**Arquitectura del Sistema**

**Integrantes:**  Rodrigo Gustavo Barra Pulgar

Octavio Andres Quezada Ayala

Paul Andres Lopez Ortiz

**Profesor:**  Fernando Gonzalo Herrera Francesconi

## **Resumen ejecutivo**

TeLoCambio adopta una arquitectura **Cliente–Servidor desacoplada**:

* **Frontend**: SPA con **React + TypeScript**.
* **Backend**: **Django REST Framework (DRF)** exponiendo **API REST**.
* **Base de datos**: **PostgreSQL**.
* **Tenencia**: **multi-tenant** con aislamiento estricto por comunidad.
* **Comunicación**: HTTP/HTTPS con JSON, API **versionada**.
* **Despliegue**: contenedores (Docker), entornos dev/staging/prod, CI/CD.
* **Alcance clave**: publicaciones de **bienes y servicios**, búsqueda y filtros, **agendamiento y doble confirmación**, **calificación 1–5 con conteo**, **panel administrador**, **auditoría**.
* **Negociación**: **sin chat interno**; la coordinación final se realiza por **WhatsApp** (deeplink). La **agenda y la doble confirmación** quedan **registradas en la plataforma**.

El proyecto **TeLoCambio** se construirá sobre una **arquitectura Cliente-Servidor completamente desacoplada**, un paradigma de diseño moderno que es fundamental para la agilidad y el crecimiento a largo plazo del sistema. Esto significa que la interfaz con la que interactúa el usuario (**Frontend**) y el motor que procesa toda la lógica y los datos (**Backend**) no son solo aplicaciones separadas, sino entidades autónomas que se comunican a través de un "contrato" bien definido: una **API**.

Este modelo se elige estratégicamente por sus múltiples ventajas:

* **Desarrollo Paralelo y Especialización:** Permite que los equipos de frontend y backend trabajen de manera independiente y simultánea. Mientras un equipo se especializa en crear una experiencia de usuario excepcional (UI/UX), el otro se enfoca en la eficiencia, seguridad y lógica del negocio.
* **Flexibilidad Tecnológica:** Al estar desacoplados, se podría reescribir completamente el frontend con una nueva tecnología en el futuro sin necesidad de tocar una sola línea de código del backend, y viceversa.
* **Escalabilidad Independiente:** Si la plataforma experimenta un alto tráfico de usuarios, podemos escalar los recursos del servidor (backend) sin afectar al cliente, y vice-versa.

## 

## 

## **Cliente (Frontend): Una Experiencia de Usuario Moderna con una SPA**

El cliente será una **Single-Page Application (SPA) o Aplicación de Página Única**, desarrollada con la librería **React**. Esta no es una página web tradicional. En lugar de recargar la página completa con cada clic, el navegador del usuario carga la aplicación **una sola vez**.

A partir de ese momento, toda la interacción del usuario, como buscar un objeto, filtrar resultados, se gestiona de forma **dinámica**. La aplicación solo solicita los datos estrictamente necesarios al servidor, y actualiza únicamente la parte de la pantalla que necesita cambiar. Esto resulta en una experiencia de usuario **extremadamente rápida y fluida**, similar a la de una aplicación de escritorio o móvil nativa.

## **Servidor (Backend): El Cerebro Centralizado a través de una API RESTful**

El servidor actuará como el "cerebro" y el guardián de toda la lógica y los datos del proyecto. Se desarrollará como una **API RESTful** utilizando **Django Rest Framework**. Su única responsabilidad es exponer la lógica del negocio de forma segura y eficiente. Sus tareas clave son:

* **Imponer las Reglas del Negocio:** Autenticar usuarios, validar que las transacciones sean lógicas, calcular la reputación.
* **Procesar y Servir Datos:** Es el único componente con acceso directo a la base de datos. Recibe peticiones, interactúa con la base de datos de manera segura y devuelve la información al cliente en un formato estándar y universal: **JSON** (JavaScript Object Notation). Al ser el único guardián de los datos, garantiza la integridad y seguridad de la información.
* **Ser Agnóstico a la Presentación:** Al backend no le importa si los datos serán consumidos por una página web, una aplicación móvil (iOS/Android) o incluso otro sistema. Su trabajo es simplemente entregar los datos correctamente a quien tenga los permisos para solicitarlos.

## 

## 

## 

## 

## **La Comunicación: El Puente a través de Peticiones HTTP**

El Frontend (cliente) se comunicará con el Backend (servidor) exclusivamente a través de **peticiones HTTP** dirigidas a diferentes *endpoints* (direcciones URL específicas) de la API. Este "lenguaje" de comunicación es el estándar de la web.

Por ejemplo:

* **GET /api/objetos/123**: Para **solicitar y ver** la información del objeto con el ID 123.
* **POST /api/objetos**: Para **crear** una nueva publicación de un objeto.
* **PUT /api/perfil**: Para **actualizar** la información del perfil de un usuario.
* **DELETE /api/objetos/123**: Para **eliminar** la publicación del objeto con el ID 123.

Una característica fundamental de esta comunicación es que es **apátrida (stateless)**. Cada petición que el cliente envía al servidor contiene toda la información necesaria para que el servidor la entienda y la procese por sí misma, sin necesidad de recordar peticiones anteriores. Esta característica es clave para la escalabilidad, ya que permite que cualquier instancia del servidor pueda atender cualquier petición en cualquier momento.

# **Definiciones Técnicas de Desarrollo Detalladas**

A continuación, se presenta un desglose exhaustivo de las tecnologías, herramientas y metodologías seleccionadas para cada componente del sistema, justificando cada elección en el contexto de los objetivos del proyecto.

## 

## 

## 

## 

## 

## 

## **Backend (Lado del Servidor)**

El backend es el motor central de la aplicación, responsable de la seguridad, los datos y toda la lógica de negocio.

* **Framework: Django Rest Framework (DRF) sobre Python**
  + Se eligió por su filosofía "baterías incluidas", que acelera drásticamente el desarrollo. DRF proporciona soluciones robustas y probadas para tareas complejas como la  
    **autenticación de usuarios** , la gestión de permisos y la **serialización** (el proceso de convertir los datos de la base de datos a formato JSON de manera eficiente y segura). Su **ORM (Object-Relational Mapper)** integrado no solo facilita la interacción con la base de datos, sino que también añade una capa de seguridad crucial al prevenir ataques comunes como la inyección SQL.
* **Base de Datos: PostgreSQL**
  + Se seleccionó por ser una base de datos relacional de código abierto reconocida por su **robustez y confiabilidad**. Para un proyecto como TeLoCambio, donde la integridad de las transacciones y las relaciones entre usuarios y objetos es crítica, PostgreSQL garantiza consistencia de datos. Es **altamente escalable**, lo que significa que puede manejar un crecimiento futuro en el volumen de usuarios y datos sin problemas.
* **Responsable Principal: Rodrigo Barra**
  + Rodrigo estará a cargo del desarrollo del backend, con un enfoque específico en la implementación de la  
     **seguridad y los sistemas de autenticación** del proyecto.

## **Frontend (Lado del Cliente)**

El frontend es la cara visible de TeLoCambio, responsable de ofrecer una experiencia de usuario fluida, intuitiva y segura.

* **Librería/Framework: React**
  + React se utilizará para construir una interfaz de usuario **moderna y reactiva**, donde los cambios de datos se reflejan en la pantalla casi instantáneamente sin necesidad de recargar la página. Su arquitectura se basa en **componentes reutilizables** (ej. un perfil de usuario, una tarjeta de producto), lo que no solo **agiliza el desarrollo**, sino que también **facilita el mantenimiento** a largo plazo, ya que un cambio en un componente se refleja en todos los lugares donde se utiliza.
* **Lenguaje: TypeScript**
  + Es un superconjunto de JavaScript que añade **tipado estático**, funcionando como un corrector de pruebas para el código antes de que se ejecute. Su uso es una decisión estratégica para **aumentar la robustez y fiabilidad del código**. Ayuda a detectar errores comunes durante la fase de desarrollo, no en producción, lo que es vital para un proyecto que manejará datos de usuarios y transacciones.
* **Responsable Principal: Octavio Quezada**
  + Octavio se enfocará en el desarrollo del frontend, siendo el responsable de traducir los diseños y la lógica en una **interfaz y experiencia de usuario (UI/UX) excepcionales**.

## **Herramientas y Entorno de Desarrollo**

Estas son las herramientas que permitirán al equipo colaborar de manera eficiente y asegurar la calidad del producto final.

* **Control de Versiones: GitHub**
  + Todo el código fuente se gestionará en un repositorio de GitHub. Más que un simple almacenamiento, GitHub facilitará la **colaboración ordenada** a través de *Pull Requests* (revisiones de código) y *Branches* (ramas para desarrollar funcionalidades de forma aislada). Esto asegura que se mantenga un **historial completo de cambios** y sienta las bases para la **integración continua (CI/CD)**.
* **Gestión de Proyecto: Azure DevOps o Trello**
  + Se usará una de estas herramientas para implementar la metodología  
     **Scrum** de forma visible y organizada. El **backlog del producto** actuará como la lista maestra de tareas, los **sprints** se planificarán para organizar el trabajo en ciclos cortos, y el **tablero Kanban** ofrecerá una visión clara del estado de cada tarea (Pendiente, En Progreso, Hecho).
* **Pruebas e Integración:**
  + Para garantizar la calidad del software , se realizarán **pruebas unitarias** (verificar que pequeñas piezas de código funcionen de forma aislada) y **pruebas de integración** (verificar que diferentes módulos interactúen correctamente entre sí).  
    **Paul López** será el responsable principal de esta área crítica, asegurando que el sistema sea estable y confiable.
* **Despliegue (Deployment):**
  + El Producto se desplegará en una plataforma de **hosting gratuito como Azure, Heroku o Netlify**. Estas plataformas simplifican el proceso de poner la aplicación online, permitiendo que sea **accesible públicamente** para la presentación final y para realizar pruebas piloto con usuarios reales. Con opción de utilizar BlueHosting.