Universidad Nacional de Colombia - sede Bogotá
Facultad de Ingeniería
Departamento de Ingeniería de Sistemas
Curso: Ingeniería de Software

Estudiantes:

- Andres Felipe Arias Gonzalez
- Juliana Alejandra Nieto Cárdenas
- Misael Jesús Florez Anave
- Nicolas Betancur Sanchez

Grupo 1: Formula 1 - (2025-1) Martes y Jueves.

Fórmula 1

1. Para *VecirrUN* necesitamos actualización instantánea de la disponibilidad de bicicletas y la inserción de un mapa OSM. Flet ofrece esto casi “out-of-the-box”, con menor fricción que montar WebSockets sobre Kivy o añadir QtWebEngine a PySide. Además, su API puramente Python facilita que todo el equipo contribuya sin aprender Kv-lang ni C++ idioms

Criterio	Flet (elegido)	Kivy	PySide 6
Paradigma UI	Reactivo; API 100 % Python que transpila a **Flutter**	Propio (Kv + Python); escena OpenGL	Clásico Qt (imperativo, signal/slot)
Instalación/paquetes	pip install flet; ejecuta un mini-servidor WebSocket para hot-reload	pip install kivypero dependencias gráficas y drivers extra	pip install pyside6; instala bindings Qt ~120 MB
Widgets nativos de escritorio	Basados en Flutter desktop (Windows, macOS, Linux) con look moderno	Widgets propios; aspecto “móvil” menos nativo	Widgets Qt nativos (alta fidelidad)
HTML/OSM embebible	Control HtmlElement permite	Requiere kivy.garden.mapview;	QtWebEngine embebe OSM pero

	incrustar **Leaflet/OpenStreetMap** en una vista Flet sin salir de Python	menos flexible	añade 100 MB extra
Comunicación en tiempo real	WebSocket integrado; basta page.pubsub.send() para broadcast entre sesiones (ideal para disponibilidad en vivo)	Debe montarse servidor aparte (e.g. FastAPI + WS)	Qt WebSockets disponible, pero no integrado al loop principal; más boilerplate
Curva de aprendizaje	Muy corta para quien conozca Flutter o React-like	Kv-lang + Canvas propio; mayor curva	Más compleja (C++-like API, múltiples patrones)
Comunidad & madurez	Emergente (< 3 años); menos ejemplos, pero compatible con ecosistema Flutter	+10 años; activa pero centrada en apps móviles	Muy madura; enorme comunidad Qt
Tamaño binario final*	35-60 MB con PyInstaller (usa motor Flutter comprimido)	25-40 MB, depende de SDL & deps	> 120 MB por Qt + QtWebEngine

El tipo de base de datos relacional que se está utilizando es PostgreSQL

A continuación nombraremos las herramientas y bibliotecas planeadas para el desarrollo del proyecto, además de una justificación orientada a las necesidades del equipo y relación a los objetivos que buscamos llegar con el proyecto:

Alembic

Necesidad del proyecto: posibilita el versionamiento y la migración del esquema de la base de datos conforme evoluciona el sistema de préstamos de bicicletas; evita pérdidas de datos y mantiene la consistencia entre entornos.

Capacidades del equipo: se integra de forma natural con SQLAlchemy, que el equipo ya domina; su línea de comandos (revision, upgrade, downgrade) es sencilla de aprender.

Relación con los objetivos del curso: promueve el control riguroso del ciclo de vida del software y la iteración ágil, reforzando la práctica de “documentar y probar aplicaciones” con un historial claro de cambios en la base de datos.

PyTest

Necesidad del proyecto: asegura la calidad mediante pruebas unitarias y de integración; facilita la detección temprana de errores y respalda el desarrollo guiado por pruebas (TDD).

Capacidades del equipo: la sintaxis basada en funciones y “fixtures” es intuitiva, permitiendo que todos los integrantes escriban casos de prueba con rapidez.

Relación con los objetivos del curso: cumple el objetivo de “probar aplicaciones de software” y fomenta prácticas modernas de desarrollo como integración continua y entrega frecuente.

OpenStreetMap (OSM)

Necesidad del proyecto: proporciona datos cartográficos abiertos para ubicar estaciones de préstamo en un mapa real, enriquecer la experiencia del usuario y permitir futuras funcionalidades de ruteo o análisis geoespacial.

Capacidades del equipo: OSM ofrece APIs y bibliotecas Python (p. ej. osmnx, folium) que se integran sin cambiar el stack principal; el uso de estándares abiertos facilita la colaboración y evita licencias restrictivas.

Relación con los objetivos del curso: introduce la integración de servicios externos y principios de arquitectura modular, reforzando la comprensión de patrones de diseño y la construcción de sistemas extensibles.

Docker

Necesidad del proyecto: empaqueta la aplicación, la base de datos y dependencias en contenedores reproducibles, garantizando que el sistema se ejecute de forma idéntica en cualquier entorno (desarrollo, pruebas, producción).

Capacidades del equipo: permite aislar configuraciones complejas sin requerir instalaciones locales extensas; con archivos Dockerfile y docker-compose el despliegue es automatizado.

Relación con los objetivos del curso: fortalece la disciplina DevOps, complementa el control de versiones y apoya las metodologías ágiles al permitir entregas continuas y demostraciones consistentes.