Prueba Técnica para Programador Web Backend + API (3 horas)

Objetivo General:

Desarrollar un proyecto web básico en **Django + Django REST Framework** que permita crear un sistema de autenticación (login) para usuarios mediante una API RESTful, documentada y con un frontend simple que consuma dicha API.

Descripción del Proyecto

El postulante debe:

1. Crear un nuevo proyecto en Django.
2. Crear una aplicación llamada `usuarios`.
3. Crear un modelo `Usuario` si se desea extender `AbstractUser`, o usar el modelo por defecto.
4. Implementar una API REST que permita:
    - Registro de usuario (opcional).
    - Inicio de sesión (login) usando token de autenticación (JWT o token simple).
5. Documentar la API usando `drf-spectacular` o `drf-yasg` (Swagger/OpenAPI).
6. Crear una interfaz frontend sencilla (HTML + JavaScript) que permita al usuario **loguearse** y mostrar un mensaje de bienvenida si el login fue exitoso.

Requisitos Técnicos

🔹 **Modelos:**

Puedes usar el modelo `User` de Django (`django.contrib.auth.models.User`) o extenderlo.

🔹 **API REST:**

- Crear endpoints para:

    - `POST /api/login/` → retorna token de autenticación.
    - `GET /api/perfil/` → retorna los datos del usuario autenticado.
- Protege el endpoint `/api/perfil/` con autenticación.

🔹 **Documentación:**

- Implementar documentación automática (Swagger u OpenAPI) accesible en una ruta como `/api/docs/`.

🔹 **Frontend:**

- Crear un formulario de login (HTML + JS).
- Al hacer login, guardar el token (en `localStorage` o `sessionStorage`) y mostrar un mensaje tipo "¡Bienvenido [nombre de usuario]!" tras consumir el endpoint `/api/perfil/`.

🧾 Instrucciones de Entrega

1. Crear un nuevo proyecto Django.
2. Configurar base de datos PostgreSQL.
3. Crear la app `usuarios`.
4. Usar `Django REST Framework`.
5. Documentar en un archivo `README.md` los pasos para ejecutar el proyecto.
6. Asegurarse de que todo funcione en menos de 3 horas.

Criterios de Evaluación

Criterio	Descripción

✔️ API Funcional	Los endpoints deben funcionar correctamente.
✔️ Seguridad	El login debe estar protegido y los endpoints deben requerir token.
✔️ Documentación	Uso correcto de Swagger/OpenAPI.
✔️ Organización del Código	Código limpio, comentado y siguiendo buenas prácticas.
✔️ Frontend Funcional	El login debe funcionar y mostrar el mensaje de bienvenida con el token.
✔️ Uso correcto de herramientas Django	Incluye DRF, autenticación, manejo de rutas, etc.

🧪 Prueba Técnica para Desarrollador Frontend (Duración: 3 horas)

🎯 Objetivo General

Desarrollar una interfaz **frontend moderna y funcional** que consuma un conjunto de **APIs REST autenticadas con JWT**, permitiendo:

- Autenticación de usuario
- Visualización del perfil
- Edición completa del perfil
- Actualización de la foto de perfil

🌐 API a Consumir

Función	Método	URL	Notas
🔐 Login de usuario	`POST`	http://46.202.88.87:8010/usuarios/api/login/	Devuelve access y refresh token
📄 Obtener perfil	`GET`	http://46.202.88.87:8010/usuarios/api/perfil/	Requiere `access_token`
✏️ Editar perfil	`PUT`	http://46.202.88.87:8010/usuarios/api/usuario/perfil/	Requiere `access_token`
🖼️ Actualizar foto	`PATCH`	http://46.202.88.87:8010/usuarios/api/perfil/foto/	Requiere `access_token` + archivo `file`

🗒️ Credenciales de prueba

```
{
  "username": "carlosandresmoreno",
  "password": "90122856_Hanz"
}
```

📤 Datos esperados por la API

- 🔹 **Editar Perfil**

Envia un `PUT` con `Content-Type: application/json` y `Authorization: Bearer <access_token>`

```
{
  "user": {
    "first_name": "Carlos",
    "last_name": "Apellido"
  },
  "telefono": "1234",
  "tipo_usuario": "instructor",
  "tipo_naturaleza": "natural",
  "biografia": "dsdsds",
  "documento": "1234",
  "linkedin": "https://www.linkedin.com/in/carlos/",
  "twitter": "https://www.linkedin.com/in/carlos/",
  "github": "https://www.linkedin.com/in/carlos/",
  "sitio_web": "https://www.linkedin.com/in/carlos/",
  "esta_verificado": "false"
}
```

- **Subir foto de perfil**

Envia un `PATCH` con `Content-Type: multipart/form-data`:

```
PATCH /usuarios/api/perfil/foto/
Authorization: Bearer <access_token>
Content-Type: multipart/form-data
```

```
foto: archivo tipo file
```

Ejemplo con `FormData` en JavaScript:

```
const formData = new FormData();
formData.append("foto", archivo); // archivo = input.files[0]
fetch("http://46.202.88.87:8010/usuarios/api/perfil/foto/", {
  method: "POST",
  headers: {
    Authorization: "Bearer " + access_token
  },
  body: formData
});
```

🛠️ Requisitos Técnicos

🔹 **1. Login**

- Formulario de acceso.
- Solicita el `access_token` y `refresh_token`.
- Almacena el `access_token` en `localStorage`.

🔹 **2. Mostrar perfil**

- Solicita el perfil usando el token y muestra:
    - Nombre
    - Apellido
    - Correo

- Foto (si está disponible)
- Biografía, redes sociales, tipo de usuario, etc.

🔹 **3. Editar perfil**

- Formulario editable con los campos JSON descritos.
- Envío de cambios y mensaje de éxito/error.
- Validar respuesta estructurada:

```
{
  "status": "success",
  "data": { ... },
  "message": "Perfil actualizado correctamente"
}
```

🔹 **4. Subir foto**

- Permitir seleccionar archivo y subirlo.
- Mostrar foto actualizada tras la carga.
- Mostrar mensajes de éxito/error.

💻 Requisitos Visuales

- Diseño moderno, claro y funcional.
- Interfaz adaptable (responsive).
- Puedes usar Bootstrap, Tailwind, Material UI u otra librería visual.

✅ Criterios de Evaluación

Criterio	Evaluado
🔐 Uso correcto de JWT	En todas las solicitudes protegidas
🧩 Flujo lógico	Login → perfil → editar/subir foto
🎨 UI	Interfaz clara, amigable y funcional
📋 Mensajes y estados	Éxito/error basados en `status` y `message`
🧼 Código limpio	Bien estructurado, comentado y reutilizable

📦 Entrega

- Repositorio en GitHub.
- Instrucciones claras en `README.md` para probar.
- Puedes incluir un `live demo` si lo deseas (opcional).