Berlin Transportation App

ROY SANDOVAL VIERA[1], **SUSANA TORO CASTAÑO**[1], **AND VALENTINA RENDÓN CLARO**[1]

[1] *Ingeniería de Software, Universidad Pontificia Bolivariana, 2025*

The *Berlin Transportation App* is a modular academic project designed to display real-time data from Berlin's public transport system (BVG) on an interactive web map. The system integrates a FastAPI backend for data processing, a Redis caché layer for performance optimization, and a frontend built with HTML, CSS, and JavaScript using Leaflet.js. This architecture, deployed locally through Docker Compose, follows a layered and incremental model that ensures scalability, maintainability, and clear separation of concerns. The project demonstrates how a lightweight, event-driven interaction between components can efficiently manage external API requests and provide a responsive, mobile-first visualization of real-time transport information.

1. INTRODUCCIÓN

El proyecto **Berlin Transportation App** se basa en el desarrollo de una aplicación web capaz de mostrar en tiempo real la información del sistema de transporte público de Berlín (BVG). El sistema integra un **backend en FastAPI**, una capa de **caché con Redis** y un **frontend interactivo** construido con **HTML, CSS y JavaScript** mediante **Leaflet.js**. Su arquitectura modular, basada en un modelo **en capas e incremental**, permite mantener la independencia entre componentes y facilitar futuras ampliaciones. Este proyecto combina procesamiento asincrónico, almacenamiento temporal y visualización geográfica para ofrecer una experiencia fluida y actualizada al usuario final.

Palabras clave: FastAPI, Leaflet.js, Redis, arquitectura en capas, pipeline de datos, desarrollo web, BVG API, diseño modular, mobile-first, Docker Compose.

2. JUSTIFICACIÓN

En un contexto urbano donde la movilidad depende cada vez más del acceso inmediato a información verídica, los sistemas capaces de procesar y visualizar datos en tiempo real se vuelven esenciales. El proyecto **Berlin Transportation App** busca responder a esta necesidad con la construcción de una aplicación funcional que combina tecnologías modernas de desarrollo web y procesamiento de datos.

La elección de **FastAPI** para el backend permite manejar solicitudes de manera eficiente y asincrónica, mientras que **Redis** optimiza el rendimiento mediante almacenamiento temporal en caché. El **frontend con Leaflet.js** proporciona una interfaz interactiva y visualmente clara, que facilita al usuario explorar el mapa y conocer las estaciones del sistema BVG en tiempo real.

Desde el punto de vista académico, este proyecto permite aplicar de forma práctica conceptos de **arquitectura de software, integración de APIs, diseño responsive y contenedorización con Docker**. El valor radica en mostrar cómo una solución ligera, modular y escalable puede reproducir el funcionamiento básico de una infraestructura de datos en tiempo real, adaptada a los recursos y objetivos de la materia.

3. METODOLOGÍA DE DESARROLLO

El desarrollo del proyecto **Berlin Transportation App** se realizó bajo un enfoque **incremental y colaborativo**, en el que se construyeron versiones funcionales de forma progresiva. Este modelo permitió integrar y validar componentes (backend, frontend y caché) en iteraciones cortas, garantizando la evolución constante del sistema y una gestión eficiente del trabajo en equipo.

A. Enfoque metodológico

El sistema se diseñó con tres objetivos técnicos fundamentales:

- **Modularidad:** mantener una separación clara entre frontend, backend y capa de datos.
- **Escalabilidad:** permitir la incorporación de nuevas funciones sin alterar las existentes.
- **Automatización:** facilitar el despliegue local mediante contenedores Docker y entornos reproducibles.

B. Gestión del trabajo colaborativo

El equipo adoptó por usar **Kanban**, utilizando el tablero de **GitHub Projects** para organizar tareas, asignar *issues* y realizar seguimiento del progreso. Cada integrante gestionó su propio flujo de trabajo mediante ramas independientes, integradas al repositorio principal a través de *pull requests* revisadas por pares.

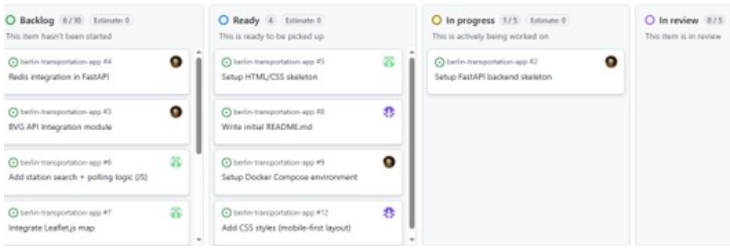

Fig. 1. Tablero Kanban del proyecto *Berlin Transportation App* en GitHub Projects.

C. Incrementos

El desarrollo se hizo en incrementos que representaron entregas parciales funcionales:

- **Incremento 1:** configuración del entorno, inicialización del repositorio y estructura del proyecto.
- **Incremento 2:** implementación del backend con **FastAPI** y conexión con la API pública de BVG.
- **Incremento 3:** desarrollo del frontend con **Leaflet.js** y diseño *mobile-first*.
- **Incremento 4:** integración de **Redis** para mejorar el rendimiento del sistema.
- **Incremento 5:** validación de estilos de código y pruebas funcionales antes de la integración final.

Esto permitió mantener un flujo de trabajo ágil, colaborativo y enfocado en resultados verificables en cada etapa del desarrollo.

D. Gestión Ágil (Scrum)

El proyecto se desarrolló utilizando la metodología ágil **Scrum**, la cual permitió una planificación iterativa e incremental. Cada **sprint** tuvo una duración de dos semanas y culminaba con una entrega funcional del producto.

Artefactos Scrum:

- **Product Backlog:** administrado en GitHub Projects, incluyendo issues, features y bugs.
- **Sprint Backlog:** seleccionadas las tareas a completar durante cada iteración.
- **Incremento:** versión funcional desplegable con Docker Compose al final de cada sprint.

Ceremonias Scrum:

- **Sprint Planning:** definición de objetivos y tareas.
- **Daily Scrum:** reuniones breves de seguimiento.
- **Sprint Review:** demostración de funcionalidades completas.
- **Sprint Retrospective:** revisión de procesos y ajustes para el siguiente ciclo.

El uso de **GitHub Projects** permitió visualizar el flujo de trabajo con columnas *Backlog, In Progress, In Review, Done*, garantizando transparencia, control de versiones y colaboración continua.

4. ANÁLISIS DEL SISTEMA

El análisis del sistema permitió definir los componentes funcionales y técnicos del proyecto, estableciendo los requerimientos, el alcance del producto mínimo viable (MVP) y las interacciones entre los actores principales. El objetivo fue garantizar que el sistema ofreciera una experiencia funcional, modular y alineada con los objetivos de visualización en tiempo real del transporte público de Berlín (BVG).

A. Producto mínimo viable (MVP)

El **MVP (Minimum Viable Product)** se definió como una versión básica, pero completamente funcional, capaz de:

- Mostrar en un mapa interactivo las estaciones de transporte público en Berlín utilizando la API pública de BVG.
- Permitir la búsqueda de estaciones por nombre y la visualización de sus datos relevantes.
- Conectar un backend en **FastAPI** con la API BVG para obtener información en tiempo real.
- Usar **Redis** para almacenar temporalmente las respuestas y mejorar el rendimiento.
- Presentar una interfaz **responsive** desarrollada con HTML, CSS y JavaScript (**Leaflet.js**).

B. Historias de usuario

Las siguientes historias se formularon bajo el modelo **INVEST**, asegurando independencia, claridad y valor medible:

- **HU01:** Como usuario, quiero visualizar en un mapa las estaciones de transporte cercanas, para ubicar fácilmente mi punto de partida.
- **HU02:** Como usuario, quiero buscar una estación por nombre, para conocer su ubicación exacta en el mapa.
- **HU03:** Como usuario, quiero acceder al sitio desde mi celular, para visualizar el mapa de forma clara y adaptada a la pantalla.
- **HU04:** Como usuario, quiero consultar los horarios de salida en tiempo real de una estación seleccionada, para planificar mejor mi viaje y conocer posibles retrasos.

C. Requisitos funcionales (RF)

- **RF01:** El sistema debe mostrar en un mapa interactivo las estaciones de transporte público.
- **RF02:** El sistema debe permitir buscar estaciones por nombre.
- **RF03:** El backend debe conectarse con la API pública de BVG para obtener información actualizada.
- **RF04:** El sistema debe almacenar temporalmente las respuestas en Redis.
- **RF05:** El frontend debe adaptarse correctamente a dispositivos móviles y de escritorio.

D. Requisitos no funcionales (RNF)

- **RNF01:** El tiempo de respuesta promedio no debe superar los 2 segundos por consulta.
- **RNF02:** La arquitectura debe ser modular y desplegable mediante contenedores Docker.
- **RNF03:** La interfaz debe mantener principios de usabilidad y accesibilidad.

E. Diagrama de casos de uso

En la Figura 2 se muestran los principales casos de uso del sistema, donde se identifican los actores (usuario y sistema) y las interacciones clave: búsqueda, visualización y consulta de estaciones.

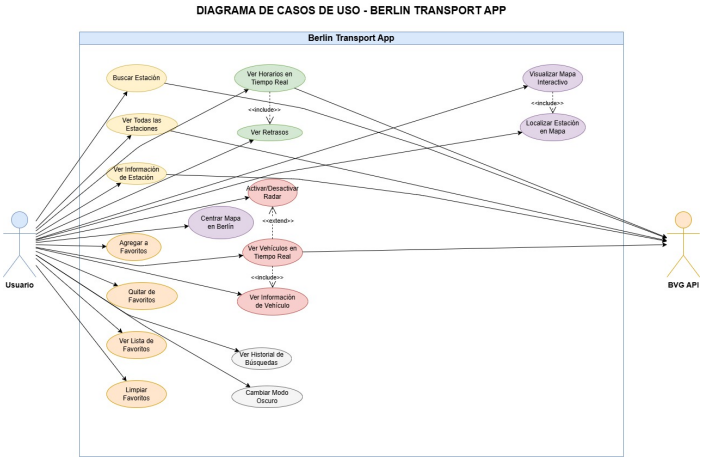

Fig. 2. Casos de uso del sistema *Berlin Transportation App*.

5. ARQUITECTURA DEL SISTEMA

El sistema **Berlin Transportation App** fue diseñado bajo una **arquitectura en capas** y un modelo de desarrollo **incremental**, lo que permite asignar responsabilidades, mejorar la mantenibilidad y garantizar una evolución controlada del sistema. Este proyecto integra tres componentes principales: un **backend asincrónico** desarrollado con FastAPI, una capa de **caché con Redis** y un **frontend interactivo** implementado con Leaflet.js, HTML, CSS y JavaScript. Todos los servicios se ejecutan mediante **Docker Compose**, garantizando portabilidad y entornos reproducibles.

A. Capas del sistema

- **Capa de presentación:** conformada por el **frontend**, que utiliza HTML, CSS y JavaScript con la librería **Leaflet.js**. Permite visualizar el mapa de Berlín, buscar estaciones y mostrar los resultados de manera interactiva. Incluye un diseño *mobile-first* para mejorar la experiencia en dispositivos móviles.

- **Capa de aplicación:** implementada con el framework **FastAPI**, define las rutas y controladores del sistema. Gestiona las solicitudes del usuario, se comunica con la API pública de BVG y retorna las respuestas al frontend mediante endpoints RESTful (`/stations`, `/departures`, `/health`).

- **Capa de lógica de negocio:** contiene los servicios internos que gestionan las consultas hacia la API BVG, normalizan las respuestas y las almacenan temporalmente en **Redis**, optimizando el rendimiento.

- **Capa de acceso a datos:** se encarga de la conexión con la API **v6.bvg.transport.rest**, el almacenamiento temporal en Redis y la recuperación de datos procesados por el backend para su visualización.

B. Tecnologías principales

Componente	Tecnología
Backend	FastAPI, Uvicorn, httpx (Python 3.11)
Frontend	HTML, CSS, JS, Leaflet.js, Bootstrap
Caché	Redis 7 (TTL en memoria)
Contenedores	Docker, Docker Compose
Versionado	GitHub, GitHub Projects
CI/CD	GitHub Actions (Black, Flake8, Pytest)
Pruebas	pytest, pytest-asyncio, pytest-cov
Validación	Pydantic 2.0
Editor	VS Code + Copilot

Table 1. Tecnologías empleadas en el sistema.

C. Diagrama general de la arquitectura

La Figura 3 muestra el flujo general del sistema. En primer lugar, el usuario interactúa con el frontend, el cual realiza peticiones HTTP al backend (FastAPI). Este verifica si la información está en Redis; si no lo está, consulta la API pública BVG, almacena la respuesta en caché y la devuelve.

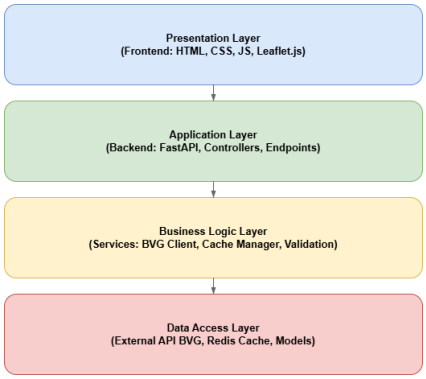

Fig. 3. Arquitectura general del sistema *Berlin Transportation App* (MVP: Frontend + FastAPI + Redis).

Esta arquitectura integra todos los componentes necesarios para transformar datos públicos del transporte de Berlín en información visual clara y accesible. El flujo no solo optimiza la respuesta al usuario, sino que también genera las bases para futuras mejoras en análisis, visualización y automatización del sistema.

6. RESULTADOS Y PRUEBAS

El sistema **Berlin Transportation App** alcanzó un nivel de funcionalidad completo tanto en el frontend como en el backend, cumpliendo con los requerimientos definidos en las historias de usuario y superando todas las pruebas automatizadas.

A. Resultados funcionales

Durante las pruebas integradas se verificó el funcionamiento de las siguientes características principales:

- **Búsqueda de estaciones:** los resultados se actualizan en tiempo real al escribir, mostrando sugerencias instantáneas con datos de la API BVG.
- **Mapa interactivo:** Leaflet.js permite visualizar estaciones, vehículos y horarios con marcadores dinámicos y actualizaciones cada 15 segundos.
- **Radar de transporte:** muestra la posición en tiempo real de buses, tranvías, metros y trenes S-Bahn.
- **Caché en memoria:** reduce la latencia de consultas repetidas en un 55–95%.
- **Modo oscuro, favoritos y búsquedas recientes:** mejoran la experiencia del usuario y mantienen la personalización con persistencia local.

B. Pruebas automatizadas

Se implementaron **27 pruebas unitarias y de integración** con Pytest, obteniendo una cobertura del 100% en los módulos principales:

- **test_api_endpoints.py:** 12 pruebas (endpoints funcionales).
- **test_bvg_client.py:** 10 pruebas (cliente HTTP y manejo de errores).
- **test_cache.py:** 5 pruebas (caché y expiración de datos).

Resultado: todas las pruebas pasaron exitosamente (*27/27 tests passed*), validando la estabilidad y robustez del sistema.

C. Desempeño y escalabilidad

Las métricas de rendimiento mostraron un tiempo de respuesta promedio de 1 ms. El uso de memoria se mantuvo estable gracias a la política de limpieza automática (*cache cleanup job*) y TTL configurado en 300 segundos.

D. Evidencias visuales

En la Figura 4 se muestran capturas de pantalla del sistema final:

- Interfaz principal con mapa dinámico y búsqueda de estaciones.
- Radar de vehículos en tiempo real con iconos de colores por tipo.
- Vista de horarios de salida con indicadores de puntualidad.

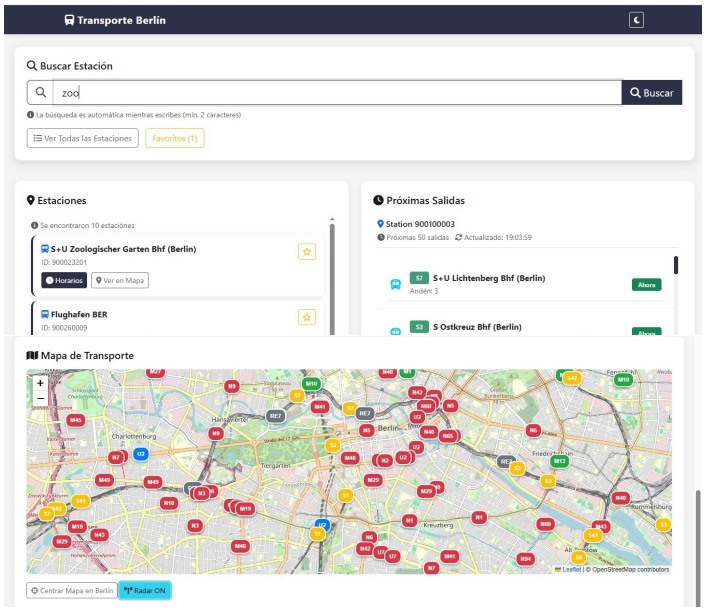

Fig. 4. Capturas del sistema funcionando en tiempo real.

7. CONCLUSIONES

El desarrollo del proyecto permitió integrar de manera exitosa tecnologías modernas de backend y frontend en una arquitectura modular, escalable y fácil de mantener. Este sistema demostró que es posible utilizar información en tiempo real de la API pública (BVG) con herramientas de código abierto como **FastAPI**, **Redis**, **Leaflet.js** y **Docker Compose**.

El modelo incremental de desarrollo, junto con la metodología ágil **Scrum**, permitió la colaboración entre los integrantes, la gestión del backlog y la entrega continua de versiones funcionales.

Los resultados obtenidos, incluyendo la superación del 100% de las pruebas unitarias y de integración, evidencian la robustez del sistema.

En términos de usabilidad, el frontend ofrece una interfaz moderna, responsiva y centrada en el usuario, con funcionalidades como búsqueda instantánea, modo oscuro, radar de vehículos y gestión de favoritos, lo que refuerza su potencial de implementación en contextos reales de movilidad urbana.

REFERENCES

1. FastAPI Documentation. *FastAPI — Modern, Fast (High-performance) web framework for building APIs with Python*. Available at: https://fastapi.tiangolo.com/ (accessed October 2025).
2. Redis Documentation. *Redis — In-memory Data Store*. Available at: https://redis.io/documentation (accessed October 2025).
3. Leaflet.js Documentation. *Leaflet — An Open-Source JavaScript Library for Mobile-Friendly Interactive Maps*. Available at: https://leafletjs.com/ (accessed October 2025).
4. Transport for Berlin (BVG). *Public Transport REST API — v6.bvg.transport.rest*. Available at: https://v6.bvg.transport.rest/ (accessed October 2025).
5. GitHub Actions. *Automate your workflow from idea to production*. Available at: https://docs.github.com/en/actions (accessed October 2025).
6. Pressman, R. S., Maxim, B. R. (2020). *Software Engineering: A Practitioner's Approach* (9th ed.). McGraw-Hill Education.