TUNIS BUSINESS SCHOOL
UNIVERSITY OF TUNIS

IPT معهد باستور تونس
Institut Pasteur de Tunis

Vaccination and Awareness API

for the Institute of Pasteur in Tunisia

Web Services

Written by:
Maha Jdidi

Abstract

Access to vaccination and preventive healthcare remains a significant challenge for many citizens in Tunisia. People often lack reliable information about the importance of vaccines, the risks of infectious diseases, and the preventive measures they should take, whether for general health, travel abroad, or protection against common contagious illnesses.

This project proposes the development of a Vaccination and Awareness API, designed to simplify the vaccination process and provide educational resources to the public. The platform will allow citizens to schedule vaccination appointments efficiently and access accurate information about recommended vaccines, their schedules, and preventive measures for a variety of diseases.

By combining vaccination services and awareness content in one accessible platform, the API acts as a practical tool and an educational resource, helping citizens take proactive steps to protect their health. Ultimately, this project aims to increase vaccination coverage, improve public health literacy, and contribute to a safer and healthier society in Tunisia.

Contents

Chapter 1

Introduction

1.1 Problem Statement

In Tunisia, vaccination services play a critical role in protecting public health, particularly through institutions such as the Institut Pasteur. However, the current vaccination appointment process often relies on manual systems, which can lead to long waiting times, poor appointment organization, and limited visibility of available vaccines. Citizens may also lack access to reliable information about vaccines, increasing confusion and vaccine hesitancy. Also, managing appointments and vaccine availability becomes challenging for healthcare institutions without a digital platform. These issues highlight the need for system that simplifies appointment booking and improves access to trustworthy vaccination information.

1.2 Scope

This project aims to develop a RESTful API for the Institut Pasteur in Tunisia to support vaccination appointment management and public awareness. The API enables users to view available vaccines, consult awareness articles, and book vaccination appointments based on their availability and chosen vaccine. The project focuses on improving accessibility, reducing waiting times, and enhancing information transparency. It does not include medical diagnosis or payment processing.

1.3 Outcomes/Objectives

At the end of this project, the following outcomes are expected:

- **Centralized Vaccination Management:** An API that organizes vaccination appointments at the Institut Pasteur in Tunisia.
- **Easy Appointment Scheduling:** A system that allows users to book vaccination appointments based on their availability and preferred vaccine.
- **Vaccine Information Access:** A platform that provides clear and updated information about available vaccines.
- **Public Awareness Enhancement:**A section designed to raise awareness about vaccination through educational articles.

1.4 System Analysis and Specifications

1. **Related Systems**: Although there is no identical API currently deployed for vaccination appointment management at the Institut Pasteur in Tunisia, similar digital health platforms exist for medical appointment scheduling and public health information. These systems typically provide basic appointment booking and service listings but often lack vaccine-specific selection and localized awareness content. This project addresses these limitations.
2. **Target Audience**: The target audience of this API includes Tunisian citizens who wish to book vaccination appointments at the Institut Pasteur. It also targets healthcare staff and administrators responsible for managing vaccines, appointment schedules, and awareness content. In addition, software developers can use the API to build web or mobile applications that interact with the vaccination system.
3. **Required Inputs and Expected Outputs**: The required inputs of the system include user personal information, selected vaccine type, preferred appointment date and time, and authentication credentials. Administrative inputs include vaccine details, availability schedules, and awareness article content. The expected outputs include appointment confirmations, lists of available vaccines, and awareness articles. All outputs are delivered in a structured JSON format through RESTful API responses.

Chapter 2

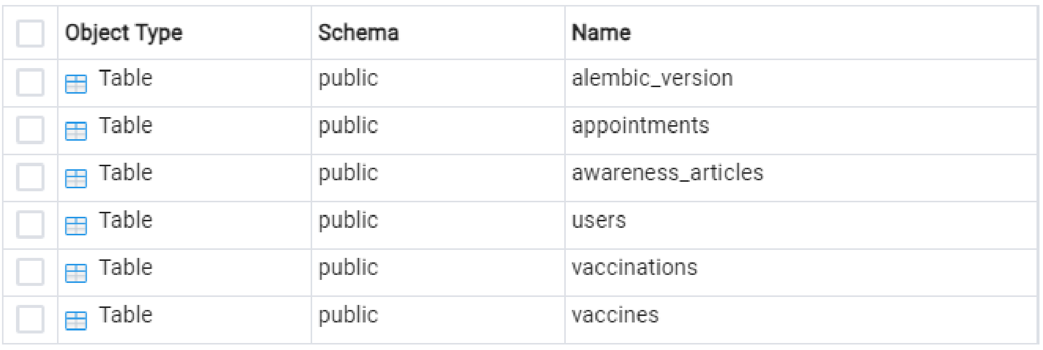
Work Explanation

this section will describe the architecture of the project and the different implementations and technologies used in it.

2.1 Database Management

2.1.1 Database Structure

This database schema diagram was generated using pgAdmin, **a popular open-source management tool for PostgreSQL that provides a user-friendly interface for designing, managing, and interacting with PostgreSQL databases (1).**



	Object Type	Schema	Name
☐	Table	public	alembic_version
☐	Table	public	appointments
☐	Table	public	awareness_articles
☐	Table	public	users
☐	Table	public	vaccinations
☐	Table	public	vaccines

Figure 2.1: Database Schema Diagram

2.1.2 Database Management System: Postgres

PostgreSQL is an open-source, powerful, and highly extensible relational database management system known for its robustness, support for advanced data types, and SQL compliance (2) . I have used PostgreSQL to organize the necessary data into tables, enabling efficient storage, access, and manipulation.

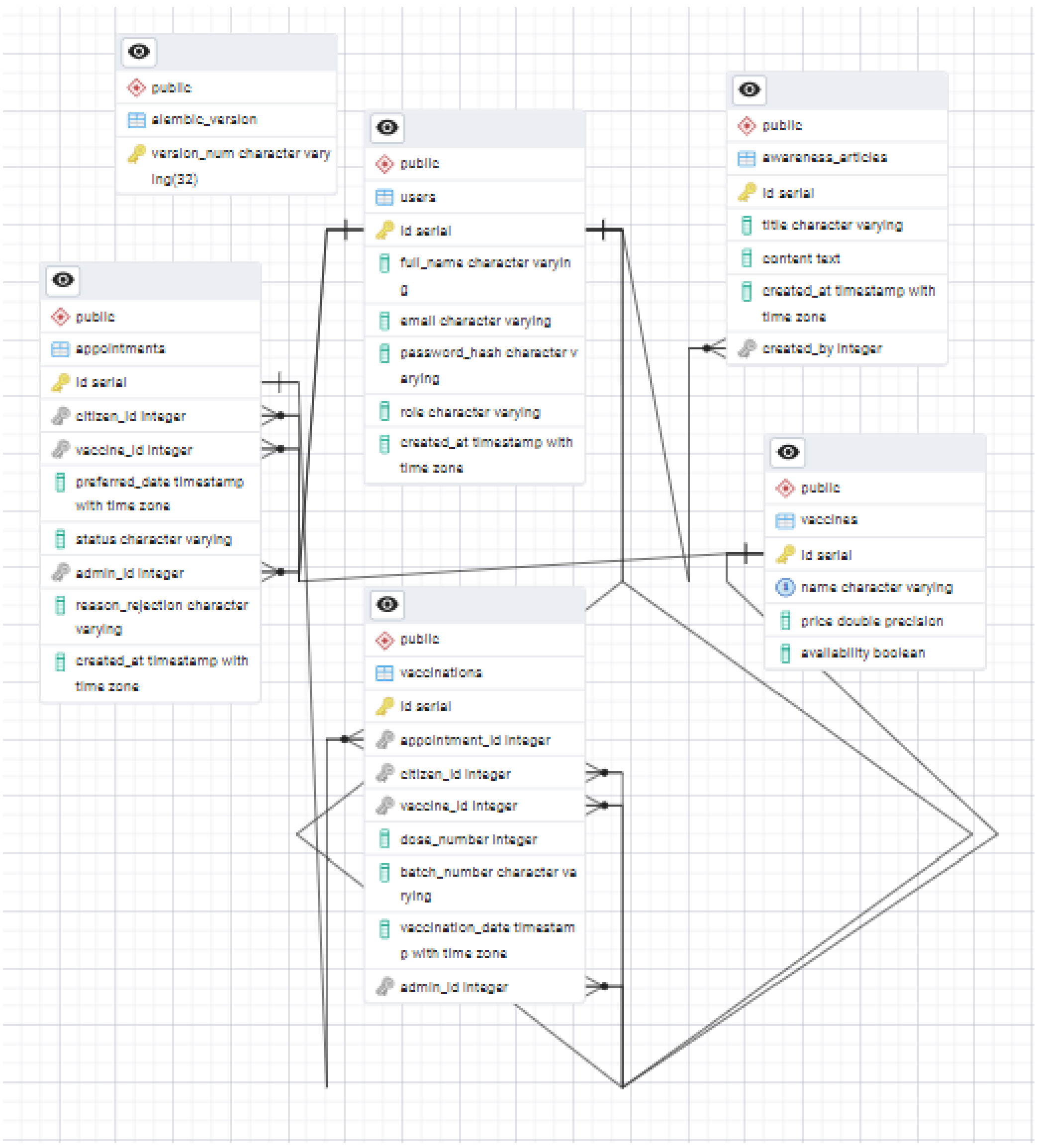


Figure 2.2: Entity-Relationship Diagram (ERD) of the Database

2.1.3 Database Migrations with Alembic and FastAPI

Alembic is used alongside FastAPI to manage database migrations, **allowing for seamless schema changes throughout the development process. The migration workflow ensures that modifications to the database schema are properly tracked and applied to maintain consistency across environments(3).** Below is a diagram showcasing the Alembic migration process, illustrating how schema changes are generated, applied, and tracked.

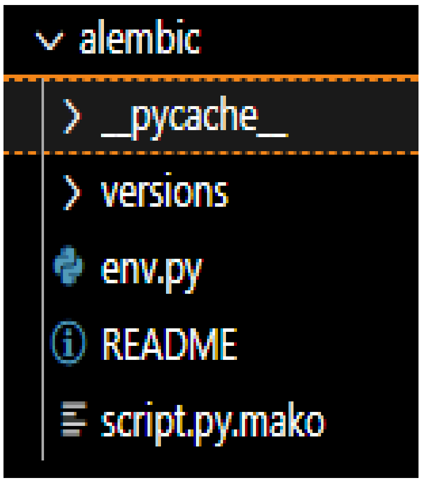


Figure 2.3: Alembic Migration Workflow

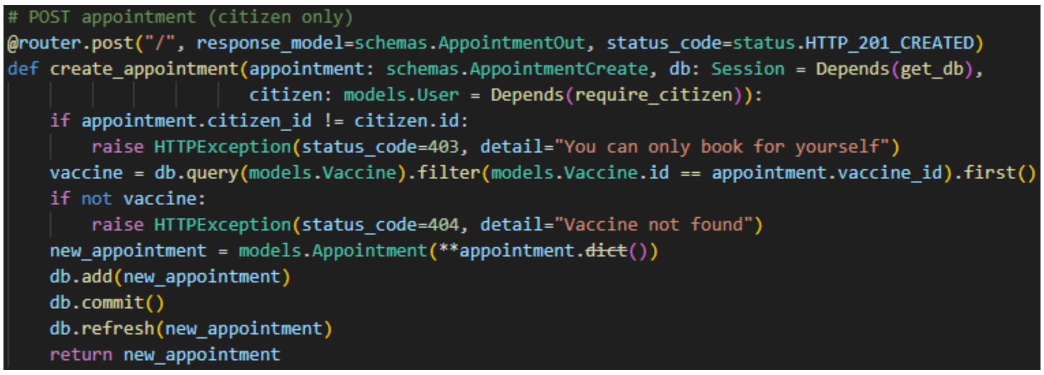
2.2 Python / FastAPI Contribution

This section highlights the contributions made using Python and FastAPI to develop the backend of the vaccination API . **FastAPI is used for creating efficient and scalable APIs, while Python serves as the foundation for handling database interactions, user authentication, and other essential functionalities (4).**

2.2.1 Appointments Requests

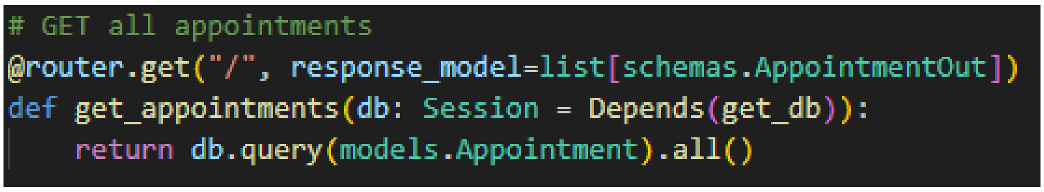
Details on the FastAPI endpoints and logic used for managing appointments in the platform.

* Create appointment by citizen:



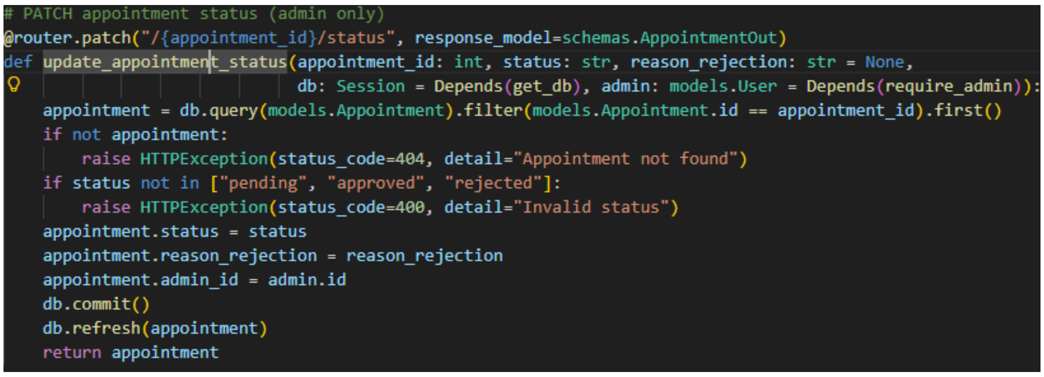
```python
# POST appointment (citizen only)
@router.post("/", response_model=schemas.AppointmentOut, status_code=status.HTTP_201_CREATED)
def create_appointment(appointment: schemas.AppointmentCreate, db: Session = Depends(get_db),
                       citizen: models.User = Depends(require_citizen)):
    if appointment.citizen_id != citizen.id:
        raise HTTPException(status_code=403, detail="You can only book for yourself")
    vaccine = db.query(models.Vaccine).filter(models.Vaccine.id == appointment.vaccine_id).first()
    if not vaccine:
        raise HTTPException(status_code=404, detail="Vaccine not found")
    new_appointment = models.Appointment(**appointment.~~dict~~())
    db.add(new_appointment)
    db.commit()
    db.refresh(new_appointment)
    return new_appointment
```

* Get all appointments data:



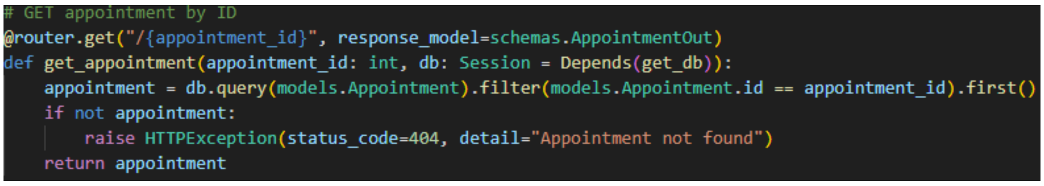
```python
# GET all appointments
@router.get("/", response_model=list[schemas.AppointmentOut])
def get_appointments(db: Session = Depends(get_db)):
    return db.query(models.Appointment).all()
```

* Approval or rejection of appointment by admin:



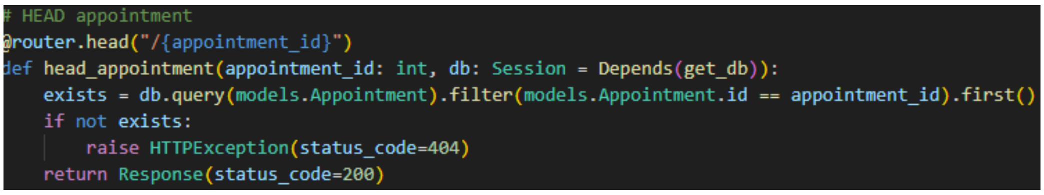
```python
# PATCH appointment status (admin only)
@router.patch("/{appointment_id}/status", response_model=schemas.AppointmentOut)
def update_appointment_status(appointment_id: int, status: str, reason_rejection: str = None,
                              db: Session = Depends(get_db), admin: models.User = Depends(require_admin)):
    appointment = db.query(models.Appointment).filter(models.Appointment.id == appointment_id).first()
    if not appointment:
        raise HTTPException(status_code=404, detail="Appointment not found")
    if status not in ["pending", "approved", "rejected"]:
        raise HTTPException(status_code=400, detail="Invalid status")
    appointment.status = status
    appointment.reason_rejection = reason_rejection
    appointment.admin_id = admin.id
    db.commit()
    db.refresh(appointment)
    return appointment
```

* Get specific appointment by id:



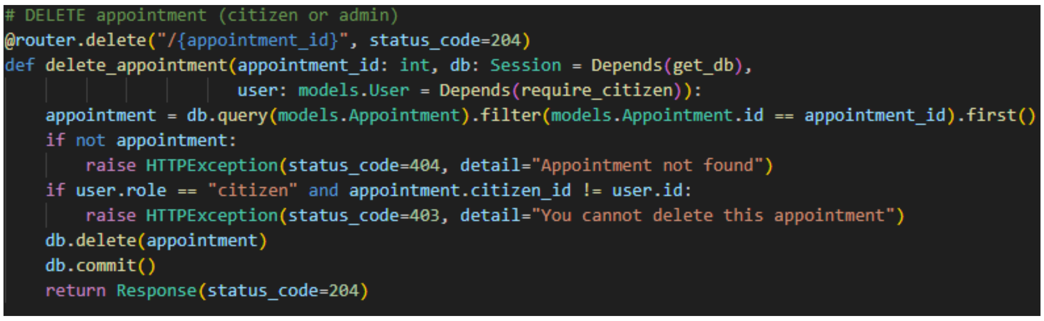
```python
# GET appointment by ID
@router.get("/{appointment_id}", response_model=schemas.AppointmentOut)
def get_appointment(appointment_id: int, db: Session = Depends(get_db)):
    appointment = db.query(models.Appointment).filter(models.Appointment.id == appointment_id).first()
    if not appointment:
        raise HTTPException(status_code=404, detail="Appointment not found")
    return appointment
```

* Head appointment:



```python
# HEAD appointment
@router.head("/{appointment_id}")
def head_appointment(appointment_id: int, db: Session = Depends(get_db)):
    exists = db.query(models.Appointment).filter(models.Appointment.id == appointment_id).first()
    if not exists:
        raise HTTPException(status_code=404)
    return Response(status_code=200)
```

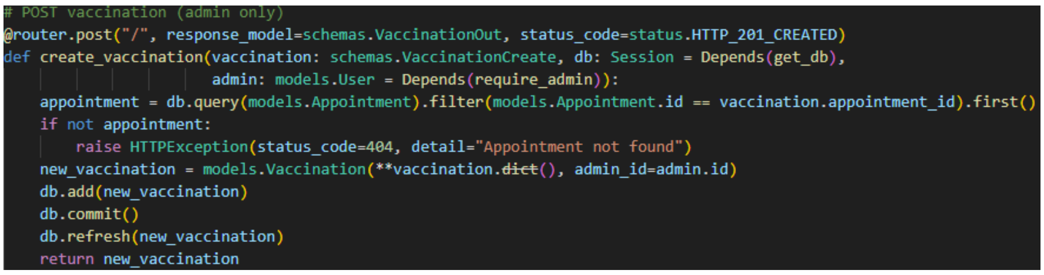
* Delete appointment:



```python
# DELETE appointment (citizen or admin)
@router.delete("/{appointment_id}", status_code=204)
def delete_appointment(appointment_id: int, db: Session = Depends(get_db),
                       user: models.User = Depends(require_citizen)):
    appointment = db.query(models.Appointment).filter(models.Appointment.id == appointment_id).first()
    if not appointment:
        raise HTTPException(status_code=404, detail="Appointment not found")
    if user.role == "citizen" and appointment.citizen_id != user.id:
        raise HTTPException(status_code=403, detail="You cannot delete this appointment")
    db.delete(appointment)
    db.commit()
    return Response(status_code=204)
```

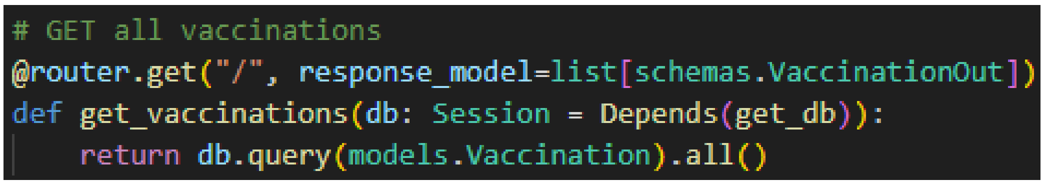
2.2.2 Vaccinations Requests

Explanation of the user Vaccinations endpoints. * Create Vaccinations by admin only:



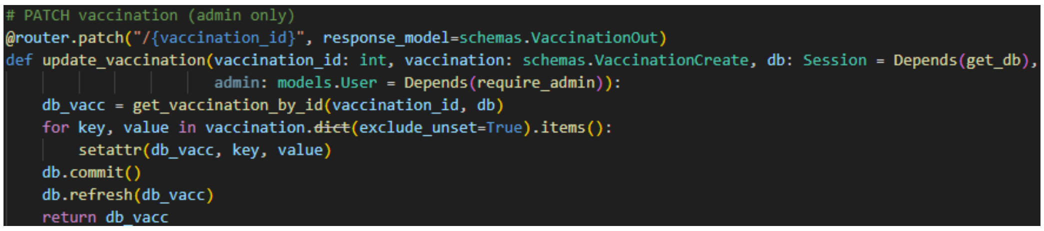
```python
# POST vaccination (admin only)
@router.post("/", response_model=schemas.VaccinationOut, status_code=status.HTTP_201_CREATED)
def create_vaccination(vaccination: schemas.VaccinationCreate, db: Session = Depends(get_db),
                       admin: models.User = Depends(require_admin)):
    appointment = db.query(models.Appointment).filter(models.Appointment.id == vaccination.appointment_id).first()
    if not appointment:
        raise HTTPException(status_code=404, detail="Appointment not found")
    new_vaccination = models.Vaccination(**vaccination.~~dict~~(), admin_id=admin.id)
    db.add(new_vaccination)
    db.commit()
    db.refresh(new_vaccination)
    return new_vaccination
```

* Get all Vaccinations data:



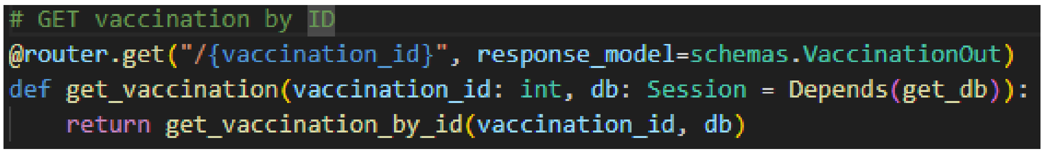
```python
# GET all vaccinations
@router.get("/", response_model=list[schemas.VaccinationOut])
def get_vaccinations(db: Session = Depends(get_db)):
    return db.query(models.Vaccination).all()
```

* Update of Vaccinations by admin:



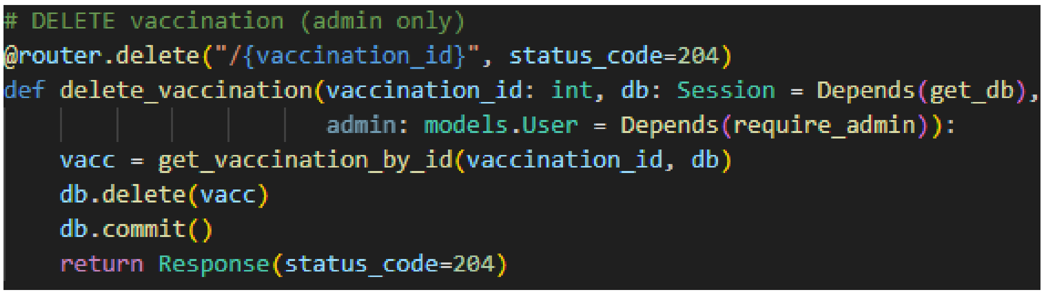
```python
# PATCH vaccination (admin only)
@router.patch("/{vaccination_id}", response_model=schemas.VaccinationOut)
def update_vaccination(vaccination_id: int, vaccination: schemas.VaccinationCreate, db: Session = Depends(get_db),
                       admin: models.User = Depends(require_admin)):
    db_vacc = get_vaccination_by_id(vaccination_id, db)
    for key, value in vaccination.~~dict~~(exclude_unset=True).items():
        setattr(db_vacc, key, value)
    db.commit()
    db.refresh(db_vacc)
    return db_vacc
```

* Get specific Vaccinations by id:



```python
# GET vaccination by ID
@router.get("/{vaccination_id}", response_model=schemas.VaccinationOut)
def get_vaccination(vaccination_id: int, db: Session = Depends(get_db)):
    return get_vaccination_by_id(vaccination_id, db)
```

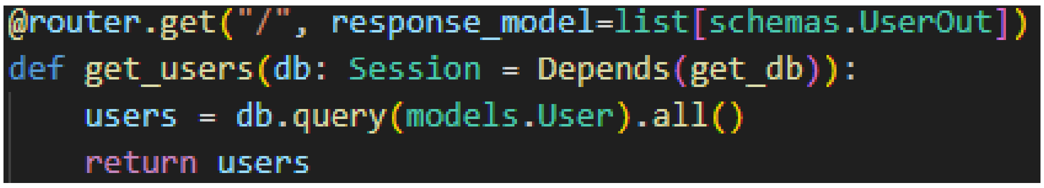
* Delete Vaccination:



```python
# DELETE vaccination (admin only)
@router.delete("/{vaccination_id}", status_code=204)
def delete_vaccination(vaccination_id: int, db: Session = Depends(get_db),
                       admin: models.User = Depends(require_admin)):
    vacc = get_vaccination_by_id(vaccination_id, db)
    db.delete(vacc)
    db.commit()
    return Response(status_code=204)
```

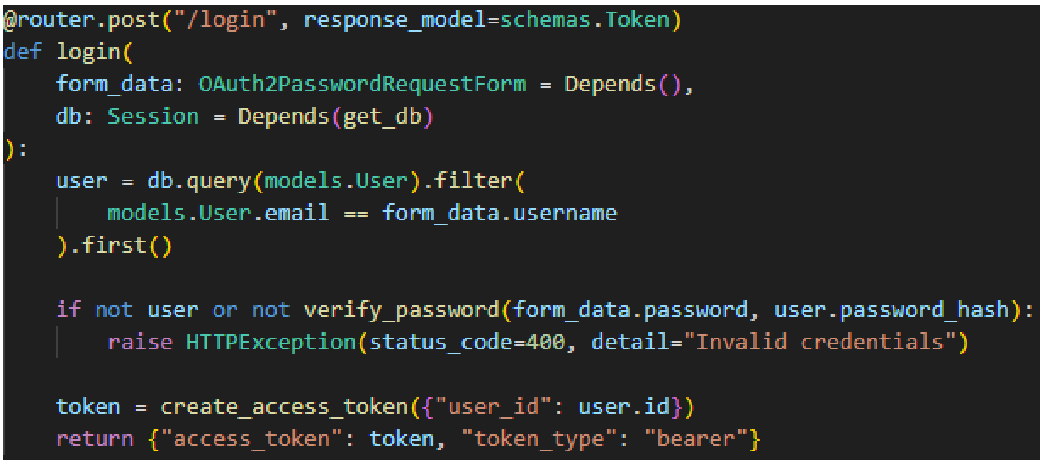
2.2.3 User Requests

Details on the FastAPI endpoints for the user. * Get all users:



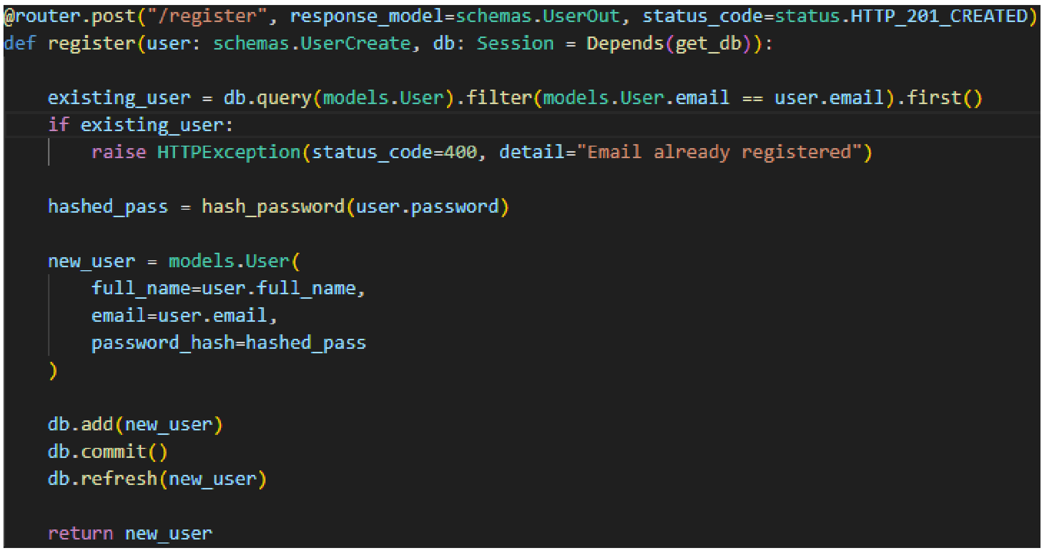
```python
@router.get("/", response_model=list[schemas.UserOut])
def get_users(db: Session = Depends(get_db)):
    users = db.query(models.User).all()
    return users
```

* Login:



```python
@router.post("/login", response_model=schemas.Token)
def login(
    form_data: OAuth2PasswordRequestForm = Depends(),
    db: Session = Depends(get_db)
):
    user = db.query(models.User).filter(
        models.User.email == form_data.username
    ).first()

    if not user or not verify_password(form_data.password, user.password_hash):
        raise HTTPException(status_code=400, detail="Invalid credentials")

    token = create_access_token({"user_id": user.id})
    return {"access_token": token, "token_type": "bearer"}
```

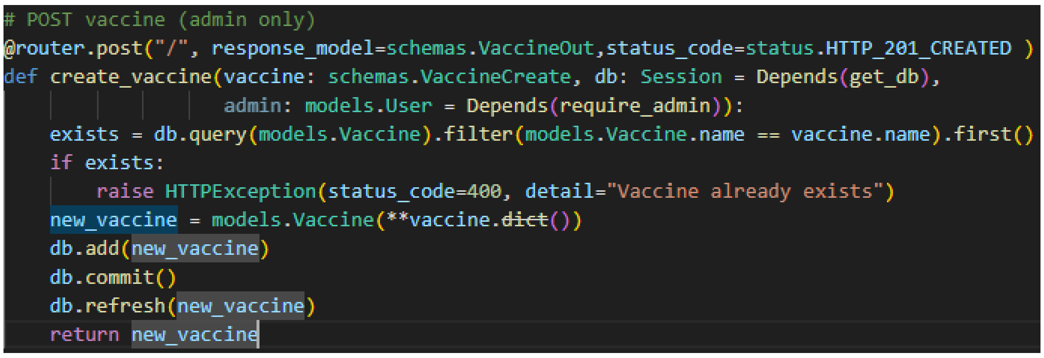
* Register:



```python
@router.post("/register", response_model=schemas.UserOut, status_code=status.HTTP_201_CREATED)
def register(user: schemas.UserCreate, db: Session = Depends(get_db)):

    existing_user = db.query(models.User).filter(models.User.email == user.email).first()
    if existing_user:
        raise HTTPException(status_code=400, detail="Email already registered")

    hashed_pass = hash_password(user.password)

    new_user = models.User(
        full_name=user.full_name,
        email=user.email,
        password_hash=hashed_pass
    )

    db.add(new_user)
    db.commit()
    db.refresh(new_user)

    return new_user
```

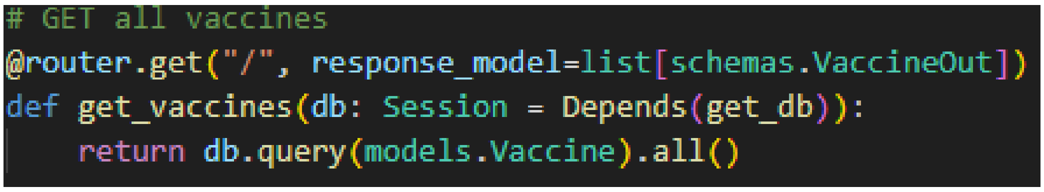
2.2.4 Vaccines Requests

Description of the API endpoints that manage vaccines. * Create vaccine by admin:



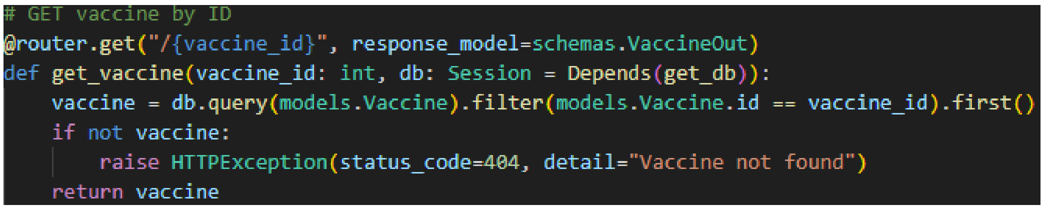
```python
# POST vaccine (admin only)
@router.post("/", response_model=schemas.VaccineOut,status_code=status.HTTP_201_CREATED )
def create_vaccine(vaccine: schemas.VaccineCreate, db: Session = Depends(get_db),
                   admin: models.User = Depends(require_admin)):
    exists = db.query(models.Vaccine).filter(models.Vaccine.name == vaccine.name).first()
    if exists:
        raise HTTPException(status_code=400, detail="Vaccine already exists")
    new_vaccine = models.Vaccine(**vaccine.~~dict~~())
    db.add(new_vaccine)
    db.commit()
    db.refresh(new_vaccine)
    return new_vaccine
```

* Get all vaccines data:



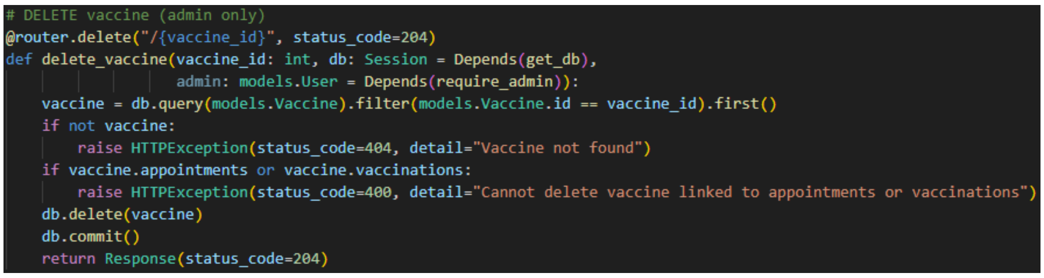
```python
# GET all vaccines
@router.get("/", response_model=list[schemas.VaccineOut])
def get_vaccines(db: Session = Depends(get_db)):
    return db.query(models.Vaccine).all()
```

* Get vaccine by id:



```python
# GET vaccine by ID
@router.get("/{vaccine_id}", response_model=schemas.VaccineOut)
def get_vaccine(vaccine_id: int, db: Session = Depends(get_db)):
    vaccine = db.query(models.Vaccine).filter(models.Vaccine.id == vaccine_id).first()
    if not vaccine:
        raise HTTPException(status_code=404, detail="Vaccine not found")
    return vaccine
```

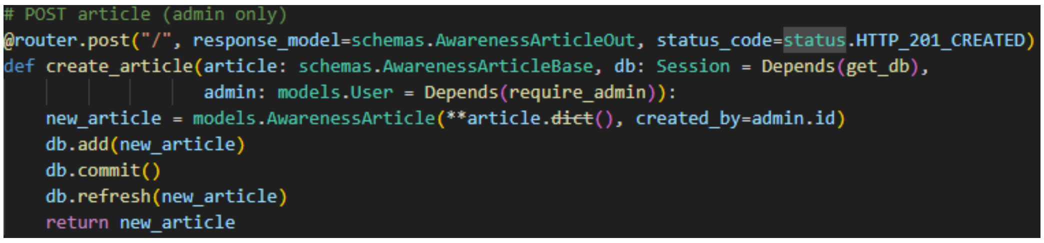
* Delete vaccine:



```python
# DELETE vaccine (admin only)
@router.delete("/{vaccine_id}", status_code=204)
def delete_vaccine(vaccine_id: int, db: Session = Depends(get_db),
                   admin: models.User = Depends(require_admin)):
    vaccine = db.query(models.Vaccine).filter(models.Vaccine.id == vaccine_id).first()
    if not vaccine:
        raise HTTPException(status_code=404, detail="Vaccine not found")
    if vaccine.appointments or vaccine.vaccinations:
        raise HTTPException(status_code=400, detail="Cannot delete vaccine linked to appointments or vaccinations")
    db.delete(vaccine)
    db.commit()
    return Response(status_code=204)
```

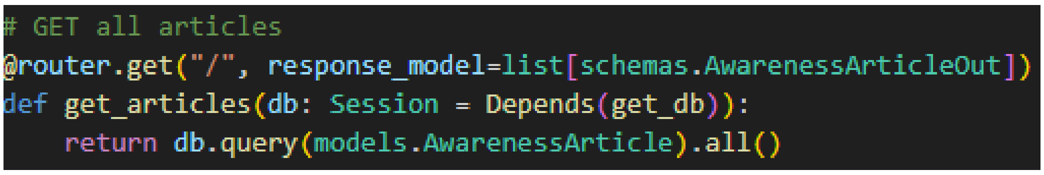
2.2.5 Awareness Article Requests

Overview of the API endpoints. * Create Awareness Article by admin:



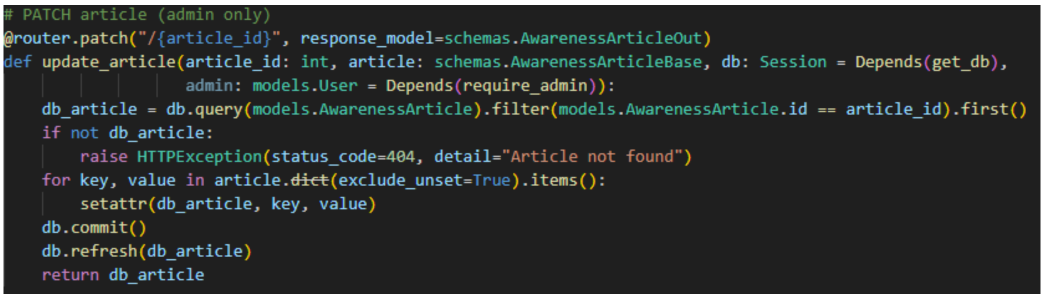
```
# POST article (admin only)
@router.post("/", response_model=schemas.AwarenessArticleOut, status_code=status.HTTP_201_CREATED)
def create_article(article: schemas.AwarenessArticleBase, db: Session = Depends(get_db),
                   admin: models.User = Depends(require_admin)):
    new_article = models.AwarenessArticle(**article.~~dict~~(), created_by=admin.id)
    db.add(new_article)
    db.commit()
    db.refresh(new_article)
    return new_article
```

* Get all Awareness Article data:



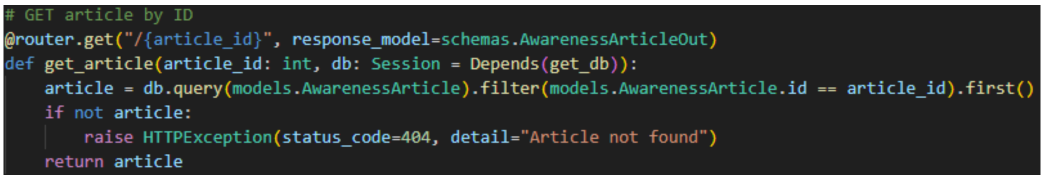
```
# GET all articles
@router.get("/", response_model=list[schemas.AwarenessArticleOut])
def get_articles(db: Session = Depends(get_db)):
    return db.query(models.AwarenessArticle).all()
```

* Update Awareness Article by admin:



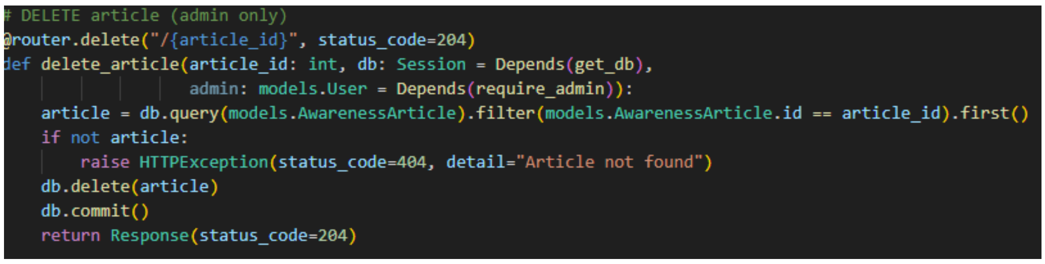
```
# PATCH article (admin only)
@router.patch("/{article_id}", response_model=schemas.AwarenessArticleOut)
def update_article(article_id: int, article: schemas.AwarenessArticleBase, db: Session = Depends(get_db),
                   admin: models.User = Depends(require_admin)):
    db_article = db.query(models.AwarenessArticle).filter(models.AwarenessArticle.id == article_id).first()
    if not db_article:
        raise HTTPException(status_code=404, detail="Article not found")
    for key, value in article.~~dict~~(exclude_unset=True).items():
        setattr(db_article, key, value)
    db.commit()
    db.refresh(db_article)
    return db_article
```

* Get Awareness Article by id:



```
# GET article by ID
@router.get("/{article_id}", response_model=schemas.AwarenessArticleOut)
def get_article(article_id: int, db: Session = Depends(get_db)):
    article = db.query(models.AwarenessArticle).filter(models.AwarenessArticle.id == article_id).first()
    if not article:
        raise HTTPException(status_code=404, detail="Article not found")
    return article
```

* Delete Awareness Article:



```python
# DELETE article (admin only)
@router.delete("/{article_id}", status_code=204)
def delete_article(article_id: int, db: Session = Depends(get_db),
                   admin: models.User = Depends(require_admin)):
    article = db.query(models.AwarenessArticle).filter(models.AwarenessArticle.id == article_id).first()
    if not article:
        raise HTTPException(status_code=404, detail="Article not found")
    db.delete(article)
    db.commit()
    return Response(status_code=204)
```

2.3 Use Case Diagram

A Use Case Diagram in Unified Modeling Language (UML) is a visual representation that illustrates the interactions between users (actors) and a system (5) . Below you can see the UCD of this API and it explains how it works.

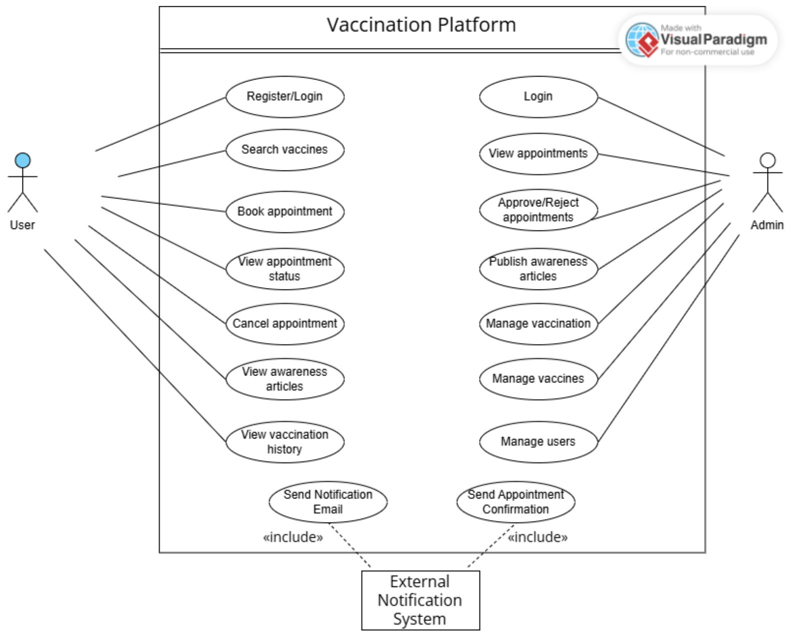
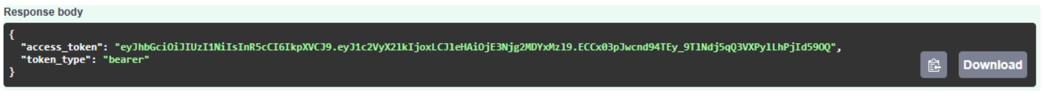


Figure 2.4: Vaccination Platform UCD

2.4 User Authentication with JWT

JSON Web Token (JWT) is an open standard (RFC 7519) that

defines a compact and self-contained way for securely transmitting information between parties as a JSON object (6). I used JWT to secure the API by authenticating users with hashed credentials, generating a signed token containing the user ID, and validating this token on each request to protect access to secured endpoints.

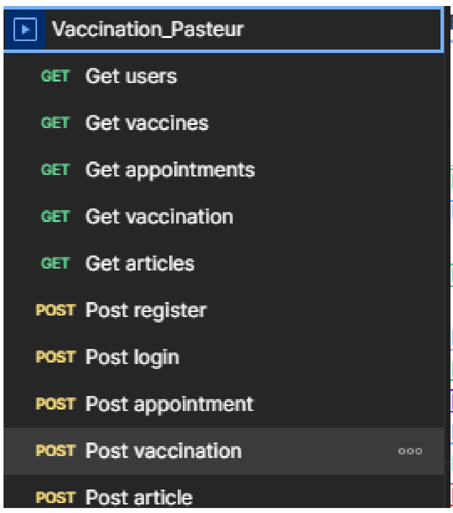


Response body

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX21kIjoxLCJleHAiOjE3Njg2MDYxMzl9.ECCx03pJwcnd94TEy_9T1Ndj5qQ3VXPy1LhPjId590Q",
  "token_type": "bearer"
}
```

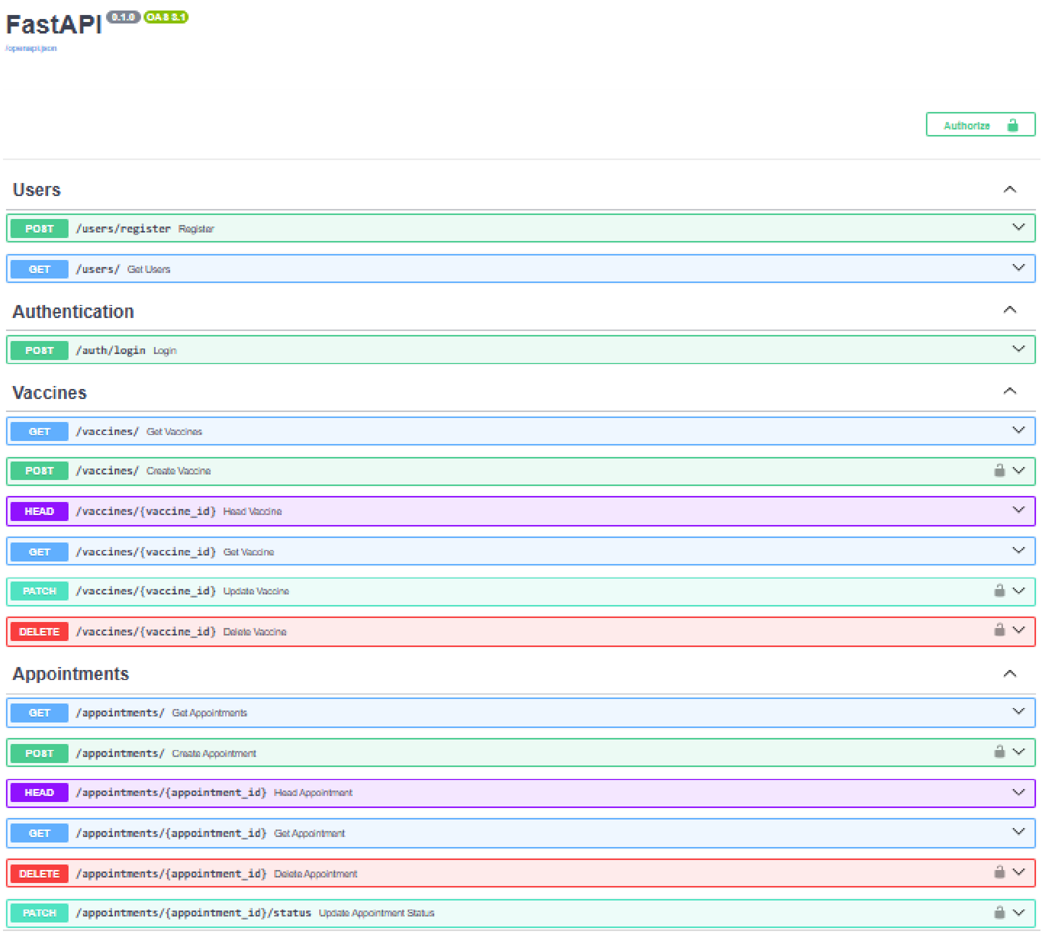
2.5 Postman and Swagger Testing

Postman is an API development and testing platform that allows developers to design, test, and debug APIs by sending HTTP requests and analyzing responses (7) . Swagger also is an open-source framework (OpenAPI) used to design, document, and test RESTful APIs through an interactive and standardized interface (8). I used both here to test my API.



2.6 Swagger API Documentation

Swagger (OpenAPI) is a specification and set of tools used to define, document, and visualize RESTful APIs in a standardized and interactive manner (9).



FastAPI 0.1.0 OAS 3.1
/openapi.json

Authorize

Users

POST /users/register Register

GET /users/ Get Users

Authentication

POST /auth/login Login

Vaccines

GET /vaccines/ Get Vaccines

POST /vaccines/ Create Vaccine

HEAD /vaccines/{vaccine_id} Head Vaccine

GET /vaccines/{vaccine_id} Get Vaccine

PATCH /vaccines/{vaccine_id} Update Vaccine

DELETE /vaccines/{vaccine_id} Delete Vaccine

Appointments

GET /appointments/ Get Appointments

POST /appointments/ Create Appointment

HEAD /appointments/{appointment_id} Head Appointment

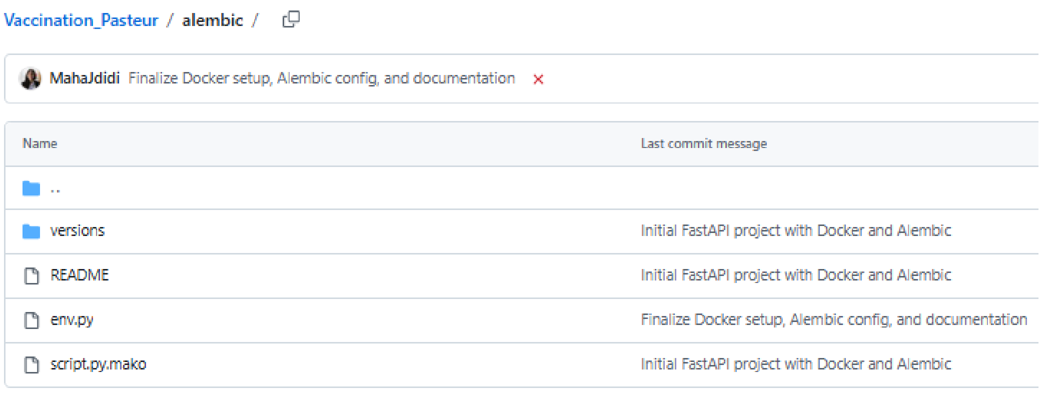
GET /appointments/{appointment_id} Get Appointment

DELETE /appointments/{appointment_id} Delete Appointment

PATCH /appointments/{appointment_id}/status Update Appointment Status

2.7 Git/GitHub for Version Control

Git is a distributed version control system that tracks changes in source code and enables collaboration among developers (10). I used Git and GitHub for version control to manage code changes, maintain history, and collaborate efficiently during development.



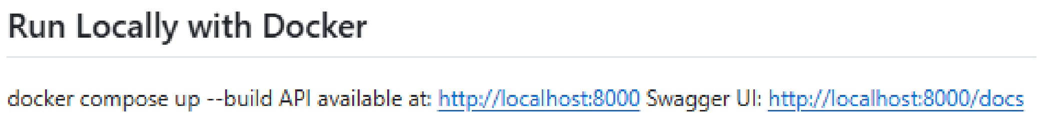
Vaccination_Pasteur / alembic /

MahaJdidi Finalize Docker setup, Alembic config, and documentation ×

Name	Last commit message
..	
versions	Initial FastAPI project with Docker and Alembic
README	Initial FastAPI project with Docker and Alembic
env.py	Finalize Docker setup, Alembic config, and documentation
script.py.mako	Initial FastAPI project with Docker and Alembic

2.8 Docker Containerization

Docker is a containerization platform that enables applications to be packaged and run in lightweight, portable containers (11). I used Docker to containerize the application, ensuring consistent environments and simplifying deployment across different systems.



Run Locally with Docker

docker compose up --build API available at: http://localhost:8000 Swagger UI: http://localhost:8000/docs

Chapter 3

Future Enhancements

This chapter presents potential improvements that could further enhance the vaccination API developed for the Institut Pasteur in Tunisia.

- **Automated Appointment Validation:** Reduce manual workload by automatically approving appointments based on vaccine availability and predefined rules.
- **Smart Notification System:** Notify citizens in real time when appointments are approved, rejected, or approaching.
- **Improved User Experience:** Enhance the platform interface to make booking and accessing vaccination information simpler and more intuitive.
- **Integration with Health Institutions:** Connect the API with national public health systems to improve coordination and data reliability.
- **Mobile Access:** Provide mobile access through a responsive interface or mobile application to reach more citizens across Tunisia.
- **Health Data Insights:** Use collected data to generate insights that support public health planning and vaccination strategies.

Chapter 4

Conclusion

In conclusion, the Vaccination and Awareness API is a platform that aims to enhance the services of the Institut Pasteur in Tunisia in order to better benefit the citizens of the country. The system leverages modern technologies to improve access to vaccination services, manage appointments efficiently, and provide reliable health awareness information. By combining digital tools with public health objectives, the platform demonstrates how technology can strengthen healthcare systems and support disease prevention efforts. Working on this project has been a meaningful and rewarding experience, as it contributes directly to improving public health services. I am proud to have developed a solution that serves the well-being of citizens and supports an important national healthcare institution.

A. APPENDIX

References

[1] "pgAdmin 4." `https://www.pgadmin.org/`

[2] "PostgreSQL." `https://www.postgresql.org/`

[3] "Alembic and FastAPI." `https://alembic.sqlalchemy.org/`

[4] "FastAPI." `https://fastapi.tiangolo.com/`

[5] "Use Case Diagram." `https://www.visual-paradigm.com/guide/uml-unified-modeling-l` `what-is-use-case-diagram/`

[6] "JSON Web Tokens." `https://jwt.io/`

[7] "Postman." `https://www.postman.com/`

[8] "Swagger API Testing." `https://swagger.io/tools/swagger-ui/`

[9] "Swagger API Documentation." `https://swagger.io/specification/`

[10] "Git Version Control." `https://git-scm.com/`

[11] "Docker." `https://www.docker.com/`