

# Prueba técnica: FullStack

—

TrueHome

## Visión general

Nuestro cliente, HomeTrue, requiere llevar un control de la agenda de sus vendedores; para ello, se le ocurrió guardar todas las actividades que realiza el **Vendedor** con sus **Clientes**

## Especificaciones

- Un **Vendedor** puede tener muchos **Clientes**
- El **Vendedor** solo le puede agendar actividades a sus **Clientes**
- Cada **Actividad** tiene que guardar
  - La fecha y hora de creación de la actividad
  - Fecha y hora en la que se agenda la actividad
  - Notas de la actividad (opcional)
  - El cliente asignado
  - El vendedor asignado
- El catálogo de actividades es:
  - Llamada al Cliente - Duración 30 minutos
  - Visita a Propiedad presencial - Duración 180 minutos
  - Visita Virtual a Propiedad - Duración 90 minutos.
- No se pueden agendar actividades en la misma fecha y hora.

## Actividades

Lo que se tiene que cumplir es lo siguiente:

1. Proponer un modelo de **Base de Datos Relacional** que refleje las necesidades del sistema.
2. Proponer e implementar un sistema de **Autenticación y Autorización**
3. Implementar una solución **CRUD** (Create, Read, Update, Delete) con Django Rest Framework para las actividades.
  - a. Solo se pueden mostrar las actividades del Vendedor logueado y solo las que tiene asignadas.
  - b. La lista de Actividades deberá filtrar por:
    - i. Tipo de actividad
    - ii. Rango de fechas (Solo con respecto a la fecha en la que se agendó)
  - c. La eliminación de la Actividad deberá ser con un borrado lógico. Proponer una forma.

- d. La lista de actividades deberá incluir el cliente asignado (ID y nombre completo), el tipo de actividad y el estatus de la actividad: **Cancelada** si se eliminó con un borrado lógico y **Activa** en caso contrario.

Ejemplo:


```
{
  "activities": [
    {
      "id": 1,
      "created_at": "1990-01-01 12:00:00", //Fecha de creación
      "schedule_at": "1990-01-01 12:00:00", // Fecha en que se debe
llevar a cabo la actividad
      "status": "active", //Actividad activa
      "type": "call",
      "client": {
        "id": 1,
        "name": "Pepito Pérez"
      }
    },
    {
      "id": 2,
      "created_at": "1990-01-01 12:00:00", //Fecha de creación
      "schedule_at": "1990-01-01 12:00:00", // Fecha en que se debe
llevar a cabo la actividad
      "status": "deactivated", //Actividad desactivada
      "type": "call",
      "client": {
        "id": 1,
        "name": "Pepito Pérez"
      }
    },
  ]
}
```

4. Desarrollar una aplicación web con React que consuma y demuestre la funcionalidad de tu API

- a. Debe estar desarrollada con **React**, preferentemente con componentes funcionales y aprovechando los hooks del ciclo de vida de los componentes. Procura organizar de manera clara y lógica los componentes creados.
  - b. Debe haber una vista de inicio/login, una principal donde se muestre el listado de clientes, y una de detalle donde se despliegan las actividades agendadas por cliente. Además de una manera de poder agendar, modificar y cancelar actividades.
  - c. El **diseño es libre**, puedes ocupar algún framework de CSS (Material, Bootstrap, TailwindUI, etc.) o clases propias. Puedes presentar las “pantallas” de tu aplicación como prefieras (múltiples páginas, modales, secciones ocultas). Lo que te facilite demostrar el manejo de las tecnologías y presentar la funcionalidad completa de tu API.
  - d. Debe **consultar tu API** por medio de fetch o axios para la creación, almacenamiento y consulta de las Actividades.
  - e. El catálogo de las actividades y las actividades creadas deberán **almacenarse en el store de Redux**.
  - f. Requiere de una pantalla de ingreso (login) y **hacer uso de la autenticación de tu API**, el manejo de las credenciales debe ser lo más seguro posible, dentro de las limitantes de una prueba.
  - g. El formulario de creación de actividad debe validar que estén llenos todos los campos.
  - h. En la vista de actividades puedes presentarlas como prefieras (listado, cards, carrusel, calendario) sólo **asegúrate de poder utilizar los filtros** creados en tu API. Puede ser lo más sencillo que se te ocurra, lo importante es demostrar el manejo de los filtros y la petición; recuerda que al pedir las actividades se debe actualizar el store de Redux.
5. Controlar las versiones con **Git**
- a. Tu código debe estar disponible en un repositorio de tu elección (gitlab, github, bitbucket, etc).

## Extra

1. Proponer un servicio que exponga información para un tablero (Dashboard). Por ejemplo: Conteo de actividades por Vendedor...
2. Pruebas unitarias
3. Generar un archivo Docker para la ejecución del sistema.
4. Pantallas extra de la aplicación React que otorguen mayor funcionalidad. (Vista de Alta de Clientes, CRUD de clientes, Registro de usuario, Vista de horarios ya ocupados por actividades, etc).

- 
5. Mejoras de usabilidad y UX, (marcar actividades retrasadas, ofrecer indicadores al usuario, alertas y animaciones, etc).
  6. Mensajes de error al intentar agendar actividades al mismo tiempo, y en fallos de la petición al API.
  7. Modificación del Reducer para optimizar peticiones al API, (no volver a hacer un *get* de las actividades después de hacer una creación, borrado, actualización)