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 challenge for many citizens in Tunisia. People often lack reliable information about vaccine schedules, availability, and the importance of timely immunization, whether for general health, travel, or protection against infectious diseases.

This project presents the development of the Vaccination Pasteur API for Institut Pasteur, Tunisia — a smart vaccination management system that automates appointment booking, tracks vaccination records, generates digital certificates, and provides educational content. Citizens can schedule appointments, receive automated reminders, and access accurate vaccine information, while administrators can monitor coverage and performance through analytics.

By combining automated vaccination management, real-time communication, and educational resources, the API helps citizens take proactive steps to protect their health and supports Institut Pasteur in improving vaccination coverage and public health outcomes across Tunisia.

Contents

Chapter 1

Introduction

1.1 Problem Statement

In Tunisia, vaccination services are essential for protecting public health, particularly through institutions such as the Institut Pasteur. However, vaccination appointment processes often rely on manual or fragmented systems, leading to long waiting times, scheduling conflicts, and limited visibility of vaccine availability. Citizens may also lack reliable information about vaccines, which can increase confusion and hesitancy. Healthcare staff face challenges managing appointments, tracking doses, and monitoring coverage without a centralized digital platform. These issues highlight the need for an automated system that streamlines appointment management and provides trustworthy vaccination information.

1.2 Scope

This project develops the Vaccination Pasteur API for the Institut Pasteur in Tunisia, supporting smart appointment management, vaccination record tracking, and public awareness. The API allows citizens to book appointments, receive automated reminders, access vaccine information, and view educational content. Administrators can monitor vaccination coverage, track missed appointments, and manage vaccine inventory. The project focuses on accessibility, efficiency, and information transparency, and 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:** A platform that organizes and automates vaccination appointments at the Institut Pasteur in Tunisia.
- **Smart Appointment Scheduling:** A system that auto-validates appointments, prevents double-booking, and sends reminders to citizens.
- **Vaccine Information Access:** Provides clear, up-to-date information about vaccines, schedules, and booster doses.
- **Public Awareness Enhancement:** Includes educational content to increase vaccination literacy and support informed decision-making.

1.4 System Analysis and Specifications

1. **Related Systems**: While there is no identical API currently deployed for vaccination management at the Institut Pasteur in Tunisia, other health platforms offer basic appointment booking. Most lack vaccine-specific selection, automated reminders, and localized awareness content. This project addresses these gaps with automation, analytics, and educational features.
2. **Target Audience**: The API serves Tunisian citizens booking vaccinations, healthcare staff and administrators managing schedules and vaccines, and developers who may build applications interfacing with the system.
3. **Required Inputs and Expected Outputs**: Inputs include user details, selected vaccine, preferred date/time, and authentication credentials. Administrative inputs include vaccine stock, schedules, and awareness content. Outputs include appointment confirmations, vaccination records, coverage analytics, and educational articles, delivered via structured JSON through RESTful endpoints.

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).**

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.

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**

Search

Object Type	Schema	Name
Table	public	alembic_version
Table	public	appointments
Table	public	awareness_articles
Table	public	booster_schedules
Table	public	notifications
Table	public	reminders
Table	public	users
Table	public	vaccination_certificates
Table	public	vaccinations
Table	public	vaccines

Figure 2.1: Database Schema Diagram

across environments(3). Below is a diagram showcasing the Alembic migration process, illustrating how schema changes are generated, applied, and tracked.

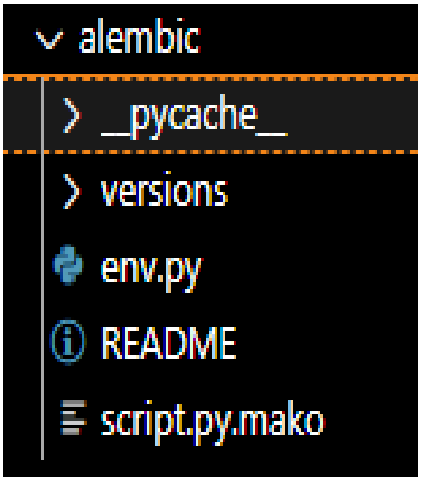


Figure 2.3: Alembic Migration Workflow

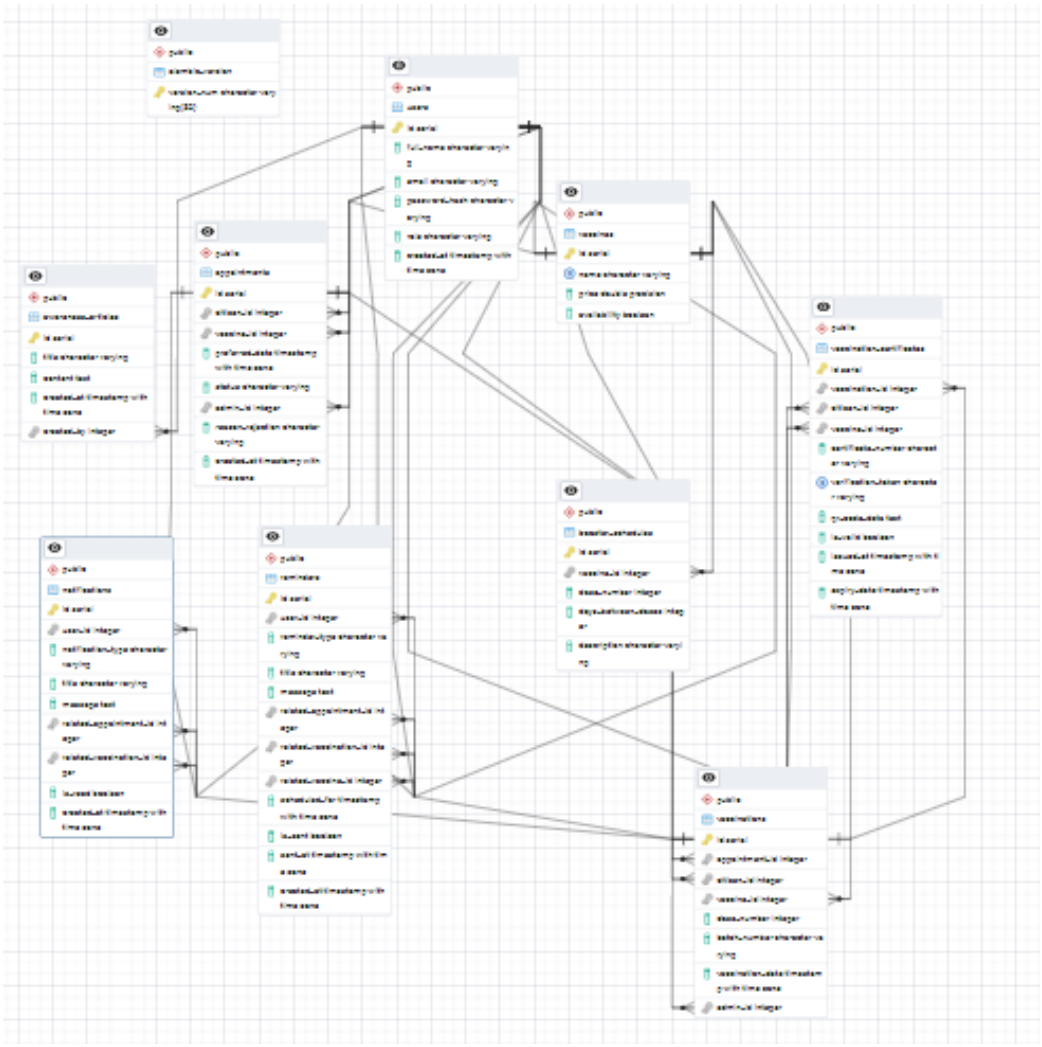


Figure 2.2: Entity-Relationship Diagram (ERD) of the Database

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:

```python
# 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:

```python
# 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:

```python
# 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:

```python
# 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:

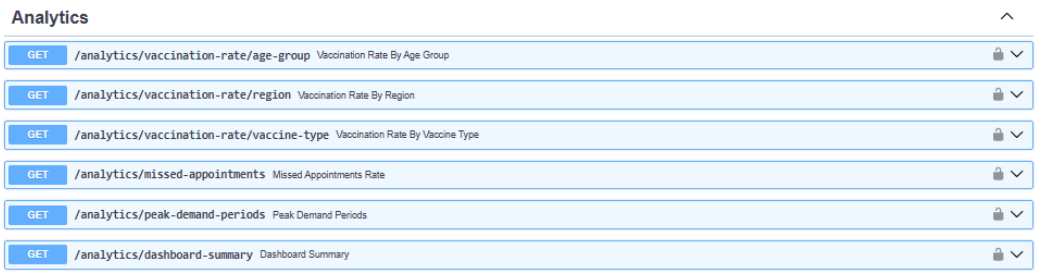
```python
# DELETE article (admin only)
@router.delete("/{article_id}", status_code=204)
def delete_article(article_id: int, db: Session = Depends(get_db),
                   admin: models.User = Depends(require_admin)):
    article = db.query(models.AwarenessArticle).filter(models.AwarenessArticle.id == 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)
```

2.2.6 Notificatios Requests

Overview of the API endpoints.

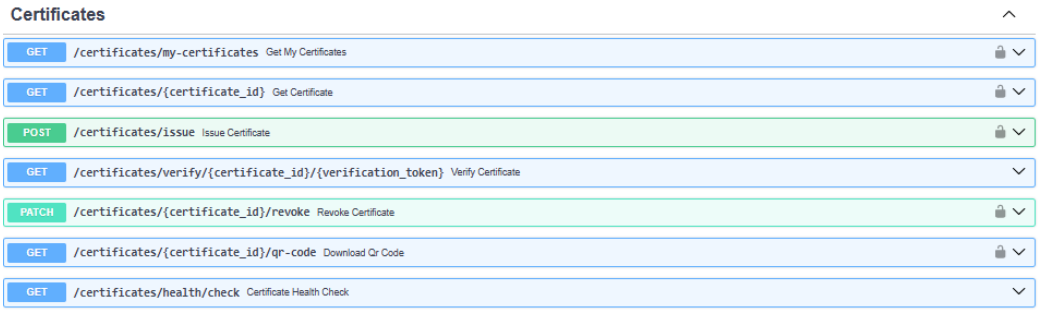
2.2.7 Analytics Requests

Overview of the API endpoints.



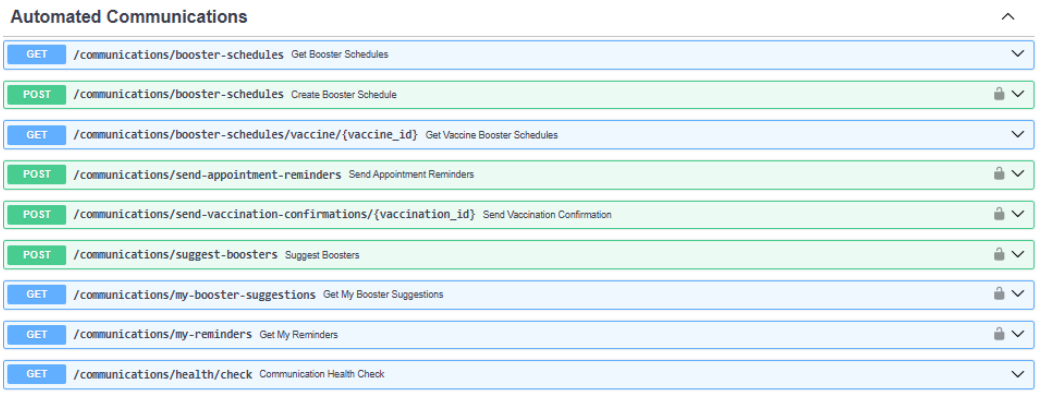
2.2.8 Certificates Requests

Overview of the API endpoints.



2.2.9 Automated Communication Requests

Overview of the API endpoints.



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.

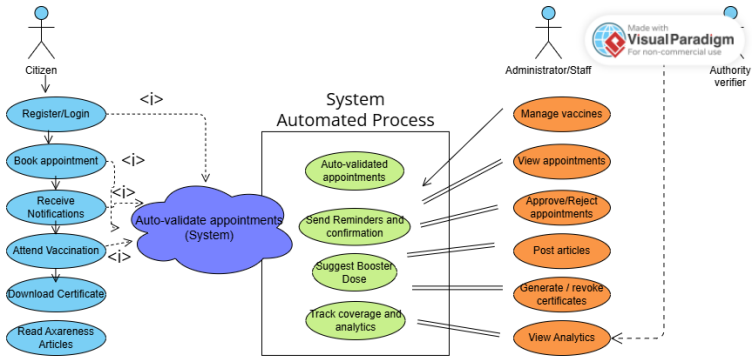


Figure 2.4: Vaccination Platform UCD

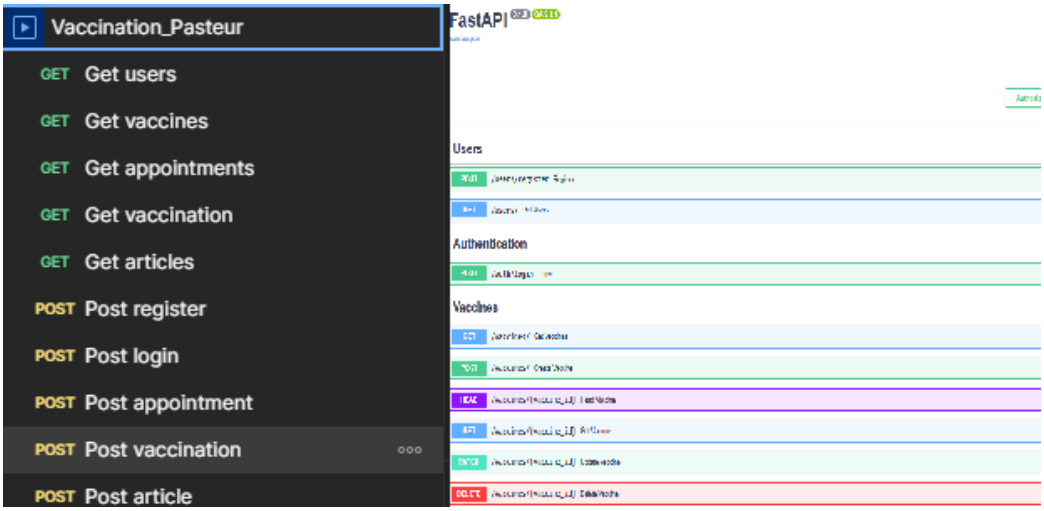
2.4 User Authentication with JWT

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.

2.5 Postman and Swagger Testing

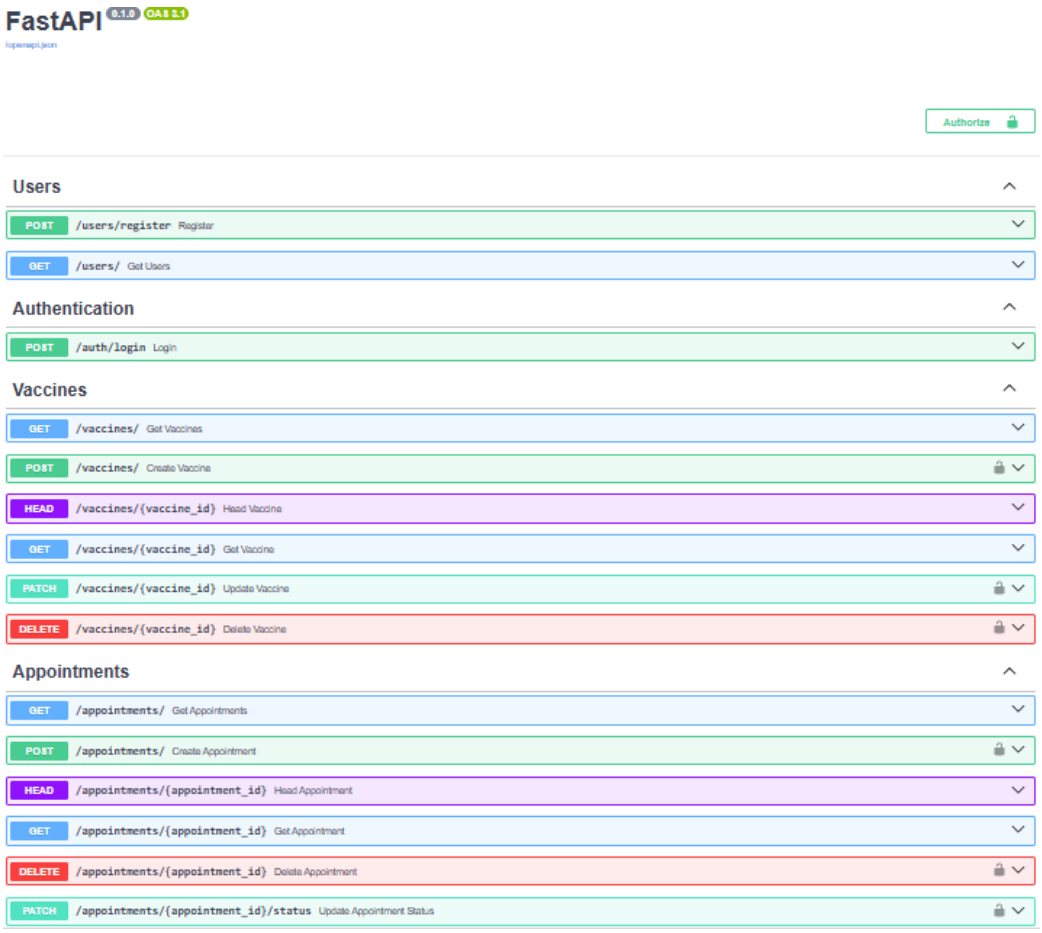
Postman is an API development and testing platform that allows developers to design, test, and debug APIs by sending HTTP requests and analyzing responses (7) . **Swagger also is an open-source**

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.



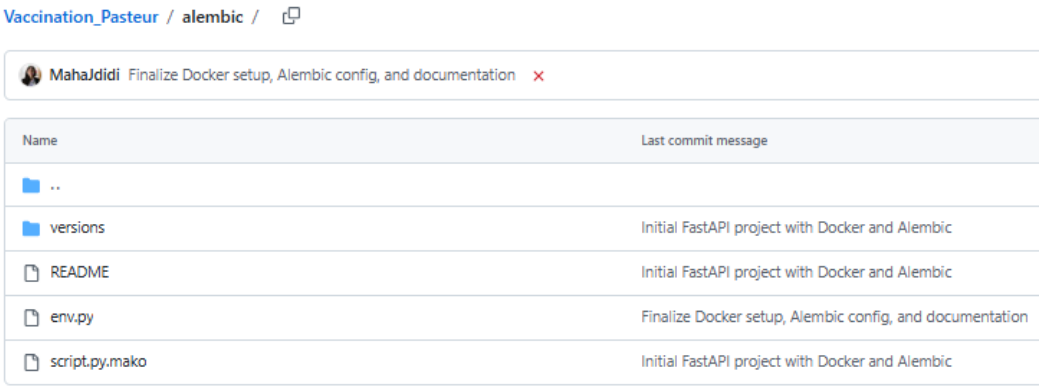
2.6 Swagger API Documentation

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



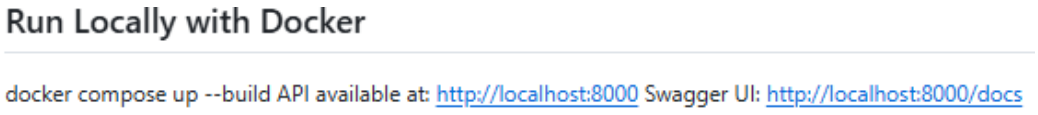
2.7 Git/GitHub for Version Control

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.



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.



Chapter 3

Future Enhancements

This chapter outlines potential improvements that could further strengthen the Vaccination Pasteur API for the Institut Pasteur in Tunisia.

- **User Interface Improvements:** Refine the platform design to simplify booking, accessing vaccination records, and viewing educational content.
- **Integration with Health Systems:** Connect with national public health databases to improve coverage monitoring and reporting.
- **Mobile Accessibility:** Offer mobile-friendly interfaces or apps to reach more citizens across Tunisia efficiently.
- **SMS/Email Gateway Integration:** Enable automated reminders and notifications for appointments, boosters, and campaign updates.
- **Multi-language Support:** Provide content and interfaces in multiple languages to ensure accessibility for all citizens.
- **Transaction Management System:** Track and manage payments, reimbursements, or donations related to vaccination campaigns securely.

Chapter 4

Conclusion

In conclusion, the Vaccination Pasteur API is a smart platform designed to enhance the services of the Institut Pasteur in Tunisia, benefiting citizens and public health efforts. The system leverages modern technologies to automate appointment management, track vaccination records, generate secure digital certificates, and provide educational resources. By combining automation, analytics, and awareness content, the platform demonstrates how technology can support efficient healthcare delivery and improve vaccination coverage. Working on this project has been a meaningful experience, contributing directly to public health in Tunisia. I am proud to have developed a solution that strengthens the Institut Pasteur's services and supports the well-being of citizens.

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/