



Institut TIC
de Barcelona

Generalitat de Catalunya
Departament d'Educació
Institut TIC Barcelona

DAM 2º. GS PROJECTE
SPRINT-3
Jairo Vigil Casellas

SPRINT 3 - Creación de la API REST para la gestión de la base de datos relacional

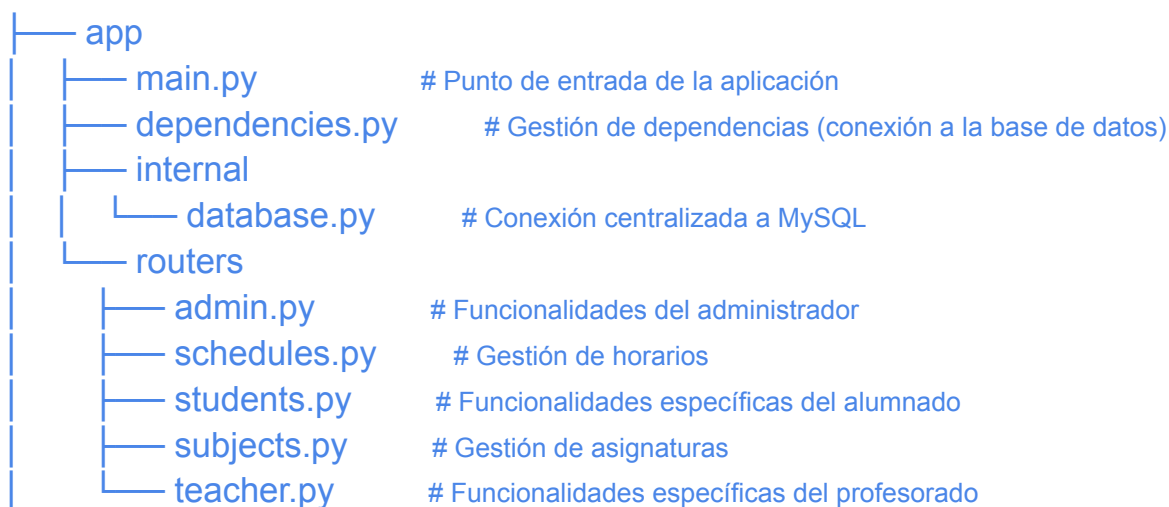
Introducción

En este sprint, he desarrollado una API REST que cumple con los requerimientos del proyecto, centrada en la gestión de usuarios, asignaturas, horarios y asistencia. He organizado la API de manera modular, siguiendo prácticas de desarrollo estructuradas y ayudándome de la documentación del Moodle. Para ello, he utilizado herramientas como FastAPI y mysql-connector, asegurando que la API sea eficiente, escalable y fácil de mantener, idóneo para futuros sprints.

Este sprint ha sido un reto, ya que no solo he trabajado en las funcionalidades principales, sino que también he aprendido a estructurar el código para que sea claro y reutilizable, lo que me llevó varias horas entre planificación, implementación y pruebas.

Estructura del proyecto

He organizado el proyecto con una estructura que separa las responsabilidades en diferentes módulos. Esto facilita la comprensión y el mantenimiento del código. La estructura es la siguiente:



He diseñado la estructura así porque me permite mantener el código organizado, con cada archivo enfocado en una funcionalidad concreta.



Desglose del código

Conexión a la base de datos ([database.py](#))

He implementado la clase Connection que centraliza la conexión a la base de datos MySQL. La clase utiliza varios métodos estáticos para abrir y cerrar conexiones.

```
from mysql import connector

class Connection:
    _connection = None

    @staticmethod
    def connect():
        Connection._connection = connector.connect(
            host="localhost",
            user="admin",
            password="password",
            database="schoolControl"
        )


    @staticmethod
    def get():
        if not Connection._connection:
            Connection.connect()
        return Connection._connection

    @staticmethod
    def close():
        if Connection._connection:
            Connection._connection.close()
            Connection._connection = None
```

connect: Establece la conexión con la base de datos.

get: Devuelve una conexión existente o crea una nueva si no la hay.

close: Cierra la conexión, asegurando que no haya fugas de recursos.

 Institut TIC de Barcelona	Generalitat de Catalunya Departament d'Educació Institut TIC Barcelona	DAM 2º. GS PROJECTE SPRINT-3 Jairo Vigil Casellas
---	---	---

Gestión de dependencias ([dependencies.py](#))

Para cada solicitud, he creado una dependencia que abre y cierra la conexión a la base de datos de manera segura.

```
from app.internal.database import Connection
```

```
def get_db():
    db = Connection.get()
    try:
        yield db
    finally:
        Connection.close()
```

Esto asegura que cada solicitud tenga su propia conexión y que esta se cierre correctamente después de usarse.

Rutas principales

[admin.py](#): Contiene las rutas para gestionar asignaturas y horarios. Incluye manejo de errores mediante try-except.

```
@router.get("/subjects")
def get_all_subjects(db = Depends(get_db)):
    try:
        cursor = db.cursor(dictionary=True)
        cursor.execute("SELECT * FROM Asignatura")
        subjects = cursor.fetchall()
        return {"ok": True, "subjects": subjects}
    except Exception as e:
        return {"ok": False, "error": str(e)}
```

Este endpoint permite a los administradores obtener todas las asignaturas almacenadas en la base de datos.

[teacher.py](#): Implementa las rutas específicas para profesores, como consultar sus horarios. Aquí también he añadido manejo de errores para robustecer la API.

```
@router.get("/{teacher_id}/schedules")
def get_teacher_schedules(teacher_id: int, db = Depends(get_db)):
    try:
```



```
cursor = db.cursor(dictionary=True)
query = """
    SELECT h.* FROM Horari h
    INNER JOIN Assignatura a ON h.id_assignatura = a.id
    WHERE a.id_usuari = %s
    """
cursor.execute(query, (teacher_id,))
schedules = cursor.fetchall()
return {"ok": True, "schedules": schedules}
except Exception as e:
    return {"ok": False, "error": str(e)}
```

Este endpoint consulta los horarios asignados a un profesor específico.

students.py: Aquí he implementado las rutas que permiten a los estudiantes consultar su información personal y las asignaturas en las que están matriculados.

```
@router.get("/{student_id}/subjects")
def get_student_subjects(student_id: int, db = Depends(get_db)):
    try:
        cursor = db.cursor(dictionary=True)
        query = """
            SELECT a.* FROM Assignatura a
            INNER JOIN UserSubject us ON us.subjectID = a.id
            WHERE us.userID = %s
            """
        cursor.execute(query, (student_id,))
        subjects = cursor.fetchall()
        return {"ok": True, "subjects": subjects}
    except Exception as e:
        return {"ok": False, "error": str(e)}
```

Cumplimiento de los requerimientos

Con esta API, cumplo con los siguientes requerimientos del proyecto:



Institut TIC
de Barcelona

Generalitat de Catalunya
Departament d'Educació
Institut TIC Barcelona

DAM 2º. GS PROJECTE
SPRINT-3
Jairo Vigil Casellas

- Los administradores pueden gestionar asignaturas y horarios.
- Los profesores pueden consultar sus horarios asignados.
- Los alumnos pueden consultar tanto su información personal como las asignaturas en las que están matriculados.
- La API es robusta y modular, con manejo de errores y separación clara de responsabilidades.

Webgrafía

- ❖ **Teoría del profesor Toni Talens:**
 - [Introducción a FastAPI](#)
 - [Base de datos y conexiones en FastAPI](#)
- ❖ **Documentación oficial:**
 - [FastAPI](#)
 - [mysql-connector](#)
- ❖ **Otros recursos:**
 - [Estructuración de proyectos en FastAPI](#)
 - [Gestión de errores en FastAPI](#)

hablar de los getter, setter, update & delete - swagger ui

poner logs de la api rest y pruebas

pensar en un problema que he tenido, alternativas y solución