README_TEST - Validación manual de APIs (Guía paso a paso)

1. Introducción

Este documento explica cómo **probar manualmente** los endpoints del backend Universidad Digital. Una "API validada" significa que **responde correctamente**, **respeta roles/ownership**, y **retorna códigos HTTP esperados**. Alcance: solo pruebas manuales (Swagger UI, Postman, cURL).

2. Herramientas recomendadas

- **Swagger UI (`/docs`)**: explorar endpoints, schemas y ejecutar requests rápidas.
- **Postman**: pruebas repetibles con entornos y colecciones.
- **cURL**: validaciones rápidas desde consola.

3. Orden obligatorio de pruebas (sin errores por dependencias)

0. **Crear o verificar un usuario Administrador inicial (precondición)**
1. Preparar el entorno y levantar el backend
2. Login y uso del token
3. CRUD de usuarios (para crear Docente y Estudiante)
4. CRUD de materias
5. CRUD de periodos academicos
6. Inscripciones
7. Calificaciones
8. Validación de roles y ownership
9. Errores comunes
10. Checklist final

> Importante: **no intentes crear usuarios por API sin un Admin**.
> El endpoint `/users` requiere rol Administrador.

4. Paso 0 - Usuario Administrador inicial (precondición)

Objetivo: crear en la base de datos un usuario Administrador para poder autenticarte y crear otros usuarios.

Precondición técnica:

- La base de datos y las tablas deben existir.
- Si es la primera vez, levanta el backend una vez para que se creen las tablas automáticamente.

SQL en PostgreSQL (ejecutar en tu cliente SQL):

```
-- 1) Habilitar pgcrypto (para generar hash bcrypt)
CREATE EXTENSION IF NOT EXISTS pgcrypto;

-- 2) Asegurar roles base (Administrador, Docente, Estudiante)
INSERT INTO roles (name, description)
SELECT 'Administrador', 'Rol administrador'
WHERE NOT EXISTS (SELECT 1 FROM roles WHERE name = 'Administrador');
```

```
INSERT INTO roles (name, description)
SELECT 'Docente', 'Rol docente'
WHERE NOT EXISTS (SELECT 1 FROM roles WHERE name = 'Docente');

INSERT INTO roles (name, description)
SELECT 'Estudiante', 'Rol estudiante'
WHERE NOT EXISTS (SELECT 1 FROM roles WHERE name = 'Estudiante');

-- 3) Crear usuario administrador si no existe
INSERT INTO users (email, full_name, hashed_password, is_active)
SELECT 'admin@ud.edu', 'Admin', crypt('AdminPass1234', gen_salt('bf')), true
WHERE NOT EXISTS (SELECT 1 FROM users WHERE email = 'admin@ud.edu');

-- 4) Asignar rol Administrador al usuario
INSERT INTO user_roles (user_id, role_id)
SELECT u.id, r.id
FROM users u, roles r
WHERE u.email = 'admin@ud.edu' AND r.name = 'Administrador'
  AND NOT EXISTS (
    SELECT 1 FROM user_roles ur
    WHERE ur.user_id = u.id AND ur.role_id = r.id
  );
```

Consulta rápida de roles y sus IDs (para usar en `role_ids`):

```
SELECT id, name FROM roles ORDER BY id;
```

Usuario Administrador esperado:

```
email: admin@ud.edu
password: AdminPass1234
```

Confirmación del paso 0:

- El usuario existe en la tabla `users`.
- El usuario tiene el rol Administrador en `user_roles`.
- Podrás iniciar sesión con ese usuario en el Paso 2.

5. Paso 1 - Preparación del entorno

Objetivo: levantar el backend y tener la URL base.

Variables mínimas en `.env` (prefijo `APP_`):

```
APP_ENV=development
APP_DATABASE_URL=postgresql+psycopg://user:pass@localhost:5432/universidad
```

```
APP_JWT_SECRET=change_me
APP_CORS_ORIGINS=["http://localhost:3000"]
```

Levantar backend:

```
uvicorn app.main:app --reload
```

URL base:

```
http://127.0.0.1:8000
```

Confirmación del paso 1:

- El servidor responde en `http://127.0.0.1:8000/docs`.
- Si al hacer login aparece `APP_JWT_SECRET no configurado`, revisa que el `.env` tenga `APP_JWT_SECRET` y **reinicia** el backend.

6. Paso 2 - Autenticación (Login y uso del token)

6.1 Login

- **Endpoint**: `POST /auth/login`
- **Payload**:

```
{
  "email": "admin@ud.edu",
  "password": "AdminPass1234"
}
```

- **Response 200**:

```
{
  "access_token": "jwt...",
  "token_type": "bearer"
}
```

- El token también se entrega como **cookie HttpOnly**.

Confirmación del paso 2.1:

- Recibes un `access_token` en la respuesta.

6.2 Uso del token

Header recomendado:

```
Authorization: Bearer <access_token>
```

Errores:

- **401**: token inválido o no enviado.
- **403**: token válido sin permisos.

Confirmación del paso 2.2:

- Puedes acceder a `/auth/me` con el token y recibes el usuario.

6.3 Me

- URL: `/auth/me`
- Método: `GET`
- Roles: autenticado
- Headers: `Authorization: Bearer <token>`
- 200:

```
{"id":1,"email":"admin@ud.edu","full_name":"Admin","is_active":true,"created_at":"
...","roles":["Administrador"]}
```

7. Paso 3 - Usuarios (crear Docente y Estudiante)

> Debes usar **token de Administrador** en todos estos pasos.

Crear usuario

- URL: `/users`
- Método: `POST`
- Roles: Administrador
- Headers: `Authorization`
- Body:

```
{"email":"docente@ud.edu","full_name":"Docente
Uno","password":"DocentePass1234","role_ids":[2]}
```

- 201:

```
{"id":2,"email":"docente@ud.edu","full_name":"Docente
Uno","is_active":true,"created_at":"...","roles":["Docente"]}
```

- 409 si email duplicado

Confirmación del paso 3.1:

- Recibes el `id` del Docente creado.

Listar usuarios

- URL: `/users`
- Método: `GET`
- Roles: Administrador
- Headers: `Authorization`
- 200: lista de usuarios

Confirmación del paso 3.2:

- Ves al usuario Docente en la lista.

Actualizar usuario

- URL: `/users/{id}`
- Método: `PUT`
- Roles: Administrador
- Headers: `Authorization`
- Body:

```
{"full_name":"Docente Uno A","is_active":true}
```

- 200: usuario actualizado

Confirmación del paso 3.3:

- El `full_name` cambia en la respuesta.

Cambiar estado (desactivar)

- URL: `/users/{id}`
- Método: `DELETE`
- Roles: Administrador
- Headers: `Authorization`
- 200: usuario con `is_active=false`

Confirmación del paso 3.4:

- El usuario queda con `is_active=false`.

8. Paso 4 - Materias

Crear materia

- URL: `/subjects`

- Método: `POST`
- Roles: Administrador
- Headers: `Authorization`
- Body:

```
{"code":"MAT101","name":"Cálculo I","credits":4}
```

- 201: materia creada

Confirmación del paso 4.1:

- Recibes el `id` de la materia creada.

Listar materias

- URL: `/subjects`
- Método: `GET`
- Roles: Administrador, Docente, Estudiante
- Headers: `Authorization`
- 200: lista de materias

Confirmación del paso 4.2:

- Ves la materia creada en la lista.

Actualizar materia

- URL: `/subjects/{id}`
- Método: `PUT`
- Roles: Administrador
- Headers: `Authorization`
- Body:

```
{"name":"Cálculo I (actualizado)","credits":5}
```

- 200: materia actualizada

Confirmación del paso 4.3:

- El nombre y créditos cambian en la respuesta.

Desactivar materia

- URL: `/subjects/{id}`
- Método: `DELETE`
- Roles: Administrador
- Headers: `Authorization`
- 200: materia con `is_active=false`

Confirmación del paso 4.4:

- La materia queda con `is_active=false`.

9. Paso 5 - Periodos academicos

Crear periodo

- URL: `/periods`
- Método: `POST`
- Roles: Administrador
- Headers: `Authorization`
- Body:

```
{"code":"2026-1","name":"Periodo 2026-1","start_date":"2026-01-
01","end_date":"2026-06-30"}
```

- 201: periodo creado

Confirmación del paso 5.1:

- Recibes el `id` del periodo creado.

Listar periodos

- URL: `/periods`
- Método: `GET`
- Roles: Administrador, Docente, Estudiante
- Headers: `Authorization`
- 200: lista de periodos

Confirmación del paso 5.2:

- Ves el periodo creado en la lista.

Consultar periodo

- URL: `/periods/{id}`
- Método: `GET`
- Roles: Administrador, Docente, Estudiante
- Headers: `Authorization`
- 200: periodo encontrado

Confirmación del paso 5.3:

- Los datos del periodo coinciden con lo creado.

Actualizar periodo

- URL: `/periods/{id}`
- Método: `PUT`

- Roles: Administrador
- Headers: `Authorization`
- Body:

```
{"name":"Periodo 2026-1 (actualizado)","end_date":"2026-07-15"}
```

- 200: periodo actualizado

Confirmación del paso 5.4:

- El nombre o fechas cambian en la respuesta.

Desactivar periodo

- URL: `/periods/{id}`
- Método: `DELETE`
- Roles: Administrador
- Headers: `Authorization`
- 200: periodo con `is_active=false`

Confirmación del paso 5.5:

- El periodo queda con `is_active=false`.

10. Paso 6 - Inscripciones

> Necesitas un `period_id` valido del Paso 5.

Inscribir estudiante

- URL: `/enrollments`
- Método: `POST`
- Roles: Administrador, Estudiante
- Headers: `Authorization`
- Body:

```
{"user_id":3,"subject_id":1,"period_id":1}
```

- 201: inscripción creada
- 409 si el estudiante intenta inscribir a otro usuario

Confirmación del paso 6.1:

- Recibes el `id` de la inscripción creada.

Consultar inscripciones

- URL: `/enrollments`
- Método: `GET`

- Roles: Administrador, Docente, Estudiante
- Headers: `Authorization`
- 200: lista (estudiante solo ve las suyas)

Confirmación del paso 6.2:

- El estudiante solo ve sus propias inscripciones.

Cancelar inscripción

- URL: `/enrollments/{id}`
- Método: `DELETE`
- Roles: Administrador
- Headers: `Authorization`
- 200: inscripción con `is_active=false`

Confirmación del paso 6.3:

- La inscripción queda con `is_active=false`.

11. Paso 7 - Calificaciones

Registrar calificación

- URL: `/grades`
- Método: `POST`
- Roles: Administrador, Docente
- Headers: `Authorization`
- Body:

```
{"enrollment_id":1,"value":95.5,"notes":"Parcial 1"}
```

- 201: calificación creada

Confirmación del paso 7.1:

- Recibes el `id` de la calificación creada.

Consultar calificaciones

- URL: `/grades`
- Método: `GET`
- Roles: Administrador, Docente, Estudiante
- Headers: `Authorization`
- 200: lista (estudiante solo ve las suyas)

Confirmación del paso 7.2:

- El estudiante solo ve sus propias calificaciones.

Actualizar calificación

- URL: `/grades/{id}`
- Método: `PUT`
- Roles: Administrador, Docente
- Headers: `Authorization`
- Body:

```
{"value":98.0,"notes":"Actualización"}
```

- 200: calificación actualizada

Confirmación del paso 7.3:

- El valor y notas cambian en la respuesta.

12. Paso 8 - Validación de roles y permisos

Acceso permitido (Administrador crea materia)

```
POST /subjects
Authorization: Bearer <admin_token>
```

200/201 esperado.

Acceso denegado por rol (Estudiante crea materia)

```
POST /subjects
Authorization: Bearer <student_token>
```

403 esperado:

```
{"detail":"Permisos insuficientes."}
```

Acceso denegado por ownership (Estudiante consulta inscripción ajena)

```
GET /enrollments/999
Authorization: Bearer <student_token>
```

409 esperado:

```
{"detail":"Acceso no permitido."}
```

Confirmación del paso 8:

- Obtienes 200/201 cuando corresponde y 403/409 cuando corresponde.

13. Paso 9 - Validación de errores comunes

- **400**: conflictos de negocio (ej. contraseña corta para docente/admin).
- **401**: no autenticado o token inválido.
- **403**: rol sin permisos.
- **404**: recurso inexistente.
- **422**: validación Pydantic (campos faltantes o tipos inválidos).

Confirmación del paso 9:

- Cada error devuelve el código HTTP esperado.

14. Paso 10 - Uso de Swagger / OpenAPI

- Acceso: `http://127.0.0.1:8000/docs`
- Probar endpoints directamente con “Try it out”.
- Los schemas reflejan validaciones Pydantic.
- Respuestas y códigos están visibles por endpoint.

Confirmación del paso 10:

- Puedes ejecutar un request desde Swagger y ver la respuesta.

15. Checklist final de validación

- ☐ Login funciona
- ☐ Token JWT válido en headers
- ☐ CRUD de usuarios probado
- ☐ CRUD de materias probado
- ☐ CRUD de periodos probado
- ☐ Inscripciones con ownership
- ☐ Calificaciones con ownership
- ☐ Roles respetados
- ☐ Respuestas claras y seguras
- ☐ No se exponen datos sensibles

16. Conclusión

Si todos los ítems del checklist se cumplen, las APIs están listas para consumo manual por frontend y validación funcional básica.