

Login

bash

CopyEdit

POST <http://localhost:8000/api/usuarios/login/>

Content-Type: application/json

```
{
  "correo": "correo@example.com",
  "password": "tu_clave"
}
```

Registro Estudiante / Admin

bash

CopyEdit

POST <http://localhost:8000/api/usuarios/registro/>

```
{
  "nombres": "Joel",
  "apellidos": "Plazarte",
  "cedula": "1752274173",
  "correo": "oscar.plazarte20@gmail.com",
  "password": "root",
  "id_rol": 1,
  "carreras": []
}
```

Registro Docente (requiere campos extra)

bash

CopyEdit

POST <http://localhost:8000/api/usuarios/registro/>

```
{
  "nombres": "Luddcía",
  "apellidos": "Moreno",
  "cedula": "0987254321",
  "correo": "luciadd@example.com",
  "password": "abcd",
  "id_rol": 1,
  "modalidad_contratacion": "Tiempo completo",
  "tiempo_dedicacion": "40 horas",
  "carreras": []
}
```

=====

Estructura Fronted

```
src/
├── api/                # Llamadas a la API organizadas por entidad
│   ├── auth.ts        # Login, logout
│   ├── usuarios.ts    # Registro, edición (solo Admin)
│   └── horarios.ts    # Consultas de horarios por rol
├── assets/            # Imágenes, logos, fuentes, etc.
│   └── logo.png
├── auth/              # Lógica de autenticación y roles
│   ├── AuthContext.tsx
│   ├── useAuth.ts     # Custom hook
│   └── ProtectedRoute.tsx # Rutas protegidas
├── components/        # Componentes compartidos y reutilizables
│   ├── Header.tsx
│   ├── Sidebar.tsx
│   └── RoleSelector.tsx # (si se usa en alguna parte)
├── layouts/           # Layouts base por rol
│   ├── EstudianteLayout.tsx
│   ├── DocenteLayout.tsx
│   ├── AdminLayout.tsx
│   └── SuperAdminLayout.tsx
├── pages/             # Vistas principales organizadas por rol
│   ├── login/
│   │   └── LoginPage.tsx
│   ├── estudiante/
│   │   ├── Dashboard.tsx
│   │   └── VerHorario.tsx
│   ├── docente/
│   │   ├── Dashboard.tsx
│   │   ├── VerHorario.tsx
│   │   └── EditarNombre.tsx
│   ├── admin/
│   │   ├── Dashboard.tsx
│   │   ├── RegistrarUsuario.tsx
│   │   └── GestionUsuarios.tsx
│   └── superadmin/
│       ├── Dashboard.tsx
│       └── ConfiguracionGlobal.tsx # (Ej. gestión de semestres o similares)
├── routes/            # Definición centralizada de rutas
│   └── AppRouter.tsx
├── styles/            # Estilos globales y modulares
│   ├── globals.css
│   ├── login.css
│   ├── dashboard.css
│   ├── header.css
│   └── ...otros.css
```

```
|— types/          # Interfaces y enums compartidos
|   |— usuario.ts
|— utils/         # Funciones auxiliares
|   |— authHelpers.ts  # manejo de tokens en localStorage, redirección por rol
|   |— validators.ts   # Validaciones de formulario
|— App.tsx
|— main.tsx
```

=====

✅ ¿Qué debe poder hacer un Super Admin?

Según tu modelo, el **Super Admin** es como el "root developer" o gestor principal de la plataforma. Te detallo **todo lo que debe poder hacer** dividido por bloques funcionales:

USUARIOS

Acciones:

- Crear cualquier tipo de usuario (**estudiante**, **docente**, **admin**, **superadmin**)
- Consultar todos los usuarios registrados
- Ver roles de usuarios
- Eliminar o editar usuarios
- Asociar o reasociar carreras a usuarios (**usuario_carrera**)
- Asignar materias a estudiantes (**usuario_asignatura_estudiante**)
- Asignar materias a docentes (**usuario_asignatura**)
- Ver información docente (**docente**)

FACULTADES Y CARRERAS

Acciones:

- CRUD de **facultad**
 - CRUD de **carrera**
 - Consultar carreras por facultad
-

ASIGNATURAS Y SU RELACIÓN CON CARRERAS

Acciones:

- CRUD de **asignatura**
 - Asignar asignaturas a carreras (**asignatura_carrera**)
 - Marcar si son básicas o comunes
 - Consultar materias por carrera
-

AULAS

Acciones:

- CRUD de **aula**
 - Consultar aulas por facultad, piso, tipo, uso general
-

HORAS DE CLASE

Acciones:

- CRUD de **hora_clase**
 - Definir qué bloques están disponibles (día, hora_inicio, hora_fin)
-

HORARIOS

Acciones:

- CRUD de `horario`
 - Ver horarios de cualquier usuario
 - Ver colisiones por aula, hora o usuario
 - Consultar horarios por carrera
-

SEGURIDAD Y ACCESO

Acciones:

- Login con JWT
 - Rutas protegidas por token
 - Rutas protegidas por permiso de `rol.superadmin`
-

VISTAS SUGERIDAS PARA LA API

Aquí una lista sugerida de endpoints solo para Super Admin:

Endpoint	Método	Función
<code>/api/usuarios/</code>	<code>GET</code>	Listar todos los usuarios
<code>/api/usuarios/<id>/</code>	<code>GET/PUT/DELETE</code>	Ver, editar o eliminar un usuario
<code>/api/roles/</code>	<code>GET</code>	Ver todos los roles
<code>/api/facultades/</code>	<code>CRUD</code>	Gestionar facultades
<code>/api/carreras/</code>	<code>CRUD</code>	Gestionar carreras
<code>/api/asignaturas/</code>	<code>CRUD</code>	Gestionar asignaturas
<code>/api/asignatura-carrera/</code>	<code>CRUD</code>	Relacionar materias con carreras

<code>/api/aulas/</code>	CRUD	Gestionar aulas
<code>/api/horas-clase/</code>	CRUD	Gestionar bloques horarios
<code>/api/horarios/</code>	CRUD	Crear y consultar horarios
<code>/api/usuarios/<id>/asignaturas/</code>	GET/POST	Ver/asignar materias a un usuario
<code>/api/usuarios/<id>/carreras/</code>	GET/POST	Ver/asignar carreras a un usuario

✓ Protección por rol (SuperAdmin)

Crearemos un permiso tipo:

```
python
CopyEdit
class IsSuperAdmin(BasePermission):
    def has_permission(self, request, view):
        return request.user and request.user.id_rol.id_rol == 4
```

Y lo aplicarás así en cada vista:

```
python
CopyEdit
@permission_classes([IsAuthenticated, IsSuperAdmin])
```

=====

✓ Listo para probar app FACULTADES

Ahora puedes probar:

Método	URL	Descripción
GET	<code>/api/superadmin/facultades/</code>	Listar todas las facultades
POST	<code>/api/superadmin/facultades/</code>	Crear una nueva facultad

PUT /api/superadmin/facultades Editar una facultad
 /{id}/

DELETE /api/superadmin/facultades Eliminar una facultad
 /{id}/

✓ JSONs para usar el endpoint **/api/superadmin/facultades/**

POST (crear una facultad)

pgsql

CopyEdit

POST /api/superadmin/facultades/

Authorization: Bearer <access_token>

Content-Type: application/json

```
{  
  "nombre": "Facultad de Ingeniería"  
}
```

PUT (actualizar una facultad)

pgsql

CopyEdit

PUT /api/superadmin/facultades/3/

Authorization: Bearer <access_token>

Content-Type: application/json

```
{  
  "id_facultad": 3,  
  "nombre": "Facultad de Ingeniería y Tecnología"  
}
```

 Recuerda que el ID debe coincidir con el que estás actualizando.

DELETE (eliminar una facultad)

sql

CopyEdit

```
DELETE /api/superadmin/facultades/3/  
Authorization: Bearer <access_token>
```

! No necesitas enviar JSON en DELETE. Solo asegúrate de que el ID esté en la URL y el token esté en el header.

=====

JSON para pruebas app Carreras

POST

json

CopyEdit

```
{  
  "nombre": "Ingeniería de Sistemas",  
  "codigo": "IS001",  
  "id_facultad": 1  
}
```

PUT

json

CopyEdit

```
{  
  "id_carrera": 5,  
  "nombre": "Ingeniería en Software",  
  "codigo": "IS002",  
  "id_facultad": 1  
}
```

DELETE

swift

CopyEdit

```
DELETE /api/superadmin/carreras/5/
```


✓ 1. ¿Campo **paralelo** de **usuario_asignatura**?

Es necesario **mantener el campo paralelo en usuario_asignatura** porque:

- En el sistema, los docentes dictan **una misma asignatura en varios paralelos**.
- La tabla **horario** indica **en qué momento específico se dicta cada paralelo**, pero no reemplaza la relación entre docente y asignatura/paralelo.
- **usuario_asignatura** representa **qué asignatura y paralelo dicta un docente**, independientemente del horario asignado.

📌 Entonces:

- **usuario_asignatura**: Dice “el docente A dicta la materia X, paralelo 1”.
- **horario**: Dice “el paralelo 1 de la materia X, dictada por el docente A, se dicta el lunes a las 10h”.

➡ Ambos **paralelo** tienen sentido, pero **en diferentes contextos**.

=====

=====

=====

📖 1. ¿Cuál es el propósito del módulo **horario**?

El módulo **horario** se encargará de **almacenar y validar la asignación de clases reales**, tomando en cuenta:

- Qué docente dicta qué materia.
- A qué paralelo pertenece.
- En qué aula se dicta.
- En qué hora clase se dicta.
- En qué día de la semana.

- Y para qué semestre lectivo aplica.
-

2. ¿Qué validaciones debe manejar?

Aquí es donde entra lo complejo. Las validaciones del **horario** deben asegurar:

No se pueden asignar dos horarios:

- A un mismo docente en la misma hora y día.
 - A una misma aula en la misma hora y día.
 - A un mismo grupo de estudiantes en la misma hora y día.
 - A una misma asignatura con el mismo paralelo más de una vez en el mismo horario.
-

3. ¿Qué relaciones se cruzan?

Para poder crear un **horario**, necesitas haber definido previamente:

- ✓ **usuario_asignatura** (docente → asignatura).
 - ✓ **usuario_asignatura_estudiante** (estudiante → asignatura).
 - ✓ **hora_clase** (día y bloques de tiempo).
 - ✓ **aula** (capacidad y tipo).
 - ✓ **asignatura** (duración en horas, si es común).
 - ✓ **paralelo** (repetido por asignatura para múltiples grupos).
-

4. ¿Qué campos incluye la tabla **horario**?

La tabla es así:

sql

CopyEdit

```
CREATE TABLE public.horario (  
    id_horario serial PRIMARY KEY,  
    id_hora_clase integer NOT NULL,  
    id_usuario integer NOT NULL,  
    id_asignatura integer NOT NULL,  
    id_aula integer NOT NULL,  
    paralelo integer NOT NULL,  
    semestre_lectivo character varying(50) NOT NULL  
);
```

Traducción:

- `id_usuario`: el docente.
- `id_asignatura`: la materia.
- `id_hora_clase`: el bloque horario.
- `id_aula`: el aula asignada.
- `paralelo`: el grupo.
- `semestre_lectivo`: por ejemplo, "2025-A".

5. ¿Cómo funcionará el proceso de creación?

Paso 1 – El usuario SuperAdmin elige:

- Asignatura a agendar.
- Docente encargado.
- Aula disponible.
- HoraClase disponible.
- Paralelo.

- Semestre lectivo.

Paso 2 – El backend validará:

- Que el docente no tenga otra clase a esa hora.
- Que el aula esté libre.
- Que la asignatura ya esté asociada al docente.
- Que los estudiantes de ese paralelo no tengan otra clase a esa hora.

Paso 3 – Si todo está bien, se guarda el horario.

6. ¿Cómo lo implementaremos?

Lo haremos por fases:

♦ **Fase 3.1 – CRUD básico**

- Crear **horario** solo si no hay conflictos.

♦ **Fase 3.2 – Validación avanzada**

- Detectar y bloquear conflictos en:
 - Aula.
 - Docente.
 - Estudiantes.

♦ **Fase 3.3 – Visualización**

- Endpoints personalizados para:
 - Ver el horario por usuario (docente o estudiante).
 - Ver disponibilidad de aulas.
 - Ver horarios por paralelo.

¿Qué representa la tabla `horario`?

Cada fila en la tabla `horario` representa una clase específica de una asignatura en un día, hora y aula determinados, impartida por un docente a un paralelo específico.

Cosas clave que debemos tomar en cuenta

Validaciones de integridad

1. Un docente no puede estar en dos clases al mismo tiempo.
2. Un aula no puede estar ocupada por más de una clase en la misma hora.
3. Un estudiante no puede estar inscrito en dos clases al mismo tiempo.
4. La asignatura debe haber sido previamente asignada al docente (en `usuario_asignatura`).
5. El aula debe tener suficiente capacidad (verificado al insertar estudiantes en `usuario_asignatura_estudiante`).
6. Si la asignatura es común (`es_comun = true`), solo puede dictarse en aulas con `uso_general = true`.