Manual de Proyecto

Introducción

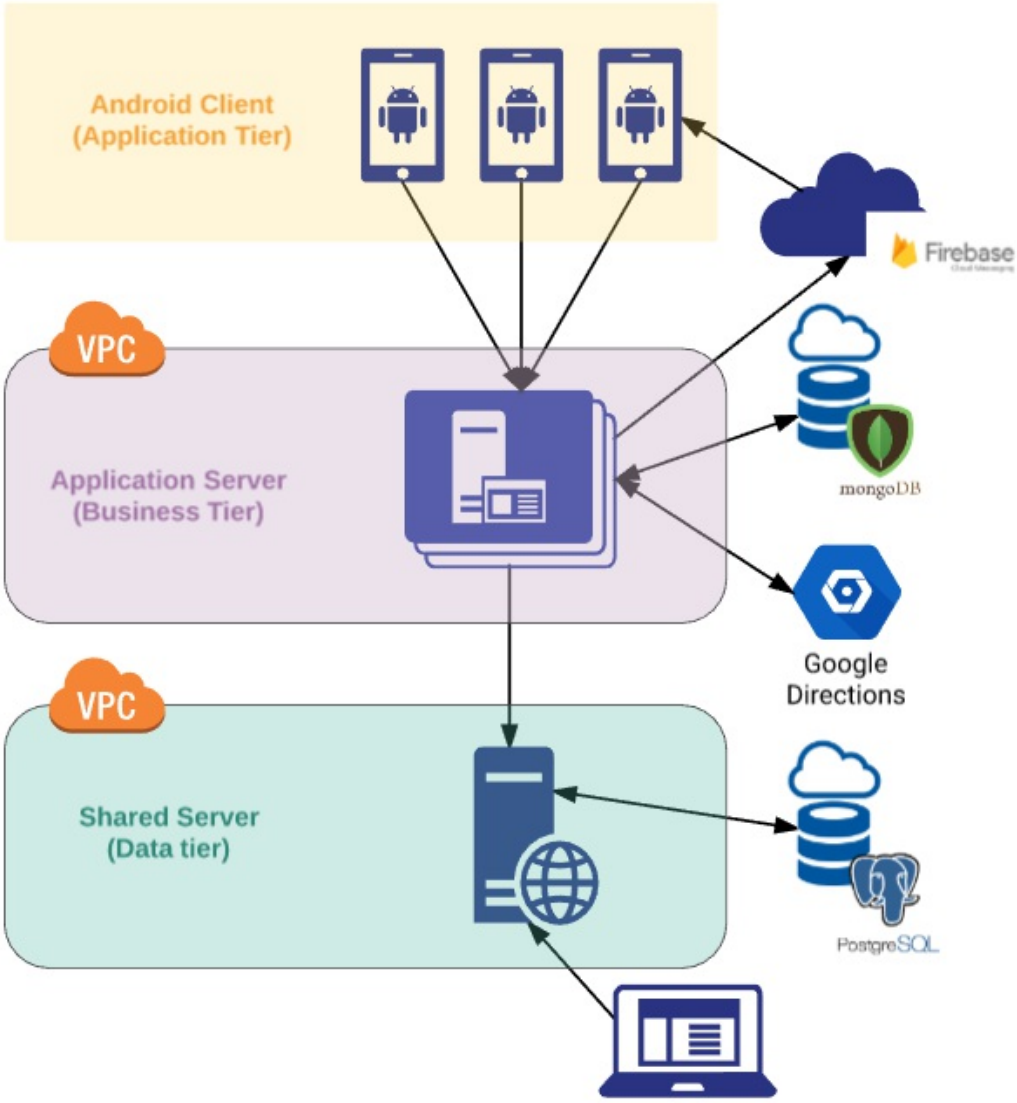
El proyecto **FIUBER** consistió en la implementación de una aplicación de viajes una aplicación intencionada para conectar a los pasajeros con los conductores de vehículos que ofrecen servicio de transporte particular.

El alcance del proyecto consiste en la implementación completa de la arquitectura de la aplicación, que está dada en 3 capas: la aplicación android de los clientes, un application server (por región) que contiene la lógica de negocios, y un shared server que ofrece servicio de base de datos centralizado, y funcionalidades compartidas.

Cada una de las tres capas, con su respectiva documentación y manual de instalación puede encontrarse en los siguientes links:

- Cliente
- Application Server
- Shared Server

Un esquema de la arquitectura completa se muestra a continuación. Más adelante se indican los requerimientos de cada una de las partes, y para cada una de ellas la planificación y las decisiones de desarrollo más importantes.



Qué servicios provee FIUBER?

Esta aplicación ofrece al usuario los siguientes servicios:

- Servicio de Login de Usuarios.
- Edición y consulta de los datos de cuenta.
- Estimación de costo de viajes.
- Selección y solicitud de ruta de viaje.
- Selección y solicitud de conductor y vehículo para realizar un viaje.
- Tracking de posición de pasajeros y conductores en tiempo real (vía Google Maps).
- Servicio de pago de viaje realizado en función de la distancia recorrida.
- Consulta de historial de viajes.
- Data analytics.
- Administración de servidores y permisos de usuarios.
- Posibilidad de introducción de reglas para flexibilizar el cálculo de costos de viaje.

Cliente

Consiste en una aplicación mobile para Android.

El *Cliente Android* representa la aplicación en sí para los usuarios finales (pasajeros y choferes). Su función es brindarle a los mismos un punto de entrada al resto del sistema, ofreciéndoles la posibilidad de crear o aceptar viajes de forma eficiente e intuitiva. Para una mayor performance y un diseño más estructurado, el *Cliente Android* delega una buena parte de la lógica de la aplicación en el *Application Server*, contra el cual se comunica constantemente.

Application Server

Un servidor w eb HTTP implementado en Python usando Gunicorn y Flask como framew orks.

El *Application Server* cumple la función de almacenar todos los datos pertinentes a los servicios provistos por la aplicación. Por ejemplo, los estados actuales de los usuarios y los viajes en curso se almacenan en esta capa. El *App Server* también funciona como gestor de transacciones contra el *Shared Server*, que es donde se almacenan todos los datos historicos de usuarios y viajes. Además de almacenar los datos de las operaciones en curso, el *App Server* provee la lógica de transición entre estados de los usuarios y el servicio de posicionamiento de choferes y pasajeros.

Shared Server

Un servidor w eb HTTP implementado en Javascript usando nodeJs. También presenta una interfaz gráfica implementada con Angular.

El *Shared Server* es el servidor que almacena los datos de usuarios y viajes a largo plazo. Está encargado de manejar y validar cualquier modificación realizada sobre el set de datos, garantizar su integridad y proveer seguridad sobre las transacciones. El *Shared Server* también provee una interfaz para interactuar con los usuarios administradores, para permitir la definición de permisos y reglas sobre los pagos.

Herramientas utilizadas

En el diagrama de arquitectura se presentó una visión general de las herramientas principales utilizadas en cada capa. El lector podrá optar por indagar sobre las herramientas específicas utilizadas en cada nivel, acudiendo a los manuales de cada uno (links en la introducción). Existen, sin embargo, algunas herramientas que resultaron ser más cruciales que otras, facilitando el desarrollo en paralelo continuo e incremental.

- Travis CI: la integración continua con Travis fue fundamental para el desarrollo del proyecto. Además de permitir una evolución progresiva y ordenada, Travis posibilita la ejecución de numerosa cantidad de comandos para automatizar tareas como el *deploy* en Heroku, la generación de reportes de coverage y la corrida de tests.

- Heroku: tanto para el *Cliente*, como para el *App Server* y el *Shared Server* procuramos mantener una versión funcional siempre corriendo en Heroku para poder permitir a las otras capas realizar pruebas y tener acceso a los logs.

- Docker: una de las primeras prioridades del proyecto consistió en incorporar Docker en todos los niveles para garantizar la compatibilidad y la consistencia en el comportamiento de cada aplicación.

Hitos de desarrollo

El desarrollo de este proyecto fue realizado en forma incremental, de forma tal que las funcionalidades fueron incorporadas de acuerdo con los siguientes hitos:

CP	Cliente	App Server	Shared Server	Común
#1	• Pantalla Login • Logs	• REST API • Deploy en Keroku • Generación de documentación • Logs • Unit Tests • Integración con Travis	• Deploy en Keroku • Generación de documentación • Servicio de gestión de App Servers • Logs • Unit Tests • Integración con Travis	• Mockups • Reporte de coverage
#2	• Pantalla principal • Vista perfil de usuario • Modificación perfil de usuario • Signup + Login contra App Server	• Servicio de autenticación • Modificación de datos de perfil de usuario • Servicio de choferes disponibles • Modelo de datos Mongodb de perfiles/choferes	• Servicio de gestión de usuarios de negocio • Servicio de gestión de datos de usuario • Modelo de datos de PostgreSql • Docker • Listado de Usuarios(backoffice)	• Diseño • Modelo de datos
#3	• Notificaciones push • Generación de documentación de código • Unit tests	• Servicio de posicionamiento • Servicio de viajes disponibles • Modelo de datos Mongodb completo	• Servicio de viajes • Servicio de cotización de viaje • Servicio de estado actual • Modelo de datos Postgresql completo • Estado actual (backoffice) • Historial de viajes (backoffice)	• Manual de instalación y configuración
Entrega	-	-	-	• Documentación actualizada • Funcionalidad completa

Puntos a mejorar y features pendientes

En general, se lograron alcanzar todos los puntos claves para cubrir el alcance definido inicialmente para el proyecto. Existen algunos elementos que nos hubiera gustado agregar pero que por restricciones temporales no nos fue posible incluir. Esta sección está destinada a enumerar algunos de esos aspectos.

Feature / Upgrade	Must Have	Should Have	Nice to Have
Sistema de recomendación de choferes			X
Servicio de reporte de denuncias			X
Botón de pánico			X
Notificaciones configurables			X
Más características de vehículos			X
Inicio y fin de viajes *User Friendly*		X	