

# **ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO**



## **FACULTAD:**

Facultad de Informática y Electrónica

## **PAO:**

Software 8° “1”

## **ASIGNATURA:**

Aplicaciones Informáticas II

## **TEMA:**

Diseño Arquitectónico de la Aplicación

## **ESTUDIANTE:**

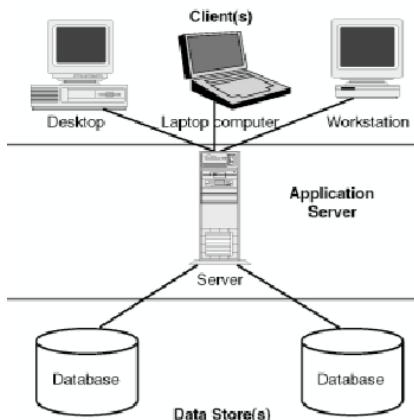
Steve Tibán (7386)

## **FECHA DE ENTREGA:**

28/10/2025

## 1. Tipo de Arquitectura

El sistema OralFlow adopta una arquitectura Cliente–Servidor, ampliamente utilizada en el desarrollo de aplicaciones web modernas por su simplicidad, escalabilidad y separación de responsabilidades. Este modelo se fundamenta en la interacción entre uno o varios clientes, que solicitan recursos o servicios, y un servidor, encargado de procesar las solicitudes y devolver las respuestas a través de una red de comunicación.



- **Cliente (Frontend):** Desarrollado con React, es responsable de la interfaz de usuario y de las interacciones con el sistema. Desde esta capa, pacientes, odontólogos y administradores pueden gestionar citas, fichas médicas y tareas administrativas mediante peticiones hacia el servidor.
- **Servidor (Backend):** Implementado con Django REST Framework, centraliza la lógica de negocio, procesa las operaciones según el rol del usuario y gestiona la persistencia de los datos en la base de datos.
- **Red de comunicación:** Actúa como medio de intercambio de datos entre cliente y servidor mediante el protocolo HTTP/HTTPS, garantizando acceso remoto, disponibilidad y sincronización en tiempo real.

La arquitectura Cliente–Servidor ofrece ventajas notables como bajo costo de implementación, facilidad de mantenimiento, escalabilidad y alta concurrencia de usuarios.

Estas características la convierten en la opción más adecuada para el proyecto, pues permite:

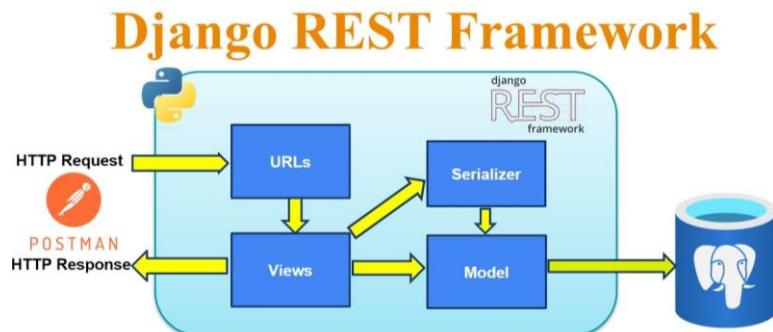
- Gestionar múltiples usuarios simultáneamente sin conflictos.
- Mantener una administración centralizada de datos.
- Separar de forma clara la interfaz, la lógica de negocio y la base de datos.
- Facilitar la integración de servicios externos, como Twilio, sin comprometer la estructura principal del sistema.

## 2. Patrón de Diseño

El sistema OralFlow se construye sobre el patrón de diseño Modelo–Vista–Template (MVT) propio del framework Django, el cual es una variante del tradicional Modelo–Vista–Controlador (MVC). Este patrón promueve la separación de responsabilidades dentro de la aplicación, organizando su estructura en capas que gestionan de forma independiente los datos, la lógica de negocio y la presentación.

No obstante, en este proyecto el componente Template no se utiliza, la capa visual es desarrollada de manera independiente en React, encargada de renderizar la interfaz de usuario y consumir los datos expuestos por el backend mediante peticiones HTTP. Por tanto, el patrón MVT se adapta a un enfoque

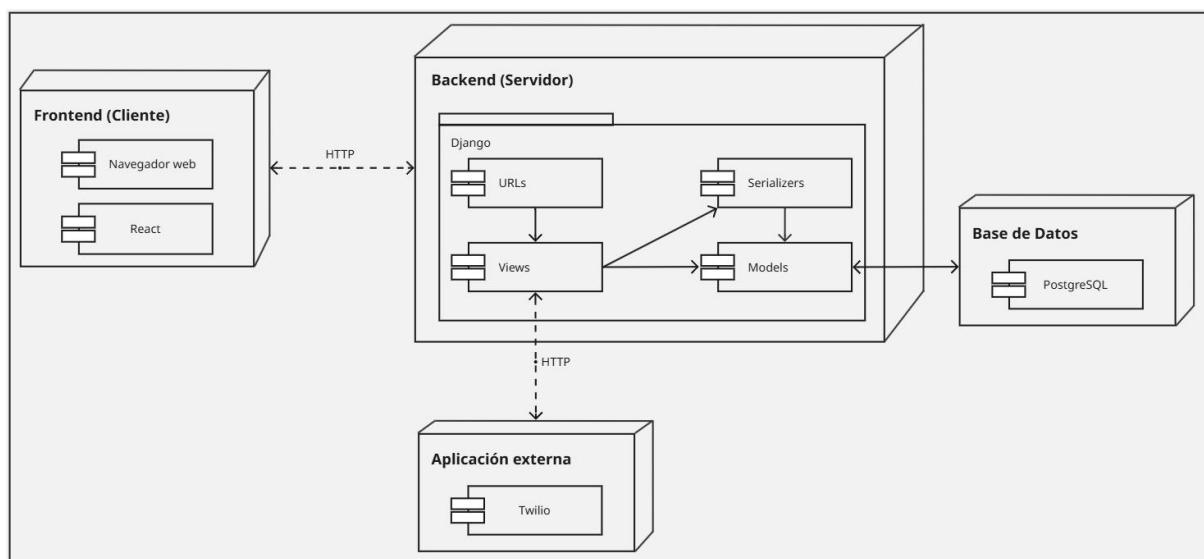
orientado a APIs REST, dando lugar a una estructura funcional equivalente a un patrón MVS (Model–View–Serializer), donde el Serializer sustituye el rol de la plantilla tradicional.



- **Modelo (Model):** Define la estructura de los datos y las relaciones entre entidades, representando objetos del dominio como Usuario, Paciente, Odontólogo, Cita, Ficha Médica, Consultorio, Horario y Bloqueo de Día. Estos modelos se gestionan mediante el ORM de Django, que permite manipular la base de datos PostgreSQL sin escribir consultas SQL directas.
- **Vista (View):** Contiene la lógica de negocio y el procesamiento de las peticiones. Las vistas reciben solicitudes HTTP del frontend, aplican las reglas del sistema (validaciones, confirmaciones, reprogramaciones, cancelaciones) y retornan las respuestas en formato JSON.
- **Serializador (Serializer):** Reemplaza al componente Template de Django tradicional. Se encarga de validar los datos entrantes, transformar los modelos en objetos JSON y definir los campos visibles para el cliente React. De esta forma, el serializer actúa como puente entre el modelo de datos del servidor y las estructuras consumidas por la interfaz.

### 3. Diagrama de Componentes

La arquitectura del sistema OralFlow se estructura bajo el modelo Cliente–Servidor y el patrón de diseño Model–View–Serializer (MVS), resultado de la adaptación del patrón MVT de Django al contexto de las APIs REST. Esta organización permite una clara separación de responsabilidades entre las capas de presentación, lógica de negocio y persistencia de datos, facilitando el mantenimiento, la escalabilidad y la integración con servicios externos.



- **Frontend (Cliente):** Desarrollado con React y ejecutado en el navegador web, es responsable de la interfaz gráfica y la experiencia del usuario. Desde esta capa, los pacientes, odontólogos y administradores realizan acciones como el registro, la gestión de citas o la consulta de fichas médicas. La comunicación con el backend se realiza mediante peticiones HTTP a la API REST, enviando y recibiendo datos en formato JSON.
- **Backend (Servidor):** Implementado con Django REST Framework, concentra la lógica de negocio del sistema.
  - Models: Definen la estructura de los datos y su persistencia en la base de datos PostgreSQL.
  - Views: Procesan las solicitudes del cliente, ejecutan las reglas de negocio y gestionan las respuestas.
  - Serializers: Validan los datos y los convierten a formato JSON para ser consumidos por el frontend.
  - URLs: Gestionan el enrutamiento de las solicitudes entrantes hacia las vistas correspondientes.
- **Base de Datos:** La información del sistema (usuarios, pacientes, odontólogos, citas, antecedentes, horarios y bloqueos) se almacena en PostgreSQL, un sistema de gestión de bases de datos relacional que garantiza consistencia, seguridad y rendimiento.
- **Aplicación Externa (Twilio):** Servicio de mensajería conectado al backend mediante peticiones HTTP. Permite el envío automatizado de notificaciones y recordatorios por WhatsApp, optimizando la comunicación con los pacientes.

#### 4. Justificaciones Generales de Decisiones Arquitectónicas

- **Arquitectura Cliente–Servidor:** Se eligió la arquitectura Cliente–Servidor por su capacidad de separar claramente las responsabilidades entre las capas de presentación y de procesamiento, lo que facilita el mantenimiento y la escalabilidad del sistema. Este modelo permite que múltiples usuarios interactúen de forma simultánea sin conflictos, mientras el servidor centralizado gestiona la lógica de negocio y la persistencia de los datos.
- **Patrón de Diseño MVT adaptado a APIs REST (MVS):** El backend se implementó con Django REST Framework, que utiliza el patrón Modelo–Vista–Template (MVT) adaptado a servicios REST, dando lugar a un esquema Modelo–Vista–Serializador (MVS). Este patrón proporciona una separación clara entre la lógica de negocio, el manejo de datos y la presentación de la información, permitiendo mantener una estructura modular y desacoplada del frontend. Los Serializers sustituyen a las plantillas tradicionales, transformando los modelos en formato JSON para el intercambio de información con el cliente React, asegurando validación de datos y consistencia entre las capas.