicciones obtenidas para cada uno de los tres modelos, nuevamente para la linea 500.

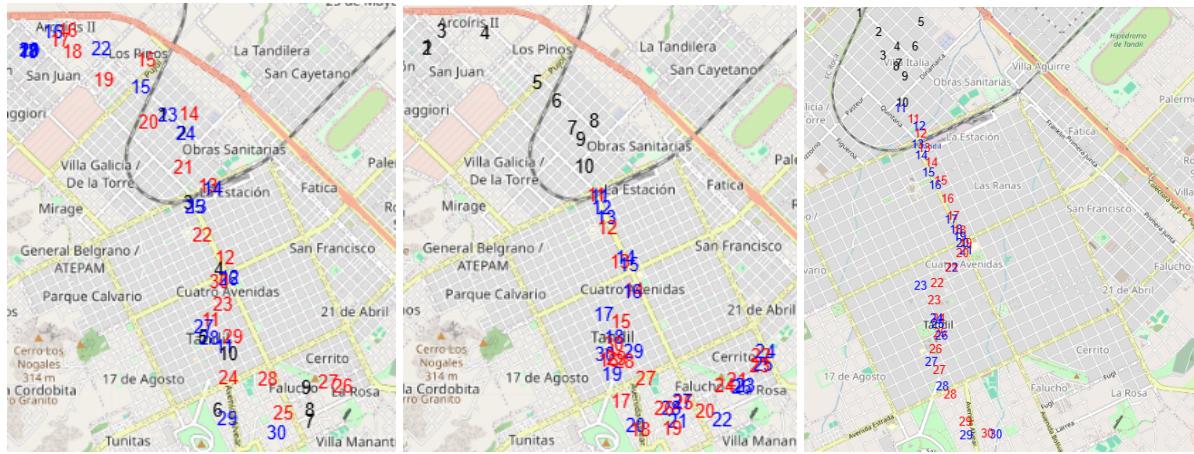


Figura 25 Predicciones del Modelo LSTM Simple con Intervalos de 5, 2 y 1 Minuto de la Línea 500 respectivamente.

Los resultados de la Figura 25 mostraron que la precisión del modelo disminuye significativamente en los intervalos de cinco y dos minutos, mientras que en el intervalo de un minuto la precisión es mayor. Sin embargo, se identificaron problemas específicos en los casos en que el colectivo se encontraba detenido al final del recorrido, donde las predicciones continuaban la trayectoria en lugar de reflejar la detención.

Dado que el modelo LSTM utilizado no considera explícitamente la relación temporal y espacial de los datos, sino que trata la secuencia de coordenadas como una simple lista, se propuso un refinamiento del conjunto de datos. En particular, eliminar registros consecutivos repetidos en los puntos de inicio y finalización de los recorridos, considerando que el tiempo en que el colectivo permanece detenido podría no ser relevante para la predicción.

Tras aplicar este filtrado, se procedió a entrenar nuevamente el modelo con los datos filtrados y a evaluar las predicciones utilizando las mismas métricas MAE y MSE. Aquí se ve que este último es menor en todos los casos, lo que sugiere que redujo la magnitud de los errores más grandes. Sin embargo, el MAE empeoró levemente, lo que podría deberse a pequeñas variaciones en los datos tras la eliminación de registros consecutivos.

Intervalos	Error Absoluto Medio (MAE)	Error Cuadrático Medio (MSE)
Cada 1 minuto	0.000797	1.644806 e-6
Cada 2 minutos	0.001363	3.980124 e-6
Cada 5 minutos	0.002768	1.458155 e-5

Tabla 5 Métricas del Modelo LSTM Simple para cada Conjunto de Datos Filtrados.

Los valores de la Tabla 5 sugieren que el filtrado no afectó significativamente la precisión en predicciones de un solo paso, pero sí ayudó a mejorar la coherencia y a reducir valores atípicos, lo que puede ser beneficioso en predicciones más largas. Para poder observar esto vamos a mostrar nuevamente las predicciones consecutivas de la Línea 500 en un mapa.

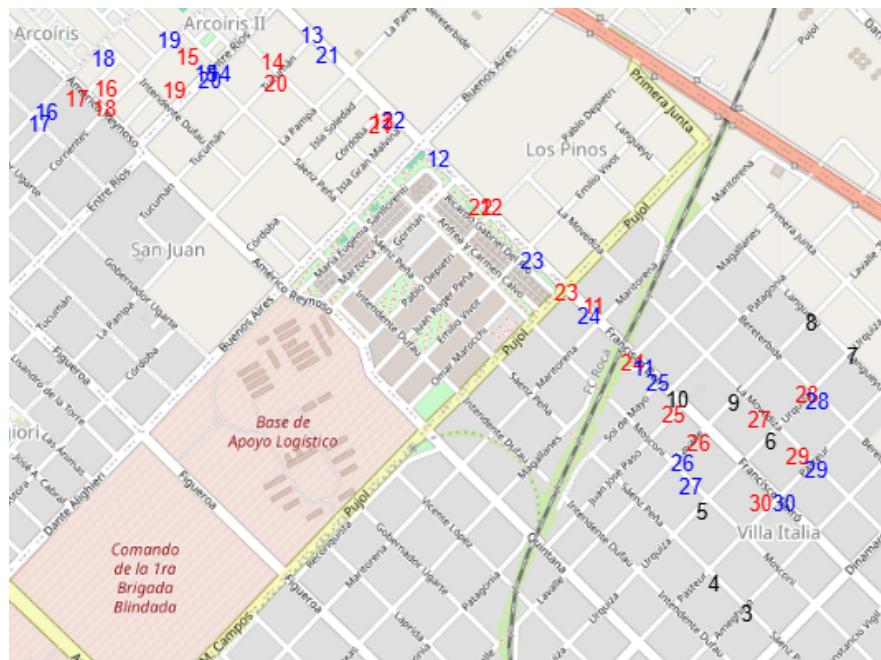


Figura 26 Predicciones del Modelo LSTM Simple con Intervalos de 1 Minuto de la Línea 500 Filtrados.

En la Figura 26 se puede observar que anteriormente los registros posteriores al punto 17 se iban a encontrar repetidos allí, sin embargo posterior al filtrado no tiene en cuenta eso y tanto la predicción como el valor real van cerca.

Para continuar optimizando el modelo, se consideró la incorporación de información adicional, incluyendo variables como la hora del día, la velocidad del colectivo, el tiempo exacto entre registros, la indicación de inicio o fin del recorrido y el día de la semana. El aumento en la cantidad de variables influyó en el tiempo de entrenamiento del modelo. Un aspecto relevante es cómo afecta el valor de la

velocidad cuando se hace una seguidilla de predicciones a partir de las anteriores, ya que para predecir se deben generar algunas de las variables agregadas, muchas son triviales, por ejemplo el tiempo entre paradas, o el dia que se mantiene, pero particularmente la velocidad depende la que se ingrese afecta la posición final de la estimación, lo cual tiene sentido ya que cuanto más rápido vaya más lejos llegará. Por eso no es posible poner un valor estimativo ahí como 20 km/h sino que se tomó la decisión de modificar el modelo para que también prediga la velocidad, buscando que la misma tenga un valor en consecuencia con la zona, dia y horario específico, obteniendo mejores resultados a la hora de hacer predicciones en cadena.

Los resultados de esta versión del modelo para la Linea 500 mostraron una leve mejora en las métricas MAE y MSE, aunque las predicciones visualizadas en el mapa no presentaron mejoras significativas a simple vista.

Intervalos	Error Absoluto Medio (MAE)	Error Cuadrático Medio (MSE)
Cada 1 minuto	0.000783	1.641043 e-6
Cada 2 minutos	0.001180	3.355344 e-6
Cada 5 minutos	0.002640	1.323942 e-5

Tabla 6 Métricas del Modelo LSTM Complejo para cada Conjunto de Datos.

Esto que se visualiza para esta línea de colectivos también es aplicable para el resto, ya que como se vio al principio de esta sección todas manejan un comportamiento similar. En este punto resta evaluar si realmente es preferible tener un modelo más complejo que tarda más en entrenar y predecir para predicciones muy similares e incluso con diferencias visuales casi imperceptibles.

4.4 Modelo GRU

Los modelos GRU (Gated Recurrent Unit) son una variante de las redes neuronales recurrentes similares a los modelos LSTM en términos de funcionalidad. Sin embargo, los GRU presentan una estructura más sencilla, con menos parámetros, lo que los hace más eficientes en ciertos escenarios. Esto puede resultar particularmente útil cuando los datos presentan una estructura temporal bien definida sin requerir la complejidad de los LSTM.

En este contexto, se realizaron pruebas similares a las efectuadas con los modelos LSTM. Para ello, se utilizaron los tres conjuntos de datos con intervalos de tiempo de uno, dos y cinco minutos. Se evaluaron dos tipos de modelos de predicción: uno simple, que únicamente considera latitud y longitud, y otro más complejo, que incorpora variables adicionales (las mismas utilizadas previamente en los modelos

LSTM). Para facilitar la comparación de rendimientos, se presentan los resultados de las métricas y algunas visualizaciones de mapas con las predicciones encadenadas. Cabe destacar que, en esta ocasión, solo se emplearon los conjuntos de datos filtrados (sin los registros donde los colectivos están detenidos), dado que, aunque las métricas no evidenciaron mejoras significativas, se considera que estos datos presentan mayor coherencia y depuración.

En el caso del modelo simple, las métricas obtenidas para todas las líneas fueron muy similares a las registradas con el modelo LSTM en las mismas condiciones.

Línea	Intervalos	Error Absoluto Medio (MAE)	Error Cuadrático Medio (MSE)
500	Cada 1 minuto	0.000872	2.415835 e-6
	Cada 2 minutos	0.001276	4.765191 e-6
	Cada 5 minutos	0.002564	1.533017 e-5
501	Cada 1 minuto	0.001005	2.991647 e-6
	Cada 2 minutos	0.001673	6.861352 e-6
	Cada 5 minutos	0.002791	1.758618 e-5
502	Cada 1 minuto	0.000983	2.911050 e-6
	Cada 2 minutos	0.001577	6.680023 e-6
	Cada 5 minutos	0.003369	2.658599 e-5
503	Cada 1 minuto	0.001124	5.917646 e-6
	Cada 2 minutos	0.001764	1.185121 e-5
	Cada 5 minutos	0.003825	3.943826 e-5
504	Cada 1 minuto	0.001037	3.170721 e-6
	Cada 2 minutos	0.001482	6.28540 e-6
	Cada 5 minutos	0.003027	1.964054 e-5
505	Cada 1 minuto	0.001862	1.179882 e-5
	Cada 2 minutos	0.003029	2.868373 e-5
	Cada 5 minutos	0.004982	7.250062 e-5

Tabla 7 Métricas del Modelo GRU Simple para cada Conjunto de Datos.

En esta tabla se puede observar un comportamiento muy similar a lo que pasaba con los modelos LSTM. Por ello, al igual que se hizo en la sección anterior, se creó un mapa para visualizar las predicciones encadenadas de la línea 500 y se observó una menor precisión en comparación con LSTM. Tras algunas pruebas, se vio que los resultados finales presentaban una mayor desviación respecto a los valores esperados, incluso en el conjunto de datos con el menor intervalo de tiempo.

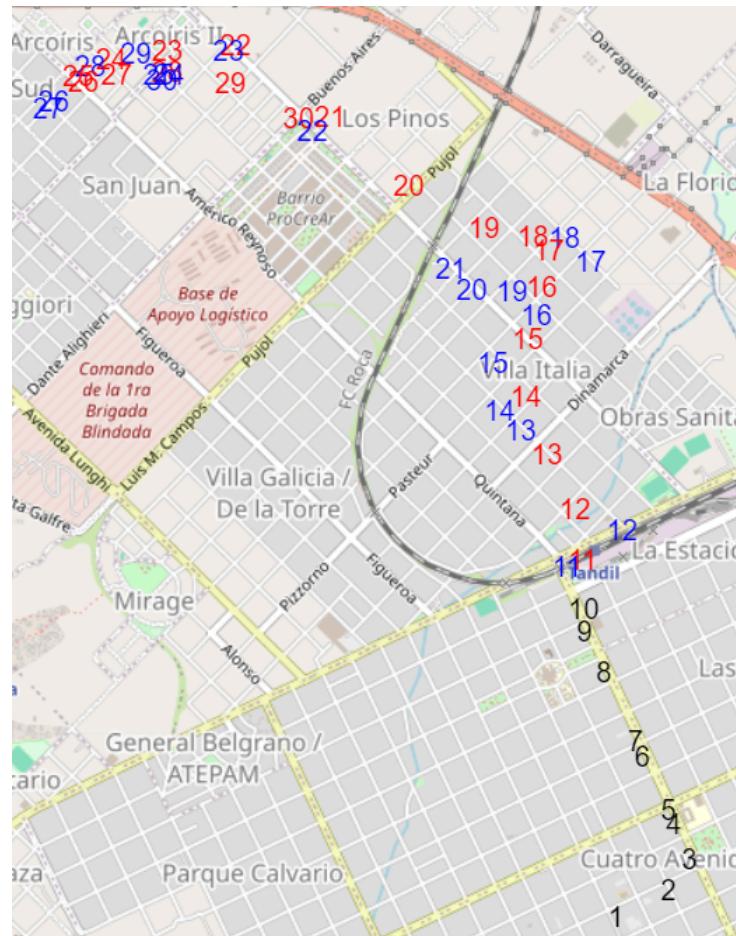


Figura 27 Predicciones del Modelo GRU Simple con Intervalos de 1 Minuto para la Línea 500.

No obstante, para confirmar si las predicciones encadenadas son efectivamente menos precisas, sería necesario comparar ambos modelos bajo una misma ventana base. Esta evaluación se realizará en la sección posterior para determinar cuál de los modelos es más adecuado para la tarea de predicción.

Posteriormente, se implementó el segundo modelo, más complejo y con un mayor número de variables de entrada. Siguiendo la misma metodología aplicada en el modelo LSTM, se incluyó la predicción de la velocidad, aunque esta variable no fue considerada en la evaluación de métricas, y se testeó con la línea 500. Los resultados obtenidos se presentan a continuación.

Intervalos	Error Absoluto Medio (MAE)	Error Cuadrático Medio (MSE)
Cada 1 minuto	0.000797	1.645412 e-6
Cada 2 minutos	0.001316	3.708445 e-6
Cada 5 minutos	0.002482	1.263330 e-5

Tabla 8 Métricas del Modelo GRU Complejo para cada Conjunto de Datos de la Línea 500.

Las predicciones encadenadas del modelo complejo mostraron un comportamiento similar al del modelo simple, con algunos casos de alta precisión y otros con errores de varias cuadras. Sin embargo, considerando intervalos de un minuto, se identificó un error de aproximadamente dos pasos, lo que equivale a una desviación de entre uno y dos minutos en la estimación para un horizonte de veinte minutos. Este margen de error podría ser aceptable dentro del contexto de la aplicación.



Figura 28 Predicciones del Modelo GRU Simple con Intervalos de 1 Minuto.

Con estos resultados, queda pendiente determinar cuál de los modelos es el más adecuado para la predicción del tiempo de llegada de los colectivos.

4.5 Comparación de Resultados

En esta sección se presentan los resultados obtenidos tras implementar y evaluar los distintos modelos utilizados para resolver el problema de estimación de tiempos de colectivos. Los modelos considerados fueron Regresión Lineal, ARIMA, LSTM y GRU. A partir de las métricas calculadas y la observación de su desempeño, se analizaron sus ventajas y limitaciones con el objetivo de seleccionar el modelo más adecuado.

Los modelos de Regresión Lineal y ARIMA fueron descartados tempranamente debido a su incapacidad para ajustarse a la complejidad inherente de los datos. En el caso de la Regresión Lineal, sus predicciones resultaron poco precisas, con errores significativamente altos reflejados en métricas como el MAE y el RMSE, lo que evidencia su falta de capacidad para capturar patrones no lineales y variaciones temporales frecuentes, propias de un sistema que depende de factores dinámicos como los tiempos de llegada de los colectivos. Por otro lado, el modelo ARIMA, aunque mejor adaptado para series temporales, presentó un rendimiento igualmente insatisfactorio. A pesar de su capacidad para modelar estacionalidad, su desempeño decayó en presencia de cambios abruptos y alta variabilidad en los datos.

Dado el pobre desempeño de estos dos modelos, se decidió centrarse en las arquitecturas LSTM y GRU, las cuales mostraron un rendimiento considerablemente superior en comparación con los enfoques anteriores. Ambos modelos fueron entrenados y evaluados utilizando tres conjuntos de datos distintos, segmentados según los intervalos de tiempo entre mediciones: uno con intervalos de 1 minuto, otro con intervalos de 2 minutos, y un último con intervalos de 5 minutos. El objetivo de esta segmentación fue determinar la influencia de la frecuencia de medición en el rendimiento de los modelos, especialmente en términos de precisión y costo computacional. Además, para cada conjunto de datos, se desarrollaron dos enfoques de modelado: uno simple, basado únicamente en las coordenadas de latitud y longitud, y otro complejo, que incluye variables adicionales (como la velocidad actual, hora y día) y que también realiza la predicción de la velocidad del colectivo como salida secundaria.

Los resultados obtenidos para LSTM demostraron que este modelo es capaz de capturar con precisión los patrones temporales complejos presentes en los datos. Al evaluar su desempeño con los distintos conjuntos de datos, se observó que el uso de intervalos de 1 minuto permitió alcanzar la mayor precisión en las predicciones, logrando una reducción significativa en los errores. Sin embargo, este beneficio tuvo un costo asociado, ya que el tiempo de entrenamiento requerido para procesar datos a intervalos tan frecuentes resultó considerablemente alto. Al aumentar los intervalos de tiempo a 2 y 5 minutos, el modelo mostró una notable pérdida de precisión, atribuida a la disminución en la cantidad de información temporal

disponible, lo cual limita su capacidad para capturar detalles y fluctuaciones que ocurren en intervalos más cortos.

Por su parte, el modelo GRU presentó un comportamiento similar en cuanto a precisión y sensibilidad a los intervalos de tiempo utilizados. Con datos segmentados en intervalos de 1 minuto, GRU alcanzó resultados comparables a los de LSTM en términos de precisión, con métricas de error muy cercanas. Al igual que LSTM, GRU mostró una caída en el rendimiento al aumentar los intervalos de tiempo a 2 y 5 minutos, lo que refuerza la importancia de utilizar mediciones más frecuentes para garantizar predicciones precisas.

Si bien el uso de intervalos de 1 minuto implica un mayor costo computacional debido a la cantidad de datos procesados, esta elección resulta coherente con la necesidad de ofrecer actualizaciones frecuentes y precisas a los usuarios, un aspecto fundamental para el correcto funcionamiento de la aplicación. Cualquiera sea la elección del modelo, el conjunto de datos adecuado para el mismo sería el de intervalos de 1 minuto.

Dicho esto los principales candidatos resultan los modelos LSTM y GRU con sus variantes simples y complejas. Sin embargo, las métricas que se fueron mostrando durante el análisis de cada uno son independientes entre sí ya que si bien reflejan valores de predicciones, los subconjuntos de datos elegidos para las mismas son diferentes, por lo tanto se consideró relevante analizar cómo son las predicciones encadenadas frente a ventanas base iguales. Entonces, esta evaluación de los cuatro modelos es tanto para el rendimiento en precisión de las predicciones como para el tiempo de entrenamiento y cálculo de los resultados. En primer lugar, la Tabla 9 muestra los tiempos de entrenamiento para los distintos modelos de las diferentes líneas, además de los promedios generales de cada modelo.

Línea	Modelo	Tiempo de Entrenamiento
500	GRU simple	531.992 segundos
	LSTM simple	505.397 segundos
	GRU complejo	862.302 segundos
	LSTM complejo	784.059 segundos
501	GRU simple	414.565 segundos
	LSTM simple	384.451 segundos
	GRU complejo	782.329 segundos
	LSTM complejo	782.348 segundos

502	GRU simple	476.864 segundos
	LSTM simple	453.014 segundos
	GRU complejo	900.428 segundos
	LSTM complejo	772.071 segundos
503	GRU simple	509.841 segundos
	LSTM simple	427.147 segundos
	GRU complejo	843.859 segundos
	LSTM complejo	858.832 segundos
504	GRU simple	420.465 segundos
	LSTM simple	399.969 segundos
	GRU complejo	762.256 segundos
	LSTM complejo	709.761 segundos
505	GRU simple	456.136 segundos
	LSTM simple	401.285 segundos
	GRU complejo	815.527 segundos
	LSTM complejo	765.212 segundos
Promedio	GRU simple	468.311 segundos
	LSTM simple	428.544 segundos
	GRU complejo	827.783 segundos
	LSTM complejo	778.714 segundos

Tabla 9 Comparación de Tiempo de Entrenamiento para los Distintos Modelos.

Al analizar los tiempos de entrenamiento de los modelos evaluados, se observa que los modelos simples (aquellos que utilizan únicamente las variables de latitud y longitud) muestran diferencias significativas con respecto a los más complejos, pero no tanto entre sí.

En los modelos complejos (que incluyen variables adicionales y predicen también la velocidad), los tiempos de entrenamiento fueron mayores debido al incremento en la cantidad de datos y parámetros a procesar. Sin embargo, aunque el GRU complejo tarda en líneas generales un poco más que el LSTM complejo, las mayores diferencias se muestran con respecto a sus variantes simples y no tanto entre sí como se analizó anteriormente.

La hipótesis de que los GRU son más rápidos que los LSTM no se confirmó en este caso. Tanto para las variantes simples o complejas, los LSTM mostraron un tiempo ligeramente menor con respecto a los GRU de la misma complejidad. Esto puede deberse a configuraciones específicas de los modelos, aunque los GRU tienen menos parámetros en general, ciertas configuraciones (como el tamaño de las capas, la cantidad de datos o el optimizador utilizado) pueden afectar la velocidad, o tambien puede ser por la naturaleza del problema, los LSTM podrían estar aprovechando mejor los patrones temporales específicos de los datos de entrada, lo que les permite entrenarse más rápido en este caso.

Los resultados de las métricas de error obtenidas para los modelos GRU y LSTM (en sus variantes simple y compleja) permiten hacer un análisis de su desempeño. Tener en cuenta que estas métricas muestran únicamente el error asociado a la predicción de un solo paso y no en predicciones consecutivas.

Línea	Modelo	Error Absoluto Medio (MAE)	Error Cuadrático Medio (MSE)
500	GRU simple	0.000915	2.565325 e-6
	LSTM simple	0.000837	2.373968 e-6
	GRU complejo	0.000870	2.439591 e-6
	LSTM complejo	0.000839	2.349658 e-6
501	GRU simple	0.000981	3.105133 e-6
	LSTM simple	0.000957	3.056931 e-6
	GRU complejo	0.000964	3.091174 e-6
	LSTM complejo	0.000935	3.075534 e-6
502	GRU simple	0.000877	2.086393 e-6
	LSTM simple	0.000900	2.088769 e-6
	GRU complejo	0.000892	2.078263 e-6
	LSTM complejo	0.000866	1.983135 e-6

503	GRU simple	0.001174	5.639501 e-6
	LSTM simple	0.001099	5.519443 e-6
	GRU complejo	0.001135	5.598680 e-6
	LSTM complejo	0.001149	5.622387 e-6
504	GRU simple	0.000937	3.054201 e-6
	LSTM simple	0.000963	3.093351 e-6
	GRU complejo	0.000897	3.015672 e-6
	LSTM complejo	0.000915	2.904847 e-6
505	GRU simple	0.003137	2.907437 e-5
	LSTM simple	0.003119	2.817261 e-5
	GRU complejo	0.003151	2.913174 e-5
	LSTM complejo	0.003079	2.772044 e-5

Tabla 10 Comparación de Métricas para los Distintos Modelos.

En la Tabla 10 se observa que las métricas obtenidas para los distintos modelos muestra que, en general, tanto las arquitecturas GRU como LSTM logran un desempeño similar en la mayoría de las líneas, con diferencias leves en el MAE y el MSE. No obstante, en términos generales, los modelos LSTM complejos tienden a obtener los mejores resultados, con los valores de error más bajos en varias líneas, lo que sugiere una mejor capacidad para capturar la secuencia temporal de los datos.

En las líneas 500, 501, 502 y 504, las diferencias entre los modelos son mínimas, lo que indica que cualquier variante de GRU o LSTM puede ser utilizada con buenos resultados. Sin embargo, en la línea 503, los errores son mayores para todos los modelos, resultando que las mejores métricas se obtienen con el LSTM simple. Por otro lado, la línea 505 presenta los errores más elevados, lo que confirma que sus tiempos son los más impredecibles. En este caso, los modelos LSTM complejos muestran un desempeño ligeramente superior, con el menor MSE en comparación con las otras variantes.

En términos generales, el LSTM complejo es el modelo con los mejores resultados, ya que en la mayoría de las líneas obtiene los menores valores de error. Aunque las diferencias son pequeñas, su capacidad para capturar dependencias a largo plazo podría ser la clave para su mejor desempeño.

Para poder tomar una buena decisión no basta con evaluar los tiempos de entrenamiento y las métricas de predicciones de un solo paso, sino más bien se consideró que podría ser relevante también medir los tiempos que tarda el modelo en realizar las predicciones consecutivas y las precisiones de las mismas, ya que esto nos dará una mejor idea de cual es el más adecuado. Para ello, en primer lugar se muestra la Tabla 11 con los tiempos de 20 predicciones consecutivas para cada variante de modelo de cada línea.

Línea	Modelo	Tiempo de 20 predicciones consecutivas
500	GRU simple	1.3180558681488037 segundos
	LSTM simple	1.2664837837219238 segundos
	GRU complejo	1.3492088317871094 segundos
	LSTM complejo	1.4504499435424805 segundos
501	GRU simple	1.9972646236419678 segundos
	LSTM simple	1.5108323097229004 segundos
	GRU complejo	1.4635677337646484 segundos
	LSTM complejo	1.5075252056121826 segundos
502	GRU simple	1.8220880031585693 segundos
	LSTM simple	1.3494832515716553 segundos
	GRU complejo	1.316563606262207 segundos
	LSTM complejo	1.332930326461792 segundos
503	GRU simple	1.3320276737213135 segundos
	LSTM simple	1.316408634185791 segundos
	GRU complejo	1.3164019584655762 segundos
	LSTM complejo	1.3331425189971924 segundos
504	GRU simple	1.1363263130187988 segundos
	LSTM simple	1.2830007076263428 segundos
	GRU complejo	1.2513277530670166 segundos
	LSTM complejo	1.249300241470337 segundos

505	GRU simple	1.161858081817627 segundos
	LSTM simple	1.2248733043670654 segundos
	GRU complejo	1.2903013229370117 segundos
	LSTM complejo	1.3475115299224854 segundos
Promedio	GRU simple	1.4612700939178467 segundos
	LSTM simple	1.3251803318659465 segundos
	GRU complejo	1.331228534380595 segundos
	LSTM complejo	1.3701432943344116 segundos

Tabla 11 Comparación de Tiempos de Predicción para los Distintos Modelos.

Dado que las diferencias en los tiempos de predicción son mínimas y todas rondan cerca del segundo, este aspecto no constituye un criterio determinante para la elección del modelo. Por lo tanto, se debería priorizar el análisis basado en precisión de un solo paso y en cadena, ya que estos factores tienen un impacto más significativo en la viabilidad y calidad de la solución.

Para evaluar el desempeño de los cuatro modelos desarrollados, se realizaron predicciones en cadena, generando más de 7000 predicciones de 20 pasos cada una, las cuales tardaron alrededor de un día en ejecutarse para cada una de las líneas. Las métricas analizadas son las siguientes:

- Media: Representa la distancia promedio entre las predicciones y los valores reales. Una media más baja indica predicciones más precisas en promedio.
- Mediana: Es el valor central de las distancias ordenadas. Es útil para entender el comportamiento central del modelo, especialmente cuando existen valores atípicos.
- Máximo y Mínimo: Reflejan las predicciones más alejadas y más cercanas a la realidad, respectivamente.
- Desviación estándar: Mide la dispersión de las distancias. Una desviación estándar más baja indica predicciones más consistentes.
- Percentiles (25%, 50%, 75%): Ayudan a entender la distribución de los errores y detectar cómo se concentran las distancias.

Hecha esta aclaración los resultados de estas métricas para cada modelo se presentan en las Tablas 12 a 17, correspondientes a cada una de las líneas de colectivos.

Modelo	Media	Mediana	Máximo	Mínimo	Desviación Estándar	Percentiles (25%, 50%, 75%)
GRU simple	394.85 metros	252.95 metros	6132.92 metros	0.78 metros	481.99 metros	134.18540216, 252.95491064, 477.1711774
LSTM simple	663.18 metros	371.01 metros	6186.82 metros	0.36 metros	820.04 metros	184.37252625, 371.01445409, 778.34199416
GRU complejo	438.58 metros	278.29 metros	6232.74 metros	0.19 metros	525.82 metros	139.01615187, 278.28796737, 554.9984271
LSTM complejo	345.77 metros	222.74 metros	6129.45 metros	0.84 metros	448.92 metros	121.00298056, 222.74330525, 416.63694723

Tabla 12 Comparación de Métricas para Predicciones Encadenadas en los Distintos Modelos de la Línea 500.

Modelo	Media	Mediana	Máximo	Mínimo	Desviación Estándar	Percentiles (25%, 50%, 75%)
GRU simple	625.58 metros	381.28 metros	4811.71 metros	2.05 metros	687.81 metros	198.82954563, 381.27653441, 765.25078648
LSTM simple	694.31 metros	407.90 metros	5055.90 metros	1.17 metros	838.91 metros	201.99651283, 407.89938089, 801.18928586
GRU complejo	406.63 metros	315.78 metros	3239.70 metros	1.21 metros	326.77 metros	177.17024552, 315.78301336, 536.35881914
LSTM complejo	675.27 metros	448.24 metros	5177.41 metros	0.67 metros	689.73 metros	223.53103356, 448.24085984, 869.68509000

Tabla 13 Comparación de Métricas para Predicciones Encadenadas en los Distintos Modelos de la Línea 501.

Modelo	Media	Mediana	Máximo	Mínimo	Desviación Estándar	Percentiles (25%, 50%, 75%)
GRU simple	498.13 metros	295.53 metros	5209.94 metros	0.89 metros	593.42 metros	148.12486661, 295.52809416, 603.57190845
LSTM simple	385.53 metros	244.22 metros	5335.45 metros	0.12 metros	451.00 metros	117.70771969, 244.21640013, 490.19817256
GRU complejo	365.51 metros	235.34 metros	5167.32 metros	0.65 metros	416.01 metros	121.95913931, 235.3428248, 456.78597796
LSTM complejo	395.56 metros	270.82 metros	4890.52 metros	0.15 metros	405.79 metros	137.82928066, 270.82092124, 519.85707397

Tabla 14 Comparación de Métricas para Predicciones Encadenadas en los Distintos Modelos de la Línea 502.

Modelo	Media	Mediana	Máximo	Mínimo	Desviación Estándar	Percentiles (25%, 50%, 75%)
GRU simple	888.63 metros	595.59 metros	8953.49 metros	1.15 metros	986.16 metros	290.12583427, 595.59289412, 1136.41569965
LSTM simple	669.15 metros	382.69 metros	8924.47 metros	0.04 metros	945.34 metros	192.8145577, 382.68683672, 761.4109667
GRU complejo	597.12 metros	350.10 metros	9165.06 metros	0.30 metros	845.23 metros	180.05151574, 350.09587712, 665.04156553
LSTM complejo	570.94 metros	335.97 metros	8841.60 metros	0.66 metros	816.12 metros	181.61022406, 335.97452631, 633.41902643

Tabla 15 Comparación de Métricas para Predicciones Encadenadas en los Distintos Modelos de la Línea 503.

Modelo	Media	Mediana	Máximo	Mínimo	Desviación Estándar	Percentiles (25%, 50%, 75%)
GRU simple	392.03 metros	261.15 metros	4265.76 metros	0.62 metros	451.56 metros	141.77891878, 261.1508877, 482.93076769
LSTM simple	479.26 metros	305.09 metros	4231.50 metros	0.71 metros	525.22 metros	159.46831256, 305.09436185, 598.2602666
GRU complejo	572.00 metros	406.04 metros	4267.84 metros	0.22 metros	544.15 metros	193.6233687, 406.03867674, 783.76940334
LSTM complejo	364.95 metros	233.40 metros	4279.85 metros	0.63 metros	443.61 metros	121.30545303, 233.39976478, 449.13888768

Tabla 16 Comparación de Métricas para Predicciones Encadenadas en los Distintos Modelos de la Línea 504.

Modelo	Media	Mediana	Máximo	Mínimo	Desviación Estándar	Percentiles (25%, 50%, 75%)
GRU simple	1367.26 metros	960.18 metros	7116.72 metros	1.11 metros	1220.77 metros	405.14733269, 960.1773379, 2042.28404766
LSTM simple	1269.43 metros	846.62 metros	7084.69 metros	1.24 metros	1213.76 metros	301.2789278, 846.62437424, 1925.97557763
GRU complejo	1265.83 metros	867.97 metros	6992.73 metros	0.64 metros	1189.85 metros	308.20959205, 867.96730572, 1924.80416397
LSTM complejo	1268.29 metros	854.31 metros	6981.30 metros	0.78 metros	1213.93 metros	293.8357923, 854.31481887, 1920.46428327

Tabla 17 Comparación de Métricas para Predicciones Encadenadas en los Distintos Modelos de la Línea 505.

Tras analizar los resultados de todas las líneas evaluadas, se observa que no hay un modelo que sea consistentemente el mejor en todas las métricas, aunque algunos muestran un rendimiento más estable en distintos escenarios.

El modelo LSTM complejo destaca en varias líneas con valores de media y mediana más bajos, lo que indica predicciones más cercanas a los valores reales. Además, sus desviaciones estándar suelen ser menores en comparación con otros modelos, lo que refleja mayor estabilidad en las predicciones. Un ejemplo claro es la Línea 500, donde el LSTM complejo obtuvo la media más baja (345.77 metros) y una desviación estándar relativamente baja (448.92 metros), lo que sugiere predicciones más consistentes.

Por otro lado, los modelos GRU y LSTM simples presentan medias y desviaciones estándar más elevadas en la mayoría de las líneas, lo que sugiere predicciones más dispersas y menos confiables. Por ejemplo, en la Línea 502, el modelo GRU simple presentó una media de 498.13 metros y una desviación estándar de 593.42 metros, lo que indica una mayor dispersión en las predicciones.

El modelo GRU complejo, aunque en algunos casos no alcanza la precisión del LSTM complejo, presenta una media y mediana competitivas, con desviaciones estándar más bajas en ciertas líneas, como en la Línea 501, donde su media fue de 406.63 metros y la desviación estándar de 326.77 metros, superando incluso al LSTM complejo en consistencia.

Un dato a tener en cuenta son los valores elevados con diferencia para todas las variantes de la línea 505. Si bien llamaron la atención al momento de hacer la revisión, lo que interesa de esta comparativa son las diferencias entre modelos y no tanto entre líneas, ya que para un mismo conjunto de datos todas las variantes de esta línea se mantienen en valores similares, y no es para alarmarse su error elevado ya que se testeó con otros subconjuntos de datos y funciona correctamente.

En términos generales, el LSTM complejo parece ser la mejor opción en la mayoría de los escenarios, ya que logra un balance entre precisión y estabilidad en sus predicciones.

Por otra parte, los valores máximos fueron un aspecto a prestar atención, si bien en todos son relativamente parecidos, resultan demasiado elevados para considerarse una predicción válida, y se esperaba que con todo el preprocesamiento que se hizo de los datos no iba a haber este tipo de errores tan groseros. Si existen valores extremos o anomalías el modelo puede generar predicciones significativamente alejadas de los valores reales. Aunque el preprocesamiento puede mitigar estos casos, es posible que algunos datos problemáticos permanezcan y afecten el desempeño. En predicciones en cadena, los errores de pasos anteriores se propagan y amplifican en los pasos subsecuentes, lo que podría explicar las desviaciones significativas en predicciones más lejanas y por ende los valores tan elevados.

Para corroborar a que se deben estos valores atípicos, se observó que hay brechas de tiempo sin registros, es decir, la gran mayoría de los registros están en lapsos de un minuto pero hay algunos registros que pasa más tiempo. Para poder observar esto se muestra un diagrama de dispersión entre las distancias consecutivas entre registros (en metros) y sus correspondientes diferencias de tiempos (en segundos).

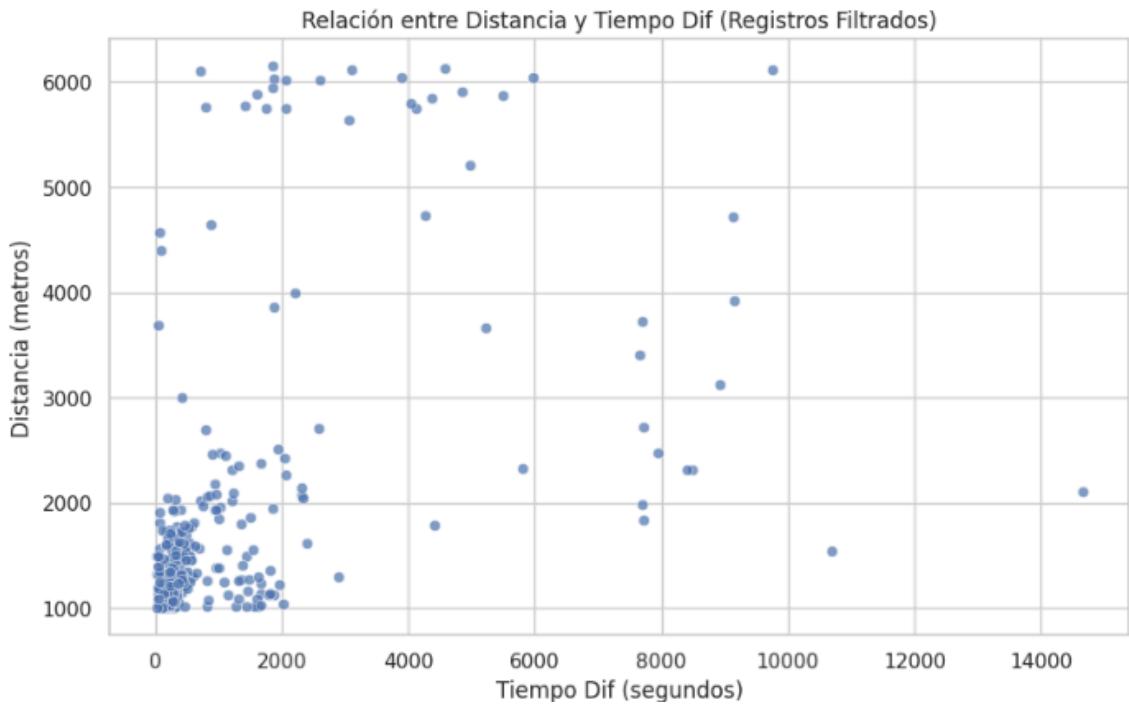


Figura 29 Gráfico de Correlación entre la Distancia y la Diferencia de Tiempo entre Dos Registros Consecutivos.

El gráfico de la Figura 29 se hizo para aquellos valores de distancia mayores a 1000 metros de la línea 500, y si bien algunos casos la diferencia de tiempo es chica y la distancia es grande se puede ver que hay varios valores superando los 2000 segundos, lo cual está muy por encima del minuto de intervalo predefinido. Si bien no hay una altísima correlación entre estas dos variables, la misma es relevante como para tenerlo en cuenta. Con esto se busca dejar en claro que no hay un error grande en la predicción sino que se debe a registros aislados en el tiempo, por lo que la que el modelo está prediciendo lo que va a pasar en el próximo minuto y lo compara con un dato que es mucho después, es decir que el análisis previo de cuál es el mejor modelo no se ve afectado.

Basándose en el análisis exhaustivo de los modelos y las métricas obtenidas, se puede llegar a las siguientes conclusiones para tomar una decisión informada sobre el modelo más adecuado.

El modelo LSTM complejo se destacó con un mejor desempeño que las demás arquitecturas tanto para predicciones simples de un solo paso como para las encadenadas. Esto demuestra que su capacidad para manejar dependencias

temporales a largo plazo es crucial para obtener mejores resultados en este tipo de predicciones.

En cuanto a los tiempos de entrenamiento, no representan un inconveniente significativo, ya que este proceso se realiza una sola vez y su impacto es marginal en la implementación del modelo. No obstante, además de ser mejor en precisión con respecto a su contraparte compleja, el LSTM también tuvo mejores tiempos que el GRU. Por otro lado, los tiempos de predicción de las distintas arquitecturas son muy similares y pequeños, por lo que este aspecto no influye en la elección del modelo.

En conclusión, considerando que el objetivo principal es optimizar las predicciones encadenadas, el modelo LSTM complejo es el más adecuado.

Capítulo 5: Diseño e Implementación de la Aplicación Móvil

En este capítulo se presenta el desarrollo de la aplicación móvil BusNow. La misma tiene como objetivo principal funcionar como una herramienta que facilite el uso del transporte público para las personas de la ciudad de Tandil, incorporando mayor cantidad y calidad de información en la toma de decisiones de los usuarios de las distintas líneas de colectivos. En la búsqueda del cumplimiento de este objetivo fue necesario definir cuales son las características que debe tener la app para resultar relevante y pertinente para los usuarios.

Al ingresar a la aplicación, el usuario puede visualizar un mapa interactivo que muestra todas las líneas de colectivos de la ciudad junto con sus recorridos y paradas. Desde esta vista, es posible seleccionar una línea específica para ver su trayecto completo, con cada una de sus paradas identificadas claramente. Aquí también es posible visualizar la ubicación actual de los colectivos en circulación, permitiendo que los usuarios puedan saber en qué punto del recorrido se encuentra en cada momento y cuánto tiempo falta para que lleguen a una parada determinada. Esta funcionalidad es clave para quienes desean minimizar su tiempo de espera y asegurarse de no perder el colectivo.

Otra de las características fundamentales de la aplicación es la capacidad de calcular recorridos desde un punto de origen hasta un destino dentro de la ciudad, utilizando las diferentes líneas de colectivos disponibles. Para ello, el usuario puede ingresar manualmente su ubicación de partida y su destino o bien seleccionar este último directamente en el mapa. La aplicación analizará todas las combinaciones posibles de líneas y trayectos para ofrecer la mejor alternativa de viaje, teniendo en cuenta factores como la caminata hasta las paradas, la distancia recorrida en colectivo y los tiempos estimados. De esta manera, los usuarios pueden elegir la opción que mejor se adapte a sus necesidades.

Para una experiencia más personalizada, la aplicación permite a los usuarios guardar información relevante, como búsquedas recientes y lugares favoritos. Esto facilita la consulta rápida de rutas y tiempos de viaje sin necesidad de ingresar los mismos datos en cada ocasión.

Desde la pantalla principal, los usuarios pueden explorar el mapa, buscar líneas y paradas, ingresar destinos para calcular rutas y acceder a su historial de viajes recientes o a sus ubicaciones guardadas. La navegación dentro de la aplicación es fluida y ágil, asegurando una experiencia cómoda y eficiente.

La aplicación ofrece una solución integral para los usuarios del transporte público en Tandil, combinando la visualización de rutas y colectivos en tiempo real con herramientas de planificación de viajes y funcionalidades personalizadas.

Conforme avance este capítulo se irán desarrollando las distintas decisiones que se fueron tomando a lo largo del proceso de desarrollo de la aplicación, comenzando por los requerimientos funcionales y no funcionales, siguiendo por las tecnologías utilizadas, el procesamiento de la información correspondiente a las paradas y recorridos de cada línea, continuando por el diseño de la interfaz y la experiencia de usuario, para finalizar con los aspectos más técnicos de la implementación de la aplicación móvil.

5.1 Requerimientos Funcionales y No Funcionales

Para garantizar que la aplicación cumpla con sus objetivos y resulte útil para los usuarios, se han definido una serie de requerimientos funcionales y no funcionales que guían su desarrollo. Estos requerimientos han sido establecidos a partir del análisis de aplicaciones similares, hecho en el Capítulo 2 Sección 1 "Análisis de Apps Existentes", tomando en cuenta sus características más relevantes en el ámbito del transporte público. Además, se han considerado las necesidades específicas de los usuarios, con énfasis en la importancia de contar con información precisa y en tiempo real sobre la ubicación de los colectivos. La aplicación busca ofrecer una experiencia intuitiva y eficiente, permitiendo a los pasajeros conocer el tiempo estimado de llegada de los colectivos a su parada y optimizar la planificación de sus viajes.

En este apartado se detallarán los requerimientos funcionales, que describen las características y comportamientos esenciales de la aplicación, así como los requerimientos no funcionales, que establecen los criterios de calidad, rendimiento, seguridad y usabilidad. También se indicará cuáles de estos requerimientos estarán presentes en la versión inicial y cuáles se considerarán para futuras actualizaciones.

5.1.1 Requerimientos Funcionales

Entre las funcionalidades clave que la aplicación debe ofrecer, se incluyen:

- Ubicación en Tiempo Real: La aplicación debe detectar y mostrar la ubicación del usuario en un mapa, así como la ubicación de los colectivos en tiempo real, utilizando la información proporcionada por los propios pasajeros.

- Recorridos y Paradas de Colectivos: Se debe brindar acceso a los recorridos de las distintas líneas, permitiendo la búsqueda de recorridos y visualización de paradas con estimaciones de tiempo de llegada.
- Estimaciones de Tiempo: La aplicación debe calcular y mostrar el tiempo estimado de llegada de los colectivos a una parada determinada, optimizando así la planificación del viaje del usuario.
- Gestión de Cuentas de Usuario: Se debe permitir el registro, inicio de sesión y gestión de cuentas, con la posibilidad de guardar rutas y lugares favoritos, así como personalizar las preferencias del usuario.
- Reportes en Tiempo Real: Los usuarios deben poder informar incidencias como retrasos o cambios en los recorridos, y estos reportes deben reflejarse en la aplicación en beneficio de otros pasajeros.
- Notificaciones: Se deben enviar alertas sobre la llegada del colectivo, incidencias en la ruta y avisos personalizados, como recordatorios de cuándo salir de casa para llegar a tiempo a la parada.

5.1.2 Requerimientos No Funcionales

Para garantizar la calidad y confiabilidad del sistema, se establecen los siguientes criterios no funcionales:

- Disponibilidad y Rendimiento: La aplicación debe manejar múltiples solicitudes sin afectar la velocidad ni la funcionalidad, garantizando actualizaciones en tiempo real con mínima latencia.
- Usabilidad: La interfaz debe ser intuitiva y accesible, asegurando una navegación fluida. En versiones futuras, se podrá incluir una guía inicial para nuevos usuarios.
- Privacidad y Anonimidad: Se debe garantizar que la información de ubicación y reportes de los usuarios sea anónima para proteger su privacidad.
- Seguridad de la Información: Las cuentas de usuario deben contar con medidas de seguridad, incluyendo autenticación segura y protección de credenciales.
- Compatibilidad: La aplicación debe ser funcional en dispositivos Android y adaptarse a distintas resoluciones de pantalla.
- Mantenimiento y Actualización: Se debe permitir la corrección de errores y la incorporación de nuevos recorridos o cambios en los mismos de forma dinámica.

5.1.3 Priorización de Requerimientos

Para optimizar el proceso de desarrollo, los requerimientos se han categorizado en función de su prioridad:

- Versión Inicial (MVP): Se enfocará en las funcionalidades esenciales, como ubicación en tiempo real, recorridos y paradas de colectivos, estimaciones de tiempo, además de garantizar el rendimiento y la usabilidad del sistema.
- Mejoras Adicionales: Se incorporará la gestión de cuentas de usuario junto con medidas de privacidad y seguridad.
- Versiones Futuras: Se añadirán reportes en tiempo real y notificaciones, junto con otras funcionalidades que puedan surgir en base a la retroalimentación de los usuarios.

Los criterios de compatibilidad, mantenimiento y actualización serán considerados de manera transversal en todo el proceso de desarrollo para asegurar la evolución continua de la aplicación.

5.2 Decisiones Tecnológicas

Tras realizar un análisis detallado de las diversas alternativas tecnológicas presentadas en las secciones 2 a 5 del Capítulo 2, se tomaron decisiones estratégicas orientadas a optimizar el rendimiento, la accesibilidad a la información y la adecuación a los requisitos específicos del proyecto. La elección de cada tecnología se fundamentó en su capacidad para garantizar un desarrollo eficiente, una implementación efectiva y una experiencia de usuario fluida y confiable.

Plataforma de Desarrollo: Android con Kotlin

Para la implementación de la aplicación, se optó por desarrollar exclusivamente para la plataforma Android utilizando Kotlin como lenguaje de programación. Esta elección responde a múltiples factores clave. En primer lugar, Kotlin es un lenguaje nativo para Android, lo que asegura un rendimiento óptimo en dispositivos con este sistema operativo. Su sintaxis moderna y concisa facilita la escritura de código limpio y eficiente, reduciendo la posibilidad de errores y mejorando la mantenibilidad del software. Además, Kotlin ofrece una integración total con las APIs de Android y es oficialmente respaldado por Google, lo que garantiza un soporte continuo y actualizaciones constantes.

Otro aspecto determinante en esta decisión es la disponibilidad de dispositivos de prueba. Dado que durante el desarrollo de la aplicación se contó exclusivamente con dispositivos Android para realizar pruebas y depuración, el desarrollo para otras plataformas, como iOS, no resultaría viable ni eficiente, ya que no podría garantizar un correcto funcionamiento sin un entorno de prueba adecuado. Por lo tanto, se

priorizó un enfoque centrado en Android, asegurando una implementación efectiva y un proceso de desarrollo más controlado.

Para desarrollar en esta plataforma se contó con el IDE de Android Studio el cual posee toda la funcionalidad necesaria para el desarrollo de aplicaciones móviles en este entorno, ofreciendo una organización clara del proyecto, la posibilidad de trabajar de forma intuitiva las interfaces gráficas e incluso emular distintos dispositivos móviles con distintas versiones de sistema operativo y recursos tecnológicos.

API de Geolocalización: Google Maps

Para la integración de funcionalidades relacionadas con la geolocalización, se eligió la API de Google Maps. Esta elección se debe a su amplia adopción en el desarrollo de aplicaciones de mapas y navegación, así como a la calidad de sus herramientas y documentación. Google Maps proporciona un conjunto robusto de funcionalidades, incluyendo la visualización dinámica de mapas, la búsqueda precisa de ubicaciones y la generación de rutas en tiempo real.

Además, esta API se destaca por su confiabilidad y escalabilidad, lo que permite manejar eficientemente un alto volumen de solicitudes sin comprometer el rendimiento. La facilidad de integración con otras herramientas de Google, como Google Places y Google Directions, añade un valor significativo a la aplicación, permitiendo la implementación de características avanzadas que enriquecen la experiencia del usuario. Otro aspecto a tener en cuenta fue la adaptabilidad de los usuarios finales a la aplicación, los cuales en su mayoría están familiarizados con las interfaces que ofrece la app de Google Maps y saben manejarse en ese entorno, lo cual resulta en una experiencia más amigable.

Base de Datos: Firebase Realtime Database

Para la gestión y almacenamiento de datos en la aplicación, se optó por utilizar Firebase Realtime Database. Esta decisión se fundamenta en la necesidad de contar con una base de datos que ofrezca sincronización en tiempo real, una característica esencial para actualizar constantemente la ubicación de los colectivos y proporcionar información precisa a los usuarios.

Firebase Realtime Database permite la actualización instantánea de los datos sin requerir intervenciones manuales o recargas en la aplicación, lo que resulta crucial para el correcto funcionamiento del sistema de seguimiento de colectivos. Además, esta plataforma ofrece soporte para notificaciones push, facilitando la comunicación en tiempo real con los usuarios y manteniéndolos informados sobre eventos relevantes.

Otro aspecto relevante de Firebase es su integración con otros servicios de Google, incluyendo autenticación de usuarios, almacenamiento en la nube y análisis de datos, lo que simplifica considerablemente el desarrollo y la escalabilidad del proyecto. Su estructura basada en JSON y su capacidad de gestionar grandes volúmenes de datos de manera eficiente lo convierten en una solución idónea para los requerimientos de la aplicación.

Cálculo de Rutas y Actualización de Ubicaciones: Solución Híbrida

En cuanto al cálculo de rutas y la actualización de ubicaciones, se decidió adoptar un enfoque híbrido que combina el procesamiento en el cliente con la gestión de ubicaciones en el servidor. Esta estrategia permite aprovechar las ventajas de ambos modelos para lograr una aplicación más eficiente y con menor carga en cada uno de los componentes del sistema.

El cálculo de rutas se realiza directamente en el cliente, utilizando la capacidad de procesamiento del dispositivo para generar rutas y estimaciones de tiempo de llegada en tiempo real. Esta solución reduce la dependencia del servidor para operaciones que pueden ejecutarse localmente, mejorando la velocidad de respuesta y disminuyendo la latencia en la obtención de resultados.

Por otro lado, la actualización de ubicaciones se gestiona de manera centralizada en el servidor, asegurando la consistencia de los datos para todos los usuarios. A través de esta estrategia, se garantiza que la información de la ubicación de los colectivos esté siempre actualizada y sincronizada, evitando discrepancias entre distintos dispositivos y mejorando la precisión del servicio.

Este enfoque híbrido equilibra el uso de recursos, optimiza la experiencia del usuario y permite un sistema escalable y eficiente, adaptado a las necesidades del proyecto.

Todas las decisiones tecnológicas adoptadas en este proyecto responden a un análisis exhaustivo de las necesidades de la aplicación y las capacidades de cada herramienta disponible. La combinación de Kotlin como lenguaje de desarrollo, Google Maps para la geolocalización, Firebase Realtime Database para la gestión de datos y un enfoque híbrido para el cálculo de rutas y actualización de ubicaciones constituye una solución sólida y eficiente, alineada con los objetivos del proyecto y orientada a ofrecer una experiencia de usuario óptima y confiable.

5.3 Procesamiento de Paradas y Recorridos

Para integrar correctamente la información de paradas y recorridos de los colectivos en la aplicación, fue necesario recopilar datos desde distintas fuentes y someterlos a un proceso de limpieza y organización.

Los recorridos fueron obtenidos desde la página de transporte de la Municipalidad de Tandil (Gobierno Abierto Tandil - Recorrido de transporte | 2019)⁶, en un archivo Excel que contenía una serie de puntos representando los trayectos de cada línea. Sin embargo, estos datos no estaban en el formato georreferenciado de latitud y longitud utilizado en la aplicación, por lo que se requirió una conversión previa. Las coordenadas originales estaban en formato MULTILINESTRING, un formato utilizado en bases de datos geoespaciales y sistemas GIS para representar líneas múltiples. En este caso, los datos se almacenaban en un sistema de coordenadas proyectadas (UTM), lo que significa que las coordenadas estaban en metros en lugar de grados de latitud y longitud. Para convertirlas al sistema geográfico WGS84, cada punto se descompuso en sus valores X e Y, que corresponden a las coordenadas proyectadas en metros. Posteriormente, se aplicó una transformación de coordenadas que permitió convertir cada par de coordenadas del sistema UTM al sistema geográfico. Finalmente, los valores resultantes se almacenaron en una lista en formato (latitud, longitud), permitiendo su visualización en sistemas que usan coordenadas geográficas estándar, como Google Maps o aplicaciones GIS.

Las paradas de colectivos fueron obtenidas a través de la API de OpenStreetMap, que proporciona información sobre la infraestructura urbana. Sin embargo, se detectó una limitación importante: sólo estaban registradas las paradas ubicadas en el centro de la ciudad, mientras que en los barrios y alrededores, donde las líneas de colectivo no cuentan con paradas señalizadas físicamente, la API no devolvía resultados. Esto implicó la necesidad de completar manualmente la mayoría de las paradas para poder representar correctamente los trayectos en la aplicación.

A continuación, se detallan los procesos realizados para ordenar, corregir y completar la información de paradas y recorridos antes de ser almacenada en la base de datos.

⁶ <http://datos.tandil.gov.ar/dataset/mapas/archivo/fad7652f-b15f-45e5-9e64-c6461f076337>

5.3.1 Procesamiento y Ordenamiento de Paradas

Una vez obtenidas las paradas desde OpenStreetMap, fue necesario organizarlas en función del recorrido de cada línea. Para ello, se compararon los datos de las paradas con los puntos del trayecto extraídos del archivo de la municipalidad, verificando su proximidad y su orden dentro de cada recorrido. Es decir, que se comparó punto a punto entre las paradas y los segmentos de los recorridos para determinar cuál era el orden correcto de las mismas. Esta tarea fue fundamental para que el algoritmo de búsqueda de recorridos funcionara correctamente y pudiera identificar de manera precisa las paradas más cercanas en cada trayecto.

Por otro lado, dado que la mayoría de las paradas fuera del centro no estaban registradas en OpenStreetMap, fue necesario completar manualmente las coordenadas faltantes. Para ello, se analizaron los recorridos desde el StreetView de Google Maps para poder ubicar las paradas que no estaban cargadas, resultando ser un trabajo muy minucioso y detallado con el fin de brindar precisión a los usuarios sin importar el lugar del recorrido ni la línea. De esta manera, se logró obtener un conjunto de paradas más completo y representativo de la realidad del servicio de transporte urbano, duplicando e incluso triplicando la cantidad de paradas dependiendo la línea particular.

5.3.2 Corrección Manual de Superposiciones en los Recorridos

Un problema identificado en la visualización de los recorridos fue la presencia de tramos superpuestos en calles y avenidas por las que los colectivos circulan en ambos sentidos. Esto generaba confusión en la representación de los trayectos, ya que el algoritmo de mapeo no podía diferenciar con precisión a qué parte del recorrido correspondía cada segmento.

Para solucionar esta superposición, se realizaron ajustes manuales en la ubicación de los puntos de los recorridos. Se desplazaron mínimamente algunos segmentos de trayecto para diferenciarlos en la visualización y se acercaron a las paradas correspondientes en esa parte del recorrido. Por ejemplo, en la Figura 30 se puede ver un tramo del recorrido de la línea 503 por la Av. Avellaneda, el cual se encontraba para el trayecto de ida y vuelta pasando por el mismo lugar y luego de ser corregido se puede ver la diferencia de los segmentos y su cercanía a las paradas 11 y 69 respectivamente (arriba a la derecha de la imagen).

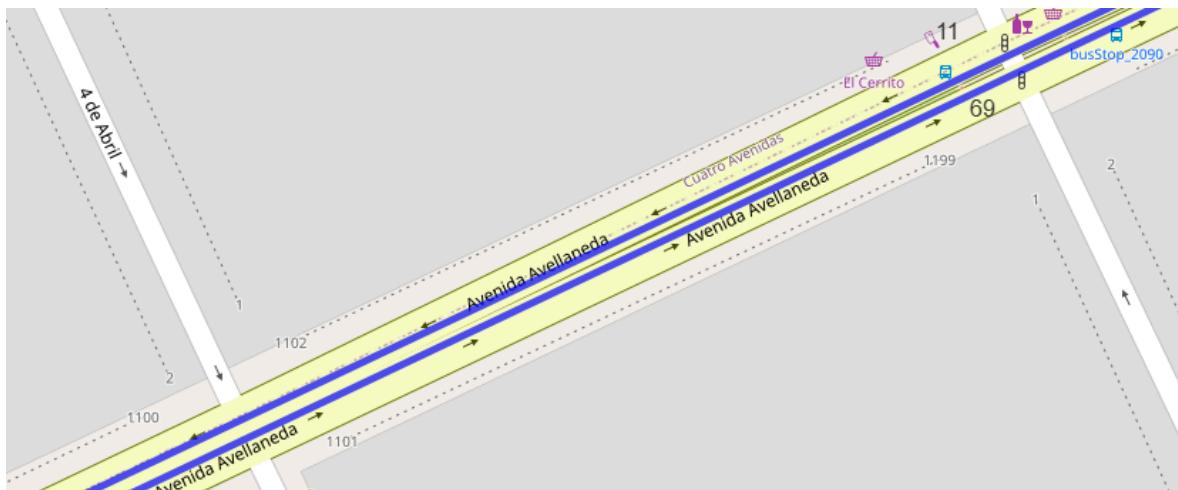


Figura 30 Tramo del Recorrido de la Línea 503 por la Av. Avellaneda.

Este proceso permitió mejorar la claridad del mapa sin alterar significativamente la estructura original de los recorridos. Cabe aclarar que en contraparte con las paradas que hay aproximadamente una cada dos cuadras, los puntos que forman el recorrido son muchos más, habiendo varios en el trayecto de una cuadra (líneas como la 503 poseen más de mil puntos), lo que implica que moverlos manualmente es un trabajo muy engorroso. No obstante era muy importante hacerlo para el correcto funcionamiento de la aplicación.

5.3.3 Almacenamiento en la Base de Datos

Una vez completado el procesamiento de paradas y recorridos, se realizó una verificación final para asegurar que no hubiera errores en la información. Se validó que las paradas estuvieran correctamente ordenadas y asociadas a sus respectivos recorridos, y que los ajustes realizados en las superposiciones garantizaran una representación clara y precisa en el mapa. Para ello se generaron mapas html para cada línea con los recorridos en azul y las paradas numeradas. Los mismos se pueden observar en las Figuras 31 a 36 presentes a continuación.

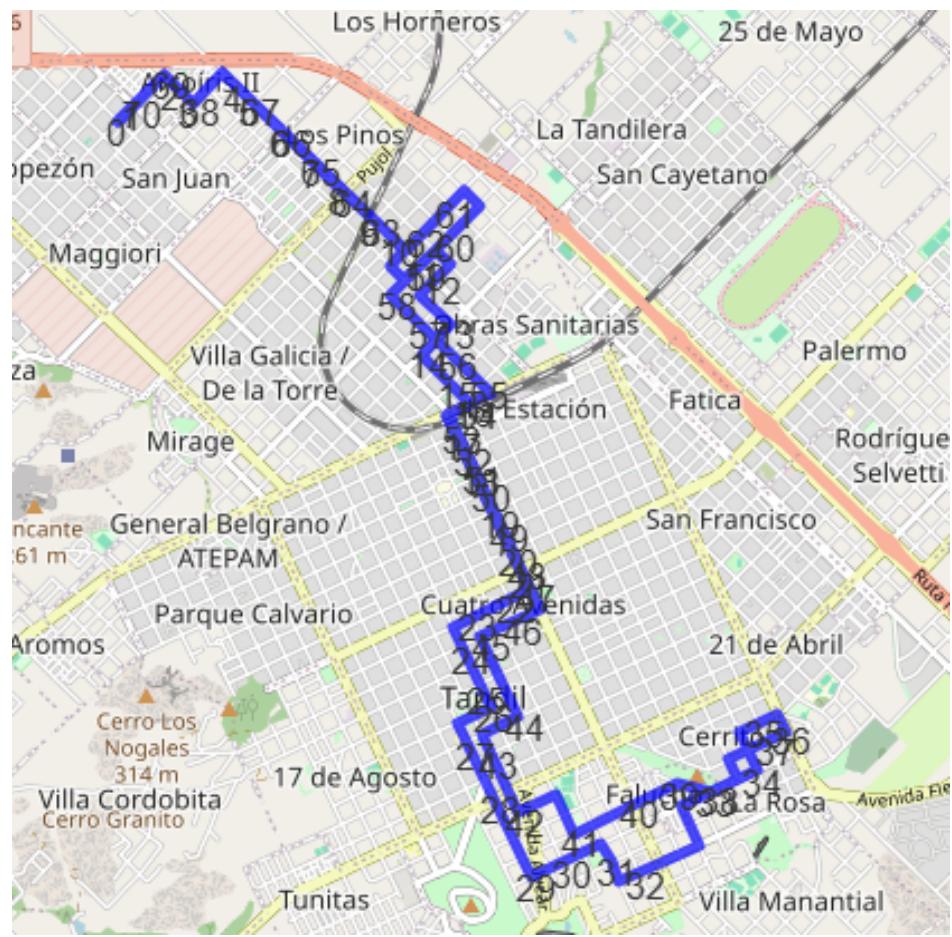


Figura 31 Mapa Generado con el Recorrido y las Paradas de la Línea 500.



Figura 32 Mapa Generado con el Recorrido y las Paradas de la Línea 501.

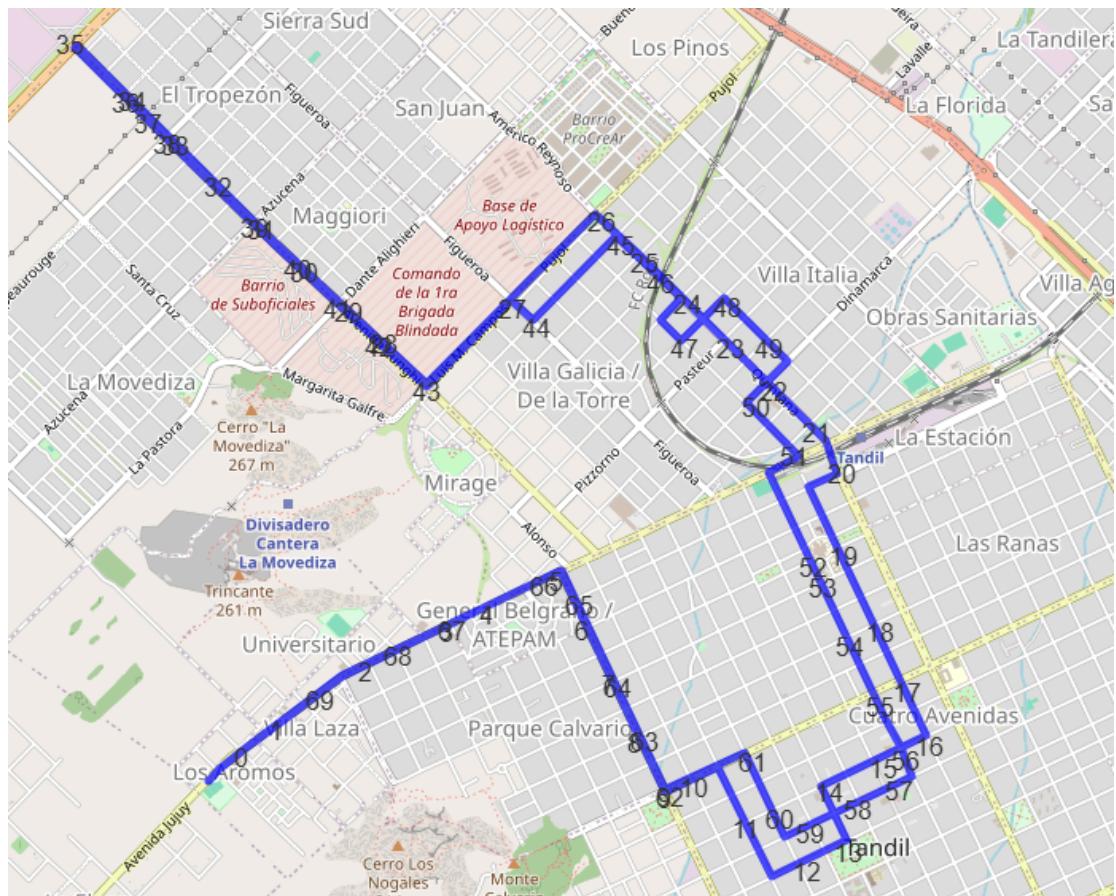


Figura 33 Mapa Generado con el Recorrido y las Paradas de la Línea 502.

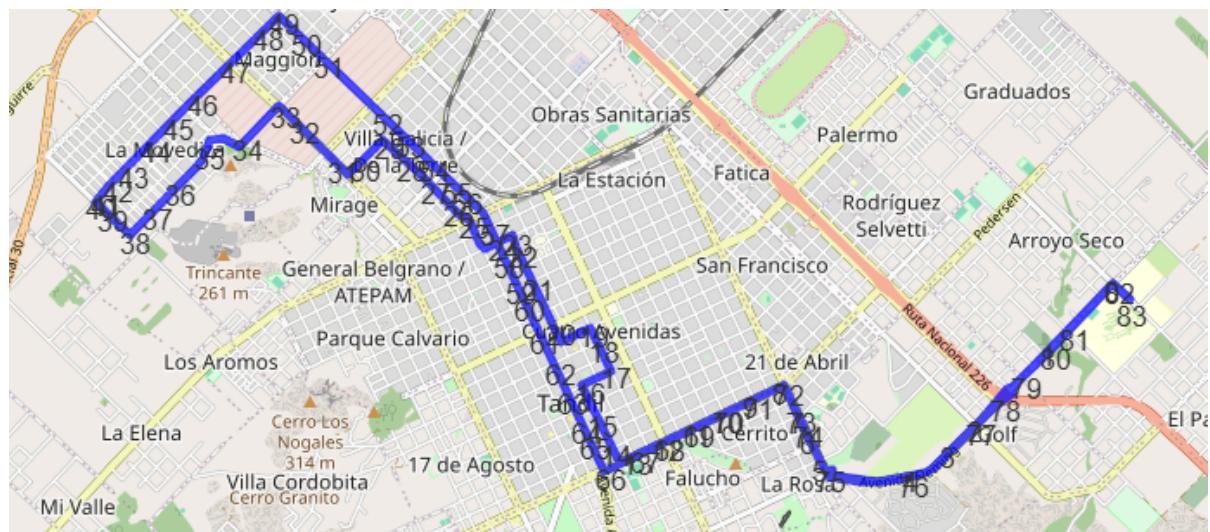


Figura 34 Mapa Generado con el Recorrido y las Paradas de la Línea 503.

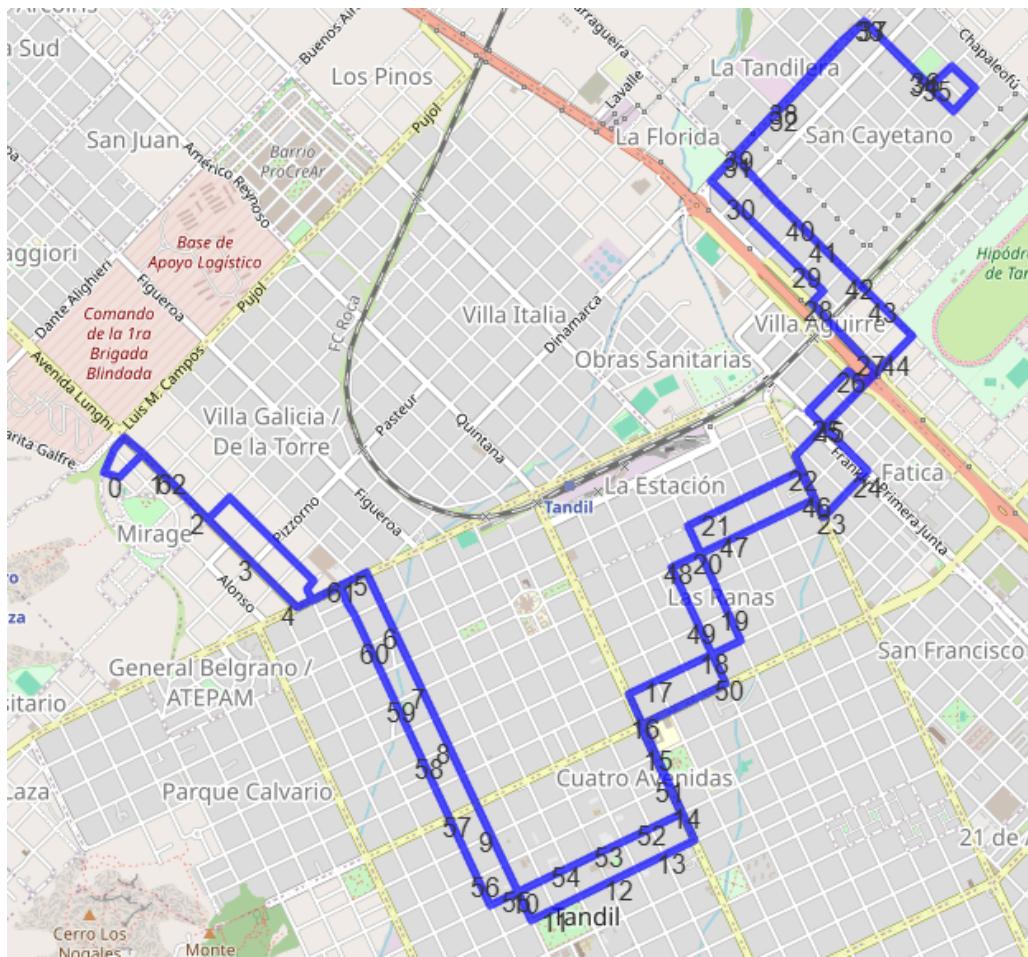


Figura 35 Mapa Generado con el Recorrido y las Paradas de la Línea 504.

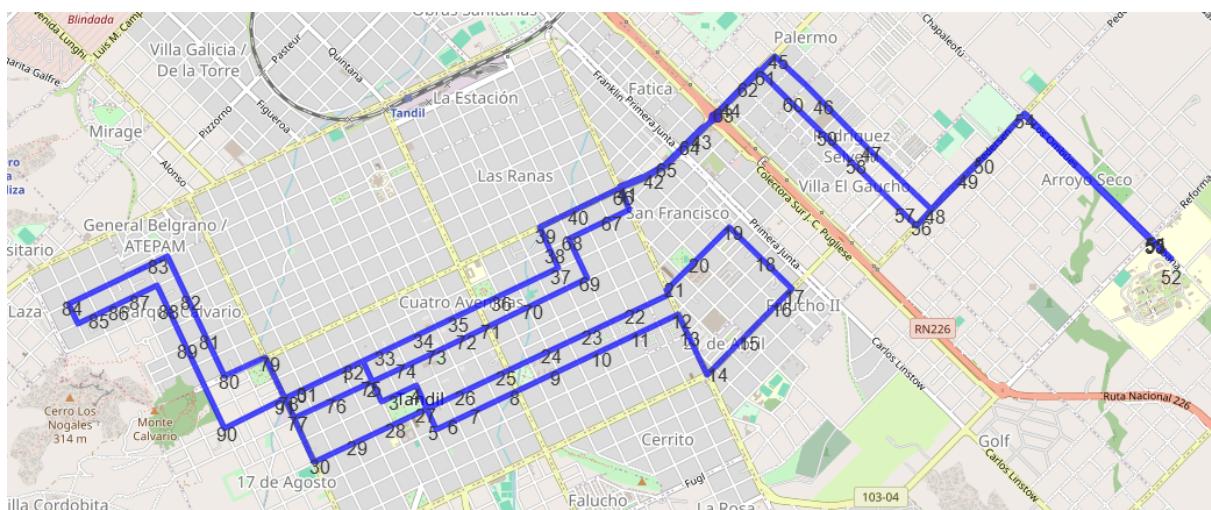


Figura 36 Mapa Generado con el Recorrido y las Paradas de la Línea 505.

Con los datos corregidos y optimizados, se procedió a su almacenamiento en Firebase Realtime Database. Gracias a este proceso, la aplicación cuenta con información confiable y estructurada sobre las paradas y recorridos de los colectivos en Tandil, permitiendo a los usuarios consultar rutas de manera precisa y eficiente.

5.4 Experiencia e Interfaz de Usuario

En el desarrollo de aplicaciones móviles, el diseño de la interfaz de usuario (UI) y la experiencia de usuario (UX) desempeñan un papel fundamental para garantizar que los usuarios puedan interactuar de manera efectiva, intuitiva y satisfactoria con el sistema. Una aplicación bien diseñada no solo facilita el uso de sus funcionalidades, sino que también mejora la percepción del usuario, aumentando su confianza y comodidad en la navegación. En este contexto, la planificación y diseño de UX/UI de la presente aplicación fueron aspectos clave para garantizar su éxito y adopción por parte de los usuarios.

El diseño UX/UI es un componente esencial en cualquier desarrollo de software, ya que impacta directamente en la usabilidad, accesibilidad y satisfacción del usuario. Una interfaz bien estructurada y visualmente atractiva permite que la aplicación sea más fácil de entender y utilizar, reduciendo la curva de aprendizaje y fomentando su uso recurrente. En el caso de esta aplicación, cuyo propósito principal es proporcionar información en tiempo real sobre el transporte público, el diseño debía enfocarse en la claridad y eficiencia para asegurar que los usuarios puedan acceder a la información necesaria con rapidez y precisión.

Desde una perspectiva funcional, una buena UX/UI también influye en la retención de usuarios y en la percepción de la marca o servicio. Aplicaciones con interfaces confusas, sobrecargadas o difíciles de navegar tienden a generar frustración y abandono por parte de los usuarios. En cambio, aquellas que ofrecen una navegación fluida, interacciones intuitivas y un diseño visual agradable logran mejorar la experiencia general, fomentando la fidelización y el uso continuo.

5.4.1 Diseño de la Experiencia de Usuario (UX)

La experiencia de usuario abarca todos los aspectos relacionados con la interacción del usuario con la aplicación, desde la facilidad para encontrar información hasta la eficiencia en la realización de tareas específicas. Para optimizar la UX de esta aplicación, se implementaron principios de diseño centrado en el usuario, priorizando la accesibilidad, la simplicidad y la efectividad en la navegación.

Uno de los principales objetivos en el diseño de la experiencia de usuario fue garantizar un flujo de navegación intuitivo. La estructura de la aplicación se organizó de manera lógica, permitiendo que los usuarios puedan acceder rápidamente a funciones esenciales como la búsqueda de rutas, la localización de colectivos y la configuración de notificaciones. Se evitó el uso de elementos innecesarios que pudieran generar confusión y se priorizó la simplicidad, asegurando que cada pantalla tenga una función clara y específica.

Otro aspecto fundamental en la UX fue la retroalimentación inmediata en las interacciones. Para mejorar la percepción de fluidez y respuesta del sistema, se implementaron animaciones y mensajes de confirmación que indican al usuario cuando una acción ha sido realizada con éxito o si ocurrió algo inesperado.

La accesibilidad también fue un criterio clave en este diseño. Se utilizaron tamaños de fuente legibles, contrastes adecuados y una estructura visual que facilita la comprensión de la información.

Para validar la efectividad del diseño UX, se realizaron pruebas de usabilidad con usuarios potenciales⁷. A través de estas pruebas, se identificaron posibles dificultades en la navegación y se realizaron ajustes en la disposición de los elementos y en la organización de la información para optimizar la experiencia.

5.4.2 Diseño de la Interfaz de Usuario (UI)

El diseño de la interfaz de usuario se enfocó en la creación de una estética visual atractiva y funcional, que no solo haga la aplicación más agradable a la vista, sino que también facilite la interacción con sus funciones. La elección de colores, tipografías y elementos gráficos se realizó siguiendo principios de diseño minimalista, evitando la sobrecarga de información y priorizando la claridad en la presentación de los datos.

Se optó por una paleta de colores que refuerza la identidad visual de la aplicación y mejora la diferenciación de los distintos elementos en la pantalla. Los colores principales se utilizaron para destacar funciones esenciales, mientras que los secundarios aportan jerarquía visual sin generar distracciones. De esta manera, se logra que la información más relevante, como los tiempos de llegada de los colectivos o las alertas de notificación, resalte de manera natural y sea fácilmente identificable por el usuario. La idea principal fue que los colores principales no estuvieran asociados a las líneas de colectivo para que cualquier elemento de estas sea fácil de diferenciar.

En cuanto a la tipografía, se seleccionaron fuentes de fácil lectura y con una jerarquía clara para mejorar la comprensión de los contenidos. Se emplearon tamaños adecuados para cada tipo de información, asegurando que los textos sean legibles incluso en dispositivos con pantallas pequeñas.

Los iconos jugaron un papel crucial en la UI, ya que permiten transmitir información de manera visual sin necesidad de depender completamente del texto. Se utilizaron iconos intuitivos para representar acciones como búsqueda, retroceder y selección de opciones, y se incluyeron animaciones sutiles en los botones e iconos para proporcionar una experiencia visual más dinámica y atractiva.

⁷ Familia y Amigos.

5.4.3 Interfaces de la Aplicación

La aplicación cuenta con diversas pantallas diseñadas para cubrir las necesidades del usuario de manera eficiente. Las mismas, presentadas en la Figura 22, fueron diseñadas teniendo en cuenta todos los aspectos anteriormente mencionados en los diseños de la experiencia e interfaz de usuario, buscando agilizar y facilitar la adopción de la aplicación por parte de los mismos.

En primer lugar, la app cuenta con la *Interfaz Principal*, la cual es el punto de acceso a la misma, donde los usuarios pueden seleccionar líneas de colectivo, buscar destinos y visualizar recorridos en el mapa. Se priorizó un diseño limpio y organizado para evitar la sobrecarga de información.

Luego, la *Interfaz de Búsqueda*, inspirada en servicios como Google Maps, permite encontrar ubicaciones de manera eficiente, mostrando sugerencias en tiempo real y permitiendo seleccionar destinos previamente guardados o recientemente buscados para facilitar el acceso rápido a rutas habituales.

La *Interfaz de Elección* de Rutas presenta distintas opciones de recorrido entre un punto de origen y destino, permitiendo comparar tiempos de viaje y seleccionar la alternativa más conveniente. Se incorporaron elementos visuales que resaltan las diferencias entre cada opción para facilitar la toma de decisiones.

Por otro lado, la *Interfaz de Seguimiento* muestra información detallada sobre el trayecto de un colectivo en tiempo real, incluyendo tiempos estimados de llegada y opciones para recibir notificaciones sobre cambios en la ruta. Se diseñó con un enfoque en la claridad visual para garantizar que el usuario pueda interpretar la información de manera rápida y precisa, siguiendo un estilo similar a Moovit.

En cuanto a la *Interfaz de Configuración*, permite a los usuarios personalizar aspectos de la aplicación, como iniciar sesión, cerrarla o la gestión de lugares favoritos. Se diseñó con un enfoque simplificado para garantizar que los ajustes puedan realizarse de manera rápida y sin complicaciones. Esta interfaz no hace referencia a una pantalla particular, sino a las ventanas que se muestran al seleccionar las opciones del menú desplegable de la interfaz principal.

Por último se encuentra la *Interfaz de Predicción*, la misma es accesible luego de seleccionar un colectivo particular en la pantalla principal para ver cuánto le falta para llegar a una determinada parada. Allí permite seleccionar la parada objetivo seleccionandola en el mapa o en una lista de direcciones, para posteriormente calcular el tiempo de demora del colectivo y mostrarlo en el mapa.

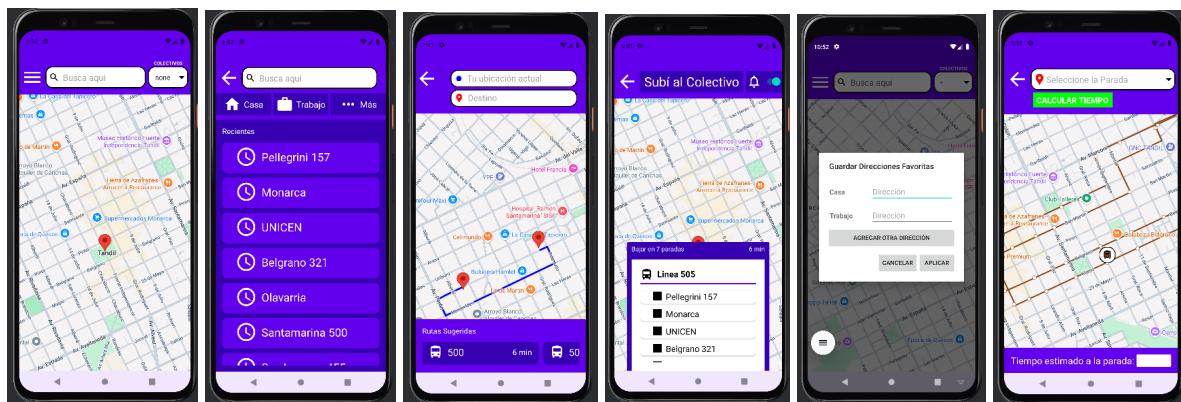


Figura 37 Interfaces de la Aplicación Móvil.

5.5 Implementación de la Aplicación

El desarrollo de una aplicación móvil implica la integración de múltiples componentes que deben interactuar de manera eficiente para brindar una experiencia óptima al usuario. Desde la arquitectura general del sistema hasta la implementación detallada de cada módulo, es fundamental contar con una estructura bien definida que facilite la escalabilidad, el mantenimiento y el correcto funcionamiento del sistema.

Este apartado describe el desarrollo de la aplicación BusNow, abarcando tanto la arquitectura del sistema como la implementación del servidor y del cliente. Implementar la aplicación consistió en llevar a cabo los requerimientos funcionales y no funcionales presentados en las secciones anteriores, para de esta manera ofrecerle al usuario final una experiencia completa al momento de utilizarla.

5.5.1 Arquitectura de la Aplicación

Durante el proceso de desarrollo de cualquier sistema resulta fundamental elegir una arquitectura ya que define cómo se organizan sus componentes, cómo interactúan entre sí y cómo se gestionan los datos y la lógica de negocio. La arquitectura establece la base sobre la cual se construirá el sistema, determinando su eficiencia, escalabilidad, mantenibilidad y seguridad.

Una de las razones principales para definir una arquitectura es la necesidad de estructurar la aplicación de manera que sus diferentes partes puedan funcionar de manera eficiente y coordinada. Sin una arquitectura bien definida, el desarrollo se volvería caótico, con código difícil de mantener y con una alta probabilidad de generar inconsistencias en el manejo de datos. La arquitectura proporciona un marco de referencia claro, facilitando la distribución de responsabilidades y evitando redundancias o conflictos en la implementación del sistema.

Otro aspecto clave es la escalabilidad. Al diseñar una aplicación, es importante prever cómo crecerá en el futuro y cómo manejará un aumento en la cantidad de usuarios o en la cantidad de datos procesados. Una arquitectura bien pensada permite que la aplicación pueda expandirse sin necesidad de realizar cambios drásticos en su estructura. Por ejemplo, en una aplicación que depende de información en tiempo real, como el seguimiento de colectivos, la arquitectura debe permitir la incorporación de nuevos usuarios sin que esto afecte el rendimiento general del sistema.

La elección de una arquitectura también está directamente relacionada con el rendimiento y la optimización de recursos. En una aplicación móvil, los dispositivos de los usuarios tienen limitaciones en cuanto a procesamiento, memoria y consumo de batería. Si la aplicación estuviera diseñada sin una arquitectura adecuada, podría sobrecargar el dispositivo del usuario, generando una experiencia lenta o ineficiente.

Además, la seguridad es un aspecto fundamental que depende en gran medida de la arquitectura elegida. En una aplicación que maneja datos es crucial proteger la información y evitar vulnerabilidades. Una arquitectura bien diseñada define claramente qué componentes tienen acceso a qué datos, cómo se gestionan los permisos y cómo se protege la comunicación.

Por todo esto, se optó por una arquitectura cliente-servidor lo que responde a varias razones fundamentales que garantizan su correcto funcionamiento, eficiencia y escalabilidad. En este modelo, la aplicación móvil actúa como el cliente, proporcionando la interfaz gráfica y permitiendo la interacción con el usuario, mientras que el servidor se encarga de procesar la información, realizar cálculos complejos y administrar la base de datos en Firebase Realtime Database. Este enfoque permite descentralizar las operaciones más costosas en términos de procesamiento y mejorar la seguridad de los datos, evitando la manipulación directa de la base de datos desde el cliente.

La estructura cliente-servidor facilita la escalabilidad y el mantenimiento del sistema. Al centralizar la lógica de negocio en el servidor, cualquier actualización o mejora en los algoritmos de predicción, en la forma en que se almacenan los datos o en la integración con nuevas fuentes de información se puede realizar sin necesidad de modificar y redistribuir la aplicación móvil a todos los usuarios.

El flujo de comunicación se da a través de solicitudes HTTP, en las cuales la aplicación envía peticiones al servidor para consultar información sobre colectivos, registrar eventos del usuario y obtener predicciones. Si bien la mayor parte de las interacciones con la base de datos pasan por el servidor, la app mantiene un listener activo en Firebase para actualizar en tiempo real la ubicación de los colectivos, evitando demoras en la frecuencia de actualización. Aunque pueda parecer que esta decisión rompe el encapsulamiento de la base de datos, se optó por utilizar esta

metodología ya que únicamente se está accediendo a las ubicaciones de los colectivos y no a datos más sensibles o cargando ningún tipo de información.

5.5.2 Desarrollo del Servidor

El servidor de la aplicación está desarrollado utilizando FastAPI, un framework moderno para la construcción de APIs en Python que permite manejar solicitudes HTTP de manera eficiente. Este servidor se encarga de recibir y procesar datos enviados por la aplicación móvil y almacenar información en una base de datos en Firebase. Además, incorpora modelos de aprendizaje profundo para realizar predicciones de tiempos de llegada de los colectivos, utilizando modelos LSTM previamente entrenados.

Para implementar el servidor, se han importado diversas bibliotecas que permiten gestionar la API, la base de datos, el procesamiento de datos y la predicción de rutas. FastAPI es la base sobre la cual se estructura el servicio web, proporcionando endpoints para la comunicación con la aplicación móvil. TensorFlow es utilizado para cargar los modelos LSTM entrenados, encargados de predecir la ubicación futura de los colectivos en base a datos históricos. La biblioteca Pandas facilita la manipulación y transformación de los datos, mientras que NumPy se usa para manejar arreglos numéricos en los cálculos del modelo. Además, la biblioteca Joblib se emplea para cargar los objetos scaler que normalizan y desnormalizan los datos antes y después de pasar por los modelos.

Como ya se mencionó previamente, la base de datos utilizada en el proyecto es Firebase, un servicio en la nube que permite el almacenamiento y sincronización de datos en tiempo real. La integración con Firebase se realiza mediante Firebase Admin, que permite autenticar el acceso a la base de datos y manipular datos almacenados en la estructura NoSQL de Firestore. Para inicializar la conexión con Firebase, se carga un archivo de credenciales en formato JSON que contiene las claves de acceso al proyecto. Dentro de la base de datos, se almacenan tanto las ubicaciones de los colectivos como los recorridos de cada línea y la información personal de cada usuario, permitiendo acceder a todos estos datos en tiempo real.

El endpoint más importante del servidor es '/predict', que recibe solicitudes de predicción de tiempos de llegada a paradas específicas. Para ello, la aplicación móvil envía información sobre la línea de colectivo, un identificador del colectivo, la ubicación de la parada y tres vectores adicionales que representan información contextual de la ruta. Una vez recibida la solicitud, el servidor selecciona el modelo LSTM correspondiente a la línea de colectivo en cuestión y obtiene las últimas diez ubicaciones registradas del colectivo desde Firebase. Si hay suficientes datos históricos, estos se normalizan utilizando el scaler correspondiente, se procesan a través del modelo LSTM y se obtiene una predicción de la ubicación futura del colectivo. Las predicciones se hacen hasta llegar a la parada objetivo que fue

pasada como parámetro de la petición, pero como las estimaciones de posición no tienen porque caer encima de una parada se ideó un método que agote todas las posibilidades hasta encontrarla. Para afirmar que la predicción cayó cerca de una parada el servidor predice 60 veces (equivale a 60 minutos) y chequea que la predicción caiga en un radio de la parada objetivo. Este radio aumenta si luego de las 60 predicciones no se encontró, llegando a repetirse este incremento hasta 2 veces. Sin embargo el chequeo del radio no es suficiente porque en paradas enfrentadas en una avenida por ejemplo el colectivo puede pasar por enfrente de la parada y faltarle toda la vuelta al recorrido por lo que aquí es donde juegan su papel los vectores de movimiento. Particularmente se compara el vector de desplazamiento de la predicción dado por la posición de las últimas dos predicciones contra tres vectores, el que está la parada, el anterior y el posterior. El porqué de la comparación con tres vectores se debe a que en zonas de zigzagueo del recorrido puede que la predicción no tenga la dirección del que se encuentra la parada sino que es más similar al anterior o siguiente, y así se agotan todas las posibilidades. Finalmente, con un máximo de 180 predicciones (60 por cada radio) el resultado se desnormaliza y se devuelve a la aplicación móvil en forma de coordenadas geográficas y tiempo estimado de llegada. Cabe aclarar que si antes de las 180 predicciones encontró la parada objetivo corta, pero sino el tiempo de vuelta es -1 para poder informar que la parada se encuentra muy lejos para ser predecida.

Otro endpoint relevante es '/get_recorridos', el cual proporciona información sobre los recorridos de los colectivos almacenados en la base de datos. Cuando se recibe una solicitud a este endpoint, el servidor accede a la referencia 'recorridos' en Firebase y extrae la información de cada línea, incluyendo su número, color representativo, las paradas a lo largo de la ruta y los puntos intermedios del recorrido. Los datos se estructuran en formato JSON y se devuelven como respuesta a la aplicación móvil. Además, dentro de este método se incluye una llamada en segundo plano que, si está activada, limpia automáticamente los registros de colectivos inactivos en la base de datos, asegurando que los datos almacenados sean siempre relevantes.

Otro de los métodos fundamentales del servidor es '/update_bus_location', el cual se encarga de registrar la ubicación de los colectivos en Firebase en tiempo real. Cuando un colectivo reporta su ubicación, el servidor evalúa si existe otro colectivo cercano en la misma línea. Para ello, compara la distancia entre la nueva ubicación reportada y las ubicaciones de colectivos ya registrados en la base de datos. Si hay un colectivo a menos de 100 metros, se asume que es el mismo vehículo y se actualiza su ubicación en Firebase. Si no hay colectivos cercanos, se crea un nuevo identificador para el vehículo y se almacena su información en la base de datos. Este proceso garantiza que la información sobre la ubicación de los colectivos sea precisa y se actualice en tiempo real para proporcionar estimaciones más exactas a los usuarios de la aplicación.

El servidor también gestiona un registro de las últimas diez ubicaciones de cada colectivo mediante el método 'update_last10'. Esta función mantiene una ventana móvil de datos históricos en Firebase, permitiendo que el modelo LSTM tenga acceso a la información necesaria para realizar predicciones. Cada vez que se actualiza la ubicación de un colectivo, su historial de posiciones se actualiza, eliminando el dato más antiguo si ya hay diez registros previos. Esto permite que el modelo LSTM tenga un conjunto de datos consistente para realizar inferencias de tiempo de llegada. Adicionalmente, el resto de endpoints se encargan de guardar y obtener los lugares favoritos y las búsquedas recientes de cada uno de los usuarios registrados en la base de datos.

Para utilizar el servidor, es necesario levantarla utilizando FastAPI, lo cual se puede hacer ejecutando el script correspondiente en Python. Una vez iniciado, el servidor comienza a escuchar peticiones en los endpoints definidos y responde de acuerdo con la información almacenada en Firebase y los cálculos realizados con los modelos de predicción. Para la comunicación entre la aplicación móvil y el servidor, se emplea la biblioteca Volley en el lado del cliente, la cual permite realizar solicitudes HTTP de manera eficiente y manejar respuestas asíncronas en Android. Cuando la aplicación móvil necesita predecir el tiempo de llegada de un colectivo, envía una solicitud HTTP POST a '/predict', incluyendo los parámetros necesarios en formato JSON. Luego, el servidor procesa la solicitud, ejecuta el modelo correspondiente y devuelve una respuesta con la predicción calculada.

5.5.3 Organización del Código

Para facilitar la comprensión del código desarrollado en Android Studio se describirá la organización del proyecto, detallando la estructura de archivos y carpetas, así como la distribución de las clases y componentes principales. Esto permitirá comprender mejor la relación entre los distintos módulos y facilitará futuras modificaciones o mejoras en el código. El mismo está estructurado en cinco Activities, cada una de ellas asociada a una interfaz gráfica específica.

En Android, una Activity es una clase que representa una pantalla de la aplicación con la que el usuario puede interactuar. Cada Activity tiene un archivo XML asociado que define la interfaz gráfica de esa pantalla. La relación entre ambos se establece en el método `onCreate()` de la Activity. El XML define la apariencia de la interfaz con elementos como botones, textos e imágenes, mientras que la Activity maneja la lógica y las interacciones del usuario.

Además, se han desarrollado varias clases auxiliares para encapsular funcionalidades comunes a varias actividades, manejar datos y mejorar la modularización. En la Figura 38 se puede observar todo lo mencionado previamente.

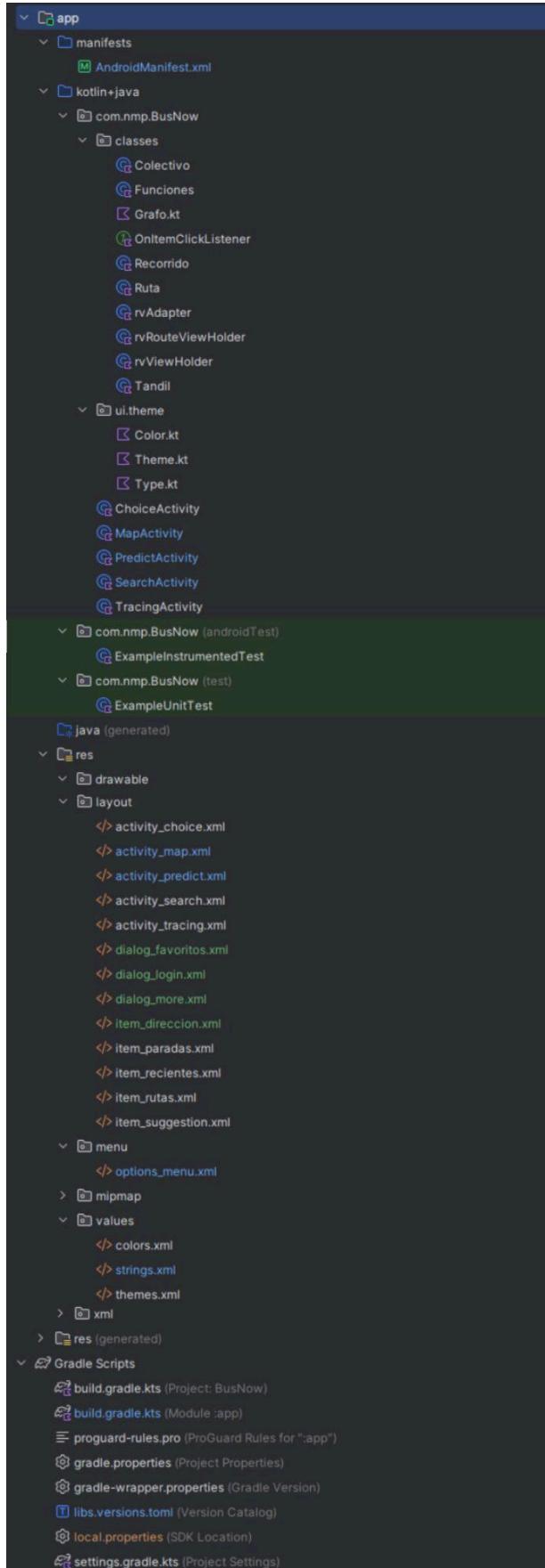


Figura 38 Organización del Código.

Esta organización brinda distintos aspectos positivos a la hora de entender y gestionar el código. En primer lugar, la modularidad permite que cada parte de la aplicación esté separada, lo que facilita la depuración y futuras mejoras. También esta arquitectura permite agregar nuevas funcionalidades sin modificar demasiado el código existente y reutilizar código, lo que evita redundancias y mejora el mantenimiento. Otro aspecto fundamental es la claridad, resultando que el código sea más comprensible.

5.5.4 Desarrollo del Cliente

En la arquitectura de la aplicación, el cliente es la parte del sistema que interactúa directamente con el usuario, permitiéndole acceder a las funcionalidades de la aplicación y enviar o recibir información desde el servidor o desde otras fuentes de datos. En este caso, el cliente corresponde a la aplicación móvil que los usuarios instalan en sus dispositivos Android y que actúa como la interfaz principal a través de la cual pueden realizar todas las acciones disponibles en el sistema.

El cliente tiene diversas funcionalidades diseñadas para ofrecer una experiencia eficiente e intuitiva. En primer lugar, la aplicación permite a los usuarios registrarse e iniciar sesión, brindando así acceso a configuraciones personalizadas y la posibilidad de guardar información relevante, como ubicaciones de interés. Para gestionar la autenticación de los usuarios, se ha utilizado Firebase Authentication, lo que facilita la administración de sesiones y credenciales de manera segura.

El sistema de autenticación se integra en la interfaz de la aplicación a través de un menú emergente, que ofrece opciones dinámicas dependiendo del estado de sesión del usuario. Si un usuario ya ha iniciado sesión, el menú muestra la opción de cerrar sesión, mientras que si no hay una sesión activa, se ofrece la posibilidad de iniciar sesión o registrarse. Esto se logra mediante la verificación del estado actual del usuario, modificando dinámicamente el texto de la opción en el menú desplegable.

Cuando un usuario desea iniciar sesión, se despliega un cuadro de diálogo donde se solicita su correo electrónico y contraseña. Si estos datos son válidos y coinciden con un usuario registrado en Firebase, la sesión se inicia correctamente, mostrando un mensaje de confirmación. En caso contrario, se notifica el error correspondiente. De manera similar, si el usuario opta por registrarse, el sistema intenta crear una nueva cuenta con los datos proporcionados y, si el proceso es exitoso, el usuario queda autenticado de inmediato.

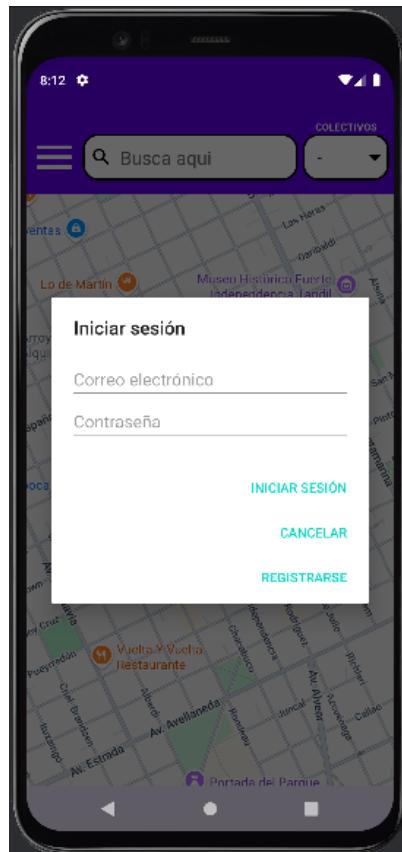


Figura 39 Cuadro de Inicio de Sesión o Registro.

Además del inicio de sesión, el sistema permite a los usuarios guardar direcciones frecuentes, como "Casa" o "Trabajo", así como agregar otras ubicaciones personalizadas. Para ello, se implementó otro cuadro de diálogo en el que el usuario puede ingresar y administrar sus direcciones favoritas. Al abrir el diálogo, la aplicación realiza una solicitud al servidor para recuperar las direcciones previamente guardadas, que se muestran en los campos correspondientes. Si el usuario desea agregar nuevas direcciones, puede hacerlo dinámicamente mediante la interfaz, ingresando tanto el nombre como la dirección.

Cuando el usuario guarda las direcciones, la aplicación valida los datos ingresados y obtiene las coordenadas geográficas de cada dirección antes de enviarlas al servidor. Si la conversión a coordenadas no es exitosa, se muestra un mensaje de error, indicando que la dirección ingresada no es válida. En caso de que todas las direcciones sean correctas, se envían al servidor para ser almacenadas y posteriormente utilizadas en el cálculo de rutas y tiempos de viaje dentro de la aplicación.



Figura 40 Cuadro de Edición de Direcciones Favoritas.

El sistema también incorpora medidas de seguridad, como la necesidad de autenticación para acceder a las direcciones favoritas. Si un usuario intenta acceder a esta funcionalidad sin haber iniciado sesión, se muestra un mensaje indicando que debe autenticarse primero. De este modo, la aplicación garantiza que cada usuario solo pueda gestionar y visualizar sus propias configuraciones personalizadas.

En cuanto a la funcionalidad que permite visualizar en un mapa la ubicación actual de los colectivos en tiempo real basado en los reportes de los usuarios que informan cuándo y dónde abordaron una unidad, la aplicación combina múltiples elementos: la selección de una línea de colectivo, la obtención y representación de su recorrido, la recepción de actualizaciones de ubicación en tiempo real y la interacción del usuario con el mapa.

Cuando un usuario selecciona una línea de colectivo en el Spinner, el sistema borra cualquier información previa en el mapa, asegurándose de que la visualización se actualice correctamente. Luego, se extrae el identificador de la línea seleccionada y se busca su información en la estructura de datos que almacena la información de todas las líneas. Esta se obtiene a través de una solicitud HTTP al servidor cuya respuesta contiene una lista de recorridos con las coordenadas de los puntos que componen el trayecto, así como las paradas intermedias. A partir de estos datos, se dibuja en el mapa el recorrido del colectivo usando una Polyline, y se marcan las paradas con pequeños círculos blancos que indican sus ubicaciones.

A continuación, se invoca la función trackBusLocations(), que es la encargada de obtener en tiempo real las ubicaciones de los colectivos de la línea seleccionada. Para ello, se establece un ValueEventListenner en la base de datos en la referencia de la línea particular, donde se almacenan las coordenadas reportadas por los usuarios. Cuando un colectivo reporta una nueva ubicación, el listener detecta el cambio y actualiza el marcador correspondiente en el mapa. Si el colectivo ya tiene un marcador, simplemente se mueve a la nueva posición; si no existe, se crea un nuevo marcador con un ícono distintivo basado en el color de la línea. Este es el único acceso que se hace directamente a la base de datos desde la aplicación y sus motivos ya fueron explicados anteriormente en la sección 5.2 Decisiones Tecnológicas.

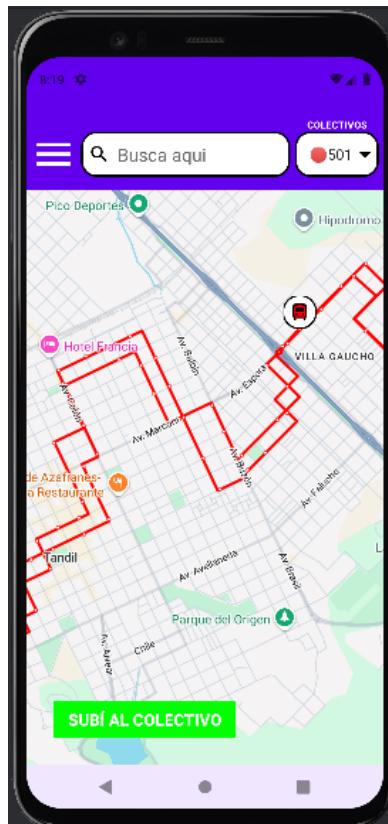


Figura 41 Visualización de Rutas y Ubicación de Colectivos.

Los usuarios pueden contribuir con estos reportes activando el modo de seguimiento mediante el botón btnSeguimiento. Al presionar este botón, si no hay un seguimiento activo, se inicia el rastreo de ubicación del usuario, lo que permite actualizar la base de datos con la información de que ha subido a un colectivo. Cuando el usuario decide finalizar el seguimiento, la aplicación detiene la actualización de su ubicación y elimina el reporte del colectivo en el que viajaba. La actualización es válida únicamente si el usuario se encuentra en una ubicación cercana al recorrido de la línea que desea informar. En caso de validarse hace el llamado al servidor, el cual busca un colectivo cercano para actualizar sus coordenadas o bien crea una unidad nueva en la base de datos.

Finalmente, el mapa se mantiene interactivo, permitiendo a los usuarios tocar los marcadores para obtener más detalles sobre la línea y la unidad específica que reportó su ubicación. Este enfoque descentralizado, basado en los reportes de los pasajeros, permite ofrecer información en tiempo real sin necesidad de depender de dispositivos GPS instalados en los colectivos, aprovechando la colaboración de la comunidad de usuarios.

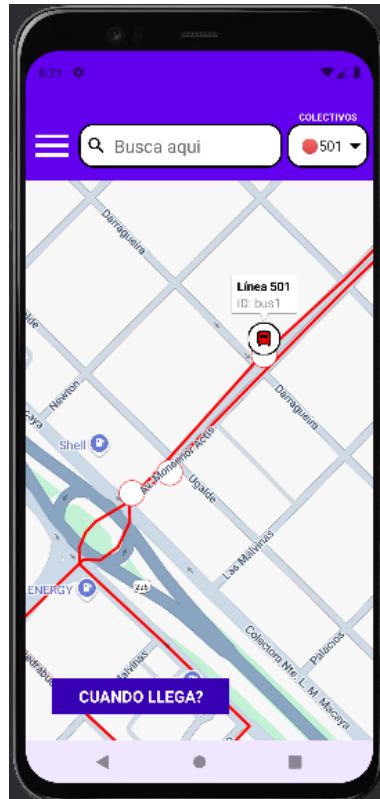


Figura 42 Visualización de Información Adicional al Apretar una Unidad Específica.

La funcionalidad de búsqueda de destino permite a los usuarios seleccionar un destino a partir de opciones recientes, lugares favoritos, ingresando la dirección manualmente o seleccionando un punto en el mapa, facilitando así la planificación de sus trayectos. Para acceder a las primeras tres funciones, el usuario puede interactuar con el botón de búsqueda, que activa un intent para abrir la actividad SearchActivity. Dentro de esta actividad, el campo de búsqueda se inicializa automáticamente con el teclado desplegado para mejorar la experiencia del usuario. Además, se implementa la integración con Google Places para ofrecer sugerencias de direcciones en tiempo real a medida que el usuario escribe. También se cuenta con accesos directos a destinos frecuentes, como "Casa" y "Trabajo", almacenados previamente por el usuario, y un botón adicional para visualizar más ubicaciones favoritas. La lista de búsquedas recientes se obtiene a través de una consulta al servidor, donde se almacenan las direcciones previamente ingresadas junto con sus coordenadas. Cuando el usuario selecciona un destino de la lista de recientes o de las sugerencias, se obtiene su latitud y longitud utilizando la función

Funciones.getLatLangSuggestion(), y se almacena en la base de datos mediante una petición HTTP. Posteriormente, se inicia nuevamente la actividad MapActivity, enviando la información del destino seleccionado a través de un intent para que el usuario pueda visualizar el lugar seleccionado en el mapa y posteriormente las rutas. Además, se implementa una lógica que permite ocultar el teclado automáticamente cuando el usuario toca fuera del campo de búsqueda, mejorando la usabilidad de la aplicación. Así, esta funcionalidad optimiza la navegación del usuario al proporcionar múltiples opciones de búsqueda y acceso rápido a destinos frecuentes, asegurando una experiencia fluida e intuitiva.



Figura 43 Recomendaciones en Tiempo Real.

Cuando el usuario aprieta el botón de indicaciones luego de seleccionado un destino, se verifica si las coordenadas de origen (en un principio la ubicación actual del usuario) y destino han sido correctamente inicializadas. Si es así, se abre ChoiceActivity, una nueva pantalla donde se desplegarán las opciones de ruta disponibles. Para ello, se pasan las coordenadas y direcciones como parámetros a través de un Intent. En esta pantalla ambas locaciones pueden ser modificadas nuevamente siguiendo la lógica de la interfaz de búsqueda, el usuario va escribiendo y le salen recomendaciones en tiempo real, una vez seleccionada la dirección se llama a Funciones.getLatLangSuggestion() para obtener las coordenadas.

Al recibir los datos del origen y destino, se invoca la función traceAllRoutes, que se encarga de generar todas las rutas posibles. Para esto, se consideran tanto los trayectos caminando como los de las diferentes líneas de colectivos disponibles. Para la primera opción se consulta la API de Google Maps para obtener la ruta óptima. A su vez, la aplicación cuenta con un grafo interno que modela las paradas de colectivos y sus conexiones, lo que permite determinar la mejor combinación de líneas de transporte público.

El proceso de selección de ruta busca ofrecer las opciones más eficientes. Para ello, se filtran las rutas considerando la distancia y el tiempo de viaje, eliminando aquellas que no representen una ventaja significativa frente a otras alternativas. Una vez procesadas las rutas, se actualiza la lista rutasList, donde cada elemento representa un recorrido posible con su respectiva duración y distancia. Esta lista se muestra en la interfaz de usuario a través de un RecyclerView, permitiendo al usuario visualizar y seleccionar la mejor opción.

Cuando el usuario elige una ruta, la aplicación dibuja el recorrido en el mapa y permite la confirmación de la selección. Para ello, se utiliza Funciones.drawRoute, que traza el camino en la interfaz. De esta manera, el usuario puede ver en detalle el trayecto recomendado antes de tomar su decisión final. La funcionalidad de selección de rutas, por lo tanto, garantiza una experiencia interactiva y eficiente, brindando información clara y precisa sobre las mejores opciones de transporte disponibles para llegar al destino deseado.

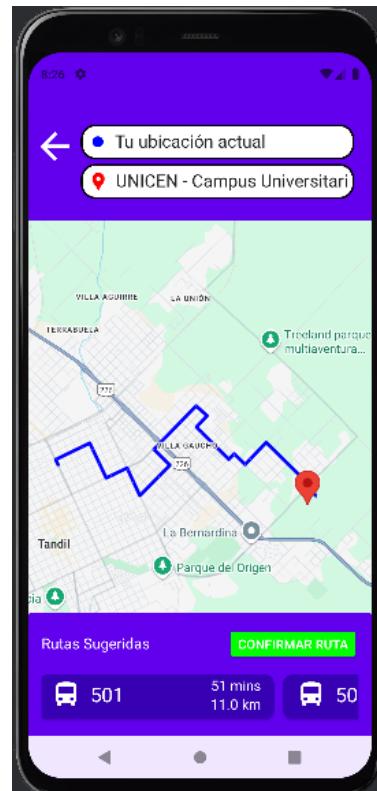


Figura 44 Visualización de Rutas Óptimas por Línea.

Una vez que el usuario ha seleccionado una ruta y confirmado su elección, la aplicación inicia el seguimiento en tiempo real del recorrido del colectivo. Este proceso comienza con la activación de un evento en la interfaz, el cual redirige al usuario a una nueva actividad encargada de visualizar y gestionar el seguimiento de la ruta.

Al llegar a la TracingActivity, se recuperan las coordenadas almacenadas en el intent y se asignan a variables correspondientes. Posteriormente, se inicializan los componentes de la interfaz, incluyendo el mapa donde se visualizará el trayecto, un botón para notificar que el usuario ha subido o bajado del colectivo, el cual funciona de con la misma lógica de la interfaz principal al mostrar los recorridos de los colectivos, y un listado de paradas próximas junto con información relevante como la duración estimada del viaje y la cantidad de paradas restantes hasta el destino. El listado de paradas próximas, el cual se obtiene a partir de la ruta seleccionada y se muestra en un RecyclerView. Cada parada se extrae en formato de dirección legible y se dispone en una lista interactiva. Esto permite que el usuario visualice con claridad las estaciones restantes antes de su destino.

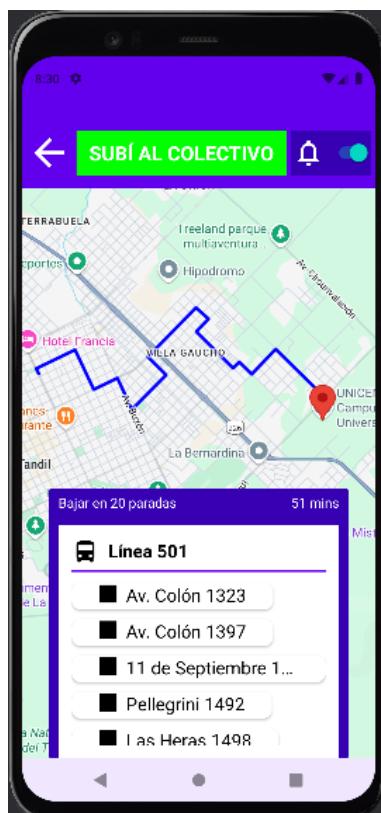


Figura 45 Seguimiento de la Ruta Seleccionada.

Además, se incorporó un interruptor de notificaciones que, aunque en la el MVP de la aplicación no tiene ninguna funcionalidad, está pensado para en versiones futuras permitir al usuario activar o desactivar alertas relacionadas con su viaje, lo que

facilitaría el seguimiento de la llegada a la parada de destino o la recepción de información relevante en tiempo real.

Por último, la funcionalidad de consultar la llegada de un colectivo a una parada específica comienza en la pantalla principal del mapa, donde los colectivos en circulación se representan con marcadores. Cuando un usuario toca un marcador de colectivo en el mapa, el sistema responde mostrando una ventana de información con los detalles de la línea y el botón para predecir su llegada. Este proceso se gestiona en la actividad MapActivity, en particular dentro del método setupMarkerClickListener, que maneja los eventos de clic en los marcadores de los colectivos. Al tocar un marcador, se almacena la información del colectivo y se habilita el botón de predicción. Si el usuario lo presiona, se lanza PredictActivity, enviando la información de la línea y el identificador del colectivo seleccionado.

En PredictActivity, al recibir la información del colectivo, se inicializan los componentes de la interfaz, incluyendo un mapa interactivo y un selector de paradas (Spinner). Se obtiene la lista de paradas de la línea seleccionada y se llena el Spinner con sus direcciones, permitiendo que el usuario elija una parada específica en este o bien puede seleccionarla en el mapa. Cuando el usuario selecciona una parada, se almacena su ubicación y se solicita nuevamente la ubicación actual del colectivo mediante una petición HTTP al servidor para asegurarse que realmente sea la última.

Una vez seleccionada la parada y obtenida la ubicación del colectivo, el usuario puede presionar el botón de cálculo para solicitar la predicción del tiempo de llegada. Al presionar el botón, la aplicación construye una nueva petición HTTP hacia el servidor, enviando la información de la línea, el identificador del colectivo, la ubicación de la parada seleccionada y datos adicionales para mejorar la precisión del cálculo, como los vectores de posición en la ruta. Esta petición es manejada en el método predictRoute, que se encarga de comunicarse con el servidor y procesar la respuesta. Si la respuesta es exitosa, se muestra el tiempo estimado de llegada en la pantalla, brindando al usuario una referencia confiable sobre cuánto tiempo debe esperar en la parada seleccionada.

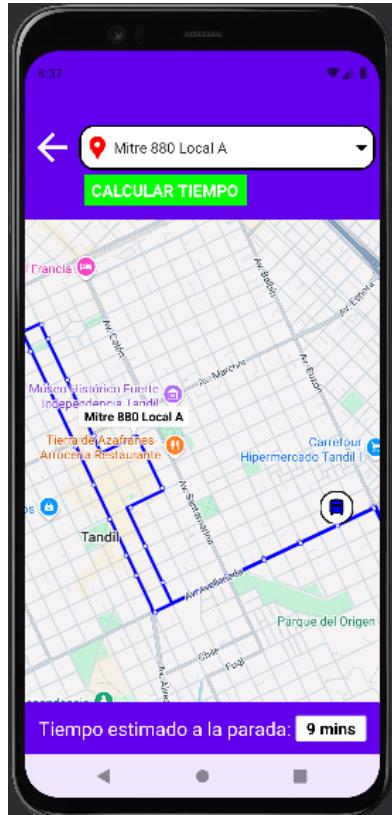


Figura 46 Predicción del Tiempo de Demora a una Parada Específica.

En caso de que la predicción falle, el sistema reintenta la consulta hasta un máximo de con la parada siguiente y luego la anterior, asegurando una mayor robustez en la funcionalidad. Esto se hace, al igual que todos los chequeos mencionados en el servidor, para agotar todas las posibilidades, ya que hay ciertas paradas que pueden ser salteadas en la predicción. En total se hacen 540 predicciones antes de informar al usuario que no es posible calcular el tiempo para esa parada seleccionada. En los casos excepcionales la respuesta tarda un poco más pero se optó por esta solución ya que se consideró que era preferible que tarde más a que no responda. Todo este proceso permite que el usuario consulte, de manera sencilla e intuitiva, cuánto falta para que un colectivo específico llegue a una determinada parada, optimizando así su tiempo de espera y mejorando la experiencia de uso de la aplicación.

Capítulo 6: Conclusión

El presente trabajo ha abordado el desarrollo de la aplicación móvil BusNow para la estimación de tiempos de espera y la localización en tiempo real de colectivos en la ciudad de Tandil, atendiendo a una necesidad insatisfecha en el sistema de transporte público local. El principal objetivo fue ofrecer una solución eficiente, accesible y adaptable a los requerimientos de los usuarios, el cual fue alcanzado con éxito. Para hacerlo, a lo largo del informe, se analizaron diversas estrategias y modelos de predicción permitiendo tomar las decisiones más adecuadas al contexto.

Entre los logros más significativos del proyecto se encuentra la implementación de un sistema basado en técnicas de *Machine Learning* para estimar tiempos de llegada y recomendar rutas óptimas. Asimismo, se adoptó una arquitectura que favorece la escalabilidad y la actualización en tiempo real, aprovechando bases de datos en la nube y tecnologías de geolocalización.

Uno de los principales desafíos enfrentados fue la recolección y el procesamiento de datos en un entorno donde no se dispone de información oficial estructurada sobre el tráfico y la frecuencia del transporte público. Para superar esta limitación, se diseñó un esquema de colaboración donde los propios usuarios pueden contribuir con información relevante. No obstante, esta metodología también implica la necesidad de mecanismos de validación y filtrado de datos para evitar posibles inconsistencias o información errónea.

En cuanto a la implementación de la aplicación, se priorizó el diseño de una interfaz intuitiva y adaptable a distintos dispositivos, garantizando que la experiencia del usuario sea fluida y accesible. Se evaluaron diversas tecnologías para la integración a la app, seleccionando aquellas que optimizan el rendimiento sin comprometer la experiencia de uso. Además, se consideraron aspectos de seguridad y privacidad, asegurando que la información personal de los usuarios esté protegida y que la aplicación cumpla con buenas prácticas en cuanto al manejo de datos sensibles.

Si bien los resultados obtenidos son alentadores, existen varias áreas que pueden explorarse en futuros desarrollos. Entre los principales avances se plantea la incorporación de notificaciones en tiempo real para el seguimiento de los recorridos seleccionados, así como la posibilidad de reportar incidencias por parte de los usuarios y mantenerse informado mutuamente de inconvenientes en el uso del transporte público. También se considera la posibilidad de integrar información en tiempo real proveniente de sensores urbanos o fuentes de datos externas, lo que permitiría mejorar las predicciones incorporando variables adicionales como el estado del tráfico, las condiciones climáticas o eventos especiales que puedan afectar el flujo del transporte público.

El desarrollo de este proyecto representó un desafío integral que combinó múltiples aspectos de la Ingeniería de Sistemas, desde la planificación y diseño hasta la implementación y validación de una solución tecnológica. El proceso permitió aplicar y consolidar numerosos conocimientos adquiridos durante la carrera e incorporar algunos faltantes. Entre el primer grupo se destacan las habilidades en programación, manejo de bases de datos, algoritmos, estructuras de datos y modelos de *Machine Learning*. La elección de tecnologías y su posterior implementación requirió una combinación de conocimientos teóricos y prácticos adquiridos en distintas asignaturas de la carrera. Por otro lado, si bien durante el periodo de formación no se adquirió conocimiento explícito en ciertas áreas como el desarrollo móvil, lenguaje kotlin, manejo de servidores, por mencionar algunas, si se brindaron las herramientas para que el aprendizaje de las mismas sea mucho más ameno y rápido, preparando a los alumnos para su futura inserción al mundo laboral.

Más allá del aspecto técnico, este trabajo evidenció la importancia de habilidades complementarias como la gestión de proyectos, la toma de decisiones basada en datos y la capacidad de resolución de problemas. Uno de los principales aprendizajes fue la necesidad de adaptar los modelos y la infraestructura del sistema a las condiciones del entorno real. La experiencia también reforzó la importancia de la documentación, aspecto clave en cualquier entorno profesional.

Desde una perspectiva formativa, este trabajo permitió identificar tanto fortalezas como áreas de mejora. Entre las fortalezas, la carrera brindó una base sólida en desarrollo de software, modelado de sistemas y optimización de algoritmos, lo que permitió abordar con conocimiento y confianza los distintos desafíos técnicos del proyecto. Sin embargo, se evidenciaron áreas en las que sería beneficioso profundizar, como la gestión de proyectos, seguridad informática y documentación detallada.

En conclusión, el desarrollo de esta aplicación representa un avance en la mejora del transporte público en Tandil, proporcionando una herramienta tecnológica que facilita la movilidad y optimiza la experiencia de los usuarios. La implementación del sistema ha permitido validar la viabilidad de modelos predictivos en este contexto, sentando las bases para futuras mejoras y ampliaciones. Con la evolución de las tecnologías y la incorporación de nuevas fuentes de datos, el proyecto tiene el potencial de seguir creciendo y consolidarse como una solución integral para la gestión eficiente del transporte público en la ciudad. Este representó no sólo un desafío técnico y académico, sino también una experiencia formativa integral, en la que se aplicaron y consolidaron conocimientos fundamentales de la carrera. La combinación de habilidades técnicas con la capacidad de adaptación y resolución de problemas permitió desarrollar una solución innovadora con impacto real en la comunidad, reafirmando el rol de la tecnología como herramienta para mejorar la calidad de vida de las personas.

Bibliografía

- Cardona, O. A. S., Zea, J. D. H., Santa Chávez, J. J., & Escobar, J. W. (2015). Modelos de regresión lineal para estimación de tiempos de viaje en sistemas de transporte masivo. *Ciencia e Ingeniería Neogranadina*, 25(1), 77-89.
- Suwardo, W., Napiah, M., & Kamaruddin, I. (2010). ARIMA models for bus travel time prediction. *J. Inst. Eng. Malaysia*, 71(2), 49-58.
- Li, J. (2017, April). Bus arrival time prediction based on random forest. In *2017 5th International Conference on Frontiers of Manufacturing Science and Measuring Technology (FMSMT 2017)* (pp. 867-872). Atlantis Press.
- Ge, Z., Yang, L., Li, J., Chen, Y., & Xu, Y. (2024). Bus Schedule Time Prediction Based on LSTM-SVR Model. *Mathematics*, 12(22), 3589.
- Zhao, J., Gao, Y., Qu, Y., Yin, H., Liu, Y., & Sun, H. (2018). Travel time prediction: Based on gated recurrent unit method and data fusion. *IEEE Access*, 6, 70463-70472.
- Chen, Z., Wen, J., & Geng, Y. (2016, November). Predicting future traffic using hidden markov models. In *2016 IEEE 24th international conference on network protocols (ICNP)* (pp. 1-6). IEEE.
- Xiao, W., & Wang, X. (2023). Spatial-temporal dynamic graph convolutional neural network for traffic prediction. *IEEE Access*, 11, 97920-97929.
- Liu, Y., Yu, H., & Fang, H. (2021, October). Application of KNN prediction model in urban traffic flow prediction. In *2021 5th Asian Conference on Artificial Intelligence Technology (ACAIT)* (pp. 389-392). IEEE.
- Vangeneugden, M., Luong, N. Q., & Mercelis, S. (2024, June). Towards Efficient Rail Transportation: Bayesian Network Modeling for Predicting Passenger Train Delays Using Secondary Train Information. In *2024 IEEE Conference on Artificial Intelligence (CAI)* (pp. 1425-1431). IEEE.

Repositorio GitHub

Todo lo trabajado durante el desarrollo de este proyecto se encuentra públicamente disponible en un repositorio Github. Asimismo, en este también se encuentra compartido el dataset curado con toda la información recopilada para la realización de esta aplicación permitiendo que pueda ser utilizada por quien la necesite. La misma incluye recorrido y paradas de cada una de las líneas de colectivos de la ciudad de Tandil.

[Repositorio GitHub BusNow](#)