Project : integration and securing of databases

LOUHOU Godwill

May 16, 2025

Contents

1. Project description

This project is about integrating and securing a database, and implementing secure front-end and backend.

1.1. Concept

The concept of this app is pretty simple, since the emphasis is on the security aspect. It allows to create an account, and create posts with content on it. Other users cannot see the content of the posts.

The features available on the app :
- Users related actions :
  - Registering a new account
  - Logging in
  - Updating user email
  - Deleting the user
- Posts related actions :
  - Creating posts
  - Updating posts
  - Getting posts
  - Deleting posts

The admin user can get information about the other users. Only this feature has been implemented, since actions on other users are trivial to implement.

The authentication is made using `JWT`, and authorization depends the user's `role`.

1.2. Tech stack

The stack chosen for this project is focused on being easy to code on and fast to ship.
- **Backend** :
  - Python
  - FastAPI
- **Frontend** :
  - Simple HTML/CSS/JS website
  - Bulma
- **Database** :
  - MongoDB Atlas
- **Other tools** :
  - Sonarqube
  - Docker
  - Snyk
  - pytest
  - curl
  - Github Actions
  - Wapiti3

2. Database

2.1. Arguments for database choice

- **Fully managed service**

MongoDB Atlas takes care of running, monitoring, and updating the database cluster for us. It makes the work way easier, with less aspects to think of.

- **Built-in high availability**

Atlas automatically replicates the data across multiple nodes (and even regions) so the app stays online through hardware failures or network outages.

- **Elastic scalability**

With higher tier subscriptions it is possible to scale up CPU, RAM, storage, or even shard the data with just a few clicks (or via API). There is no downtime.

- **Global distribution**

It allows to read/write replicas close to the users around the world to minimize latency and deliver a snappier experience.

- **Enterprise-grade security and compliance**

Atlas offers built-in encryption at rest and in transit, fine-grained access controls, compliance certifications (SOC2, GDPR, HIPAA, etc.), so there is not need to build that manually.

- **Automated backups and point-in-time recovery**

Continuous backups let us restore data to any point in time.

- **Rich ecosystem and integrations**

Atlas has an impressive amount of compatible applications, and the `API` for `python` is easy to use, making it a painless experience for the server development.

2.2. CIA Analysis

Provided by `ChatGPT`, about MongoDB Atlas :

2.2.1. Data Availability

- **Replica Sets & Multi-Region Clusters**: Automatic replication across members and regions; failover < 30 s under SLA.
- **Continuous Backups & Point-in-Time Restore**: Ongoing snapshots plus PITR ensure recovery from accidental deletes or regional failures.
- **Automatic Maintenance & Upgrades**: Rolling updates with zero-downtime options; health checks and alerts minimize unplanned outages.

2.2.2. Data Integrity

- **Write-Ahead Journaling**: Ensures committed writes survive crashes; data files replay journal on restart.

- **ACID Transactions**: Multi-document, multi-shard transactions guarantee atomicity, consistency, isolation, durability.
- **Schema Validation & Document Validation**: Enforce field types, ranges, and required fields at the database layer.
- **Audit Logging**: Immutable operation logs for forensic analysis and to detect unauthorized or anomalous modifications.

2.2.3. Data Confidentiality

- **Encryption At Rest**: AES-256 encryption of all storage volumes; customer-managed keys via AWS KMS, Azure Key Vault, or GCP KMS.
- **TLS Encryption In Transit**: Enforced TLS 1.2+ for all client-server and inter-node communications.
- **Network Isolation**: VPC peering, Private Endpoints, IP whitelisting, and firewall rules restrict traffic to authorized networks.
- **Role-Based Access Control (RBAC)**: Granular privileges per database, collection, and action; integration with LDAP and cloud IAM.
- **Field-Level Encryption**: Optional client-side encryption of specific fields so Atlas never sees plaintext values.

3. Back-end description

3.1. Project structure

```
backend/
├── app/
│   ├── database/
│   │   └── db.py
│   ├── middlewares/
│   │   ├── authenticationMiddleware.py
│   │   ├── authorizationMiddleware.py
│   │   └── sanitizationMiddleware.py
│   ├── models/
│   │   ├── connectionDetails.py
│   │   └── post.py
│   ├── utility/
│   │   └── utility.py
│   └── main.py
├── tests/
│   └── test_main.py
├── .dockerignore
├── .env
├── Dockerfile
├── pytest.ini
├── requirements.txt
└── cert.pem
```

3.2. Package : database

This package contains the file `db.py`. This file is used to create the connection to the MongoDB database. It is used in all the operations requiring database queries.

3.3. Package : middlewares

This package contains the 3 middlewares used in the backend.

3.3.1. authenticationMiddleware.py

This middleware is used to check if the user is authenticated, for the concerned routes. A handful of routes are exempt from this verification : `/login` and `/register` since they are accessed when the user is not yet logged in.

It works by checking if a `JWT` is present and valid for the protected routes. If not, doesn't allow to access the route.

3.3.2. authorizationMiddleware.py

This middleware is used to check if the user is authorized to access the concerned resource. It is mainly used for the `/users` route in this project, which allows the `admin` user to retrieve the information of all the users. The middleware can easily be used for other routes or requests, like for managing the users (CRUD).

3.3.3. sanitizationMiddleware.py

This middleware is used to sanitize the data incoming in the backend. It prevents requests from having a body bigger than `500Mib`, to prevent buffer overflow attacks per example. Other mecanisms can be added, like escaping all content passing though it.

3.4. Package : models

This package contains `pydantic` models used for the requests. It allows easy managing of the content parsed by the server.

3.4.1. connectionDetails.py

This model describes the connection informations used for the `login` and `register` actions. It also allows to check the robustness of the passwords used by the users.

```
email : str
password : str
connection_attempts : str
last_connection_attempt : str
role : str
```

3.4.2. post.py

This model describes the way the `posts` are represented. It facilitates every operation between the server and the database.

```
title : str
content : str
user : str
```

3.5. Package : utility

This package contains only one file, containing utility methods, to prevent boilerplate code.

3.5.1. utility.py

Contains methods used in various files across the project, mainly related to databases queries.

3.6. Package : tests

This package contains the test file for code coverage. It uses `pytest`.

3.6.1. test_main.py

Contains all the implemented tests.

3.7. main.py

The main file. Contains all the routes for the requests.

3.8. A note on security

The implementation of HTTPS has been made, using a self-signed certificate. This implies that it will not be reproducible if the certificate is not installed and defined as trusted on the host computer.

4. Front-end description

The front-end is a simple `HTML/CSS/JS` page. The `js` functions allows the app to send queries to the server and process the responses.

`textContent` is used in the code, preventing any XSS attack, and the server escapes any illegal character to prevent any database injection.

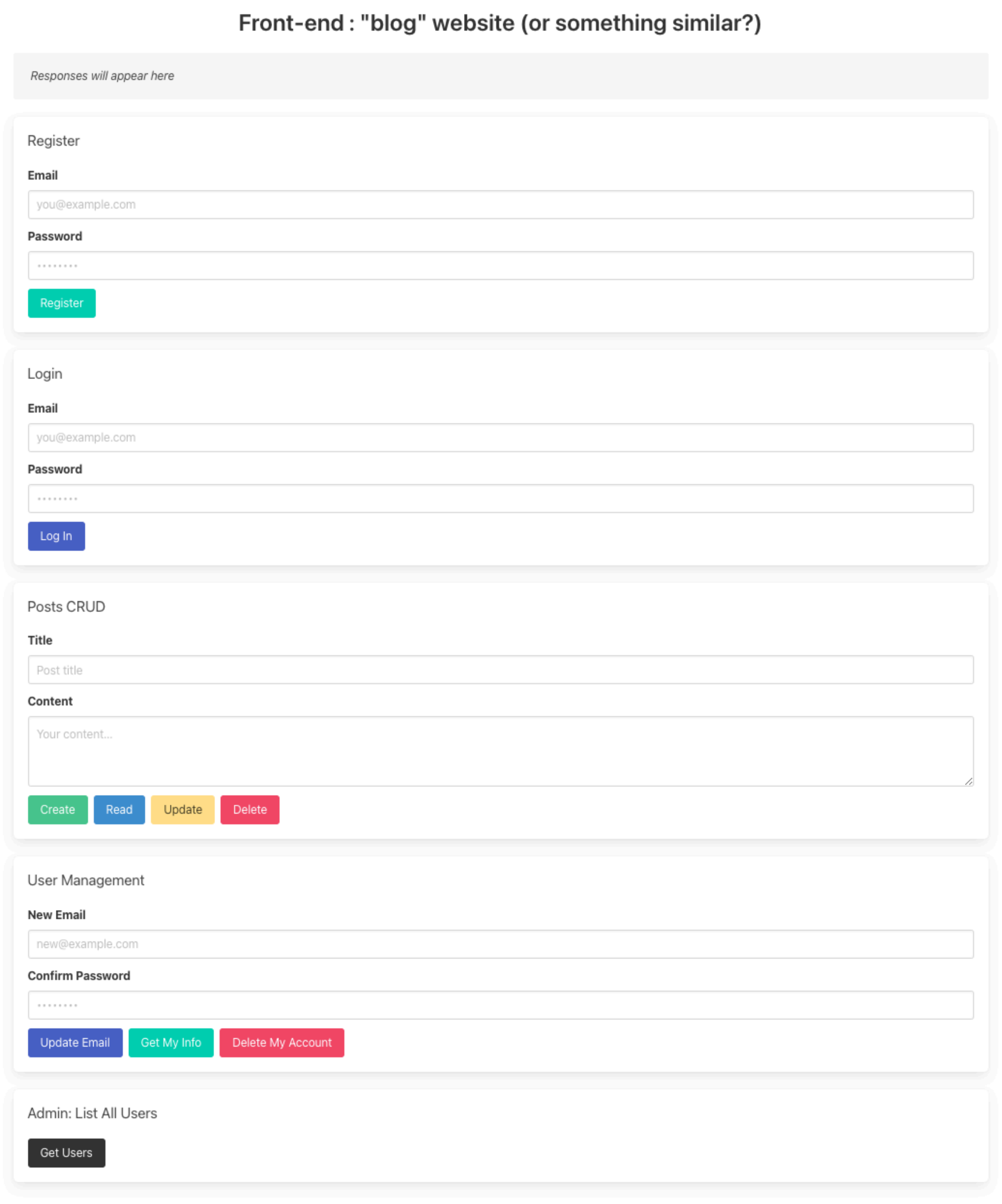


Figure 1: Front-end page

5. Architecture

Here is an overview of the different requests used in the project.

5.1. login : request

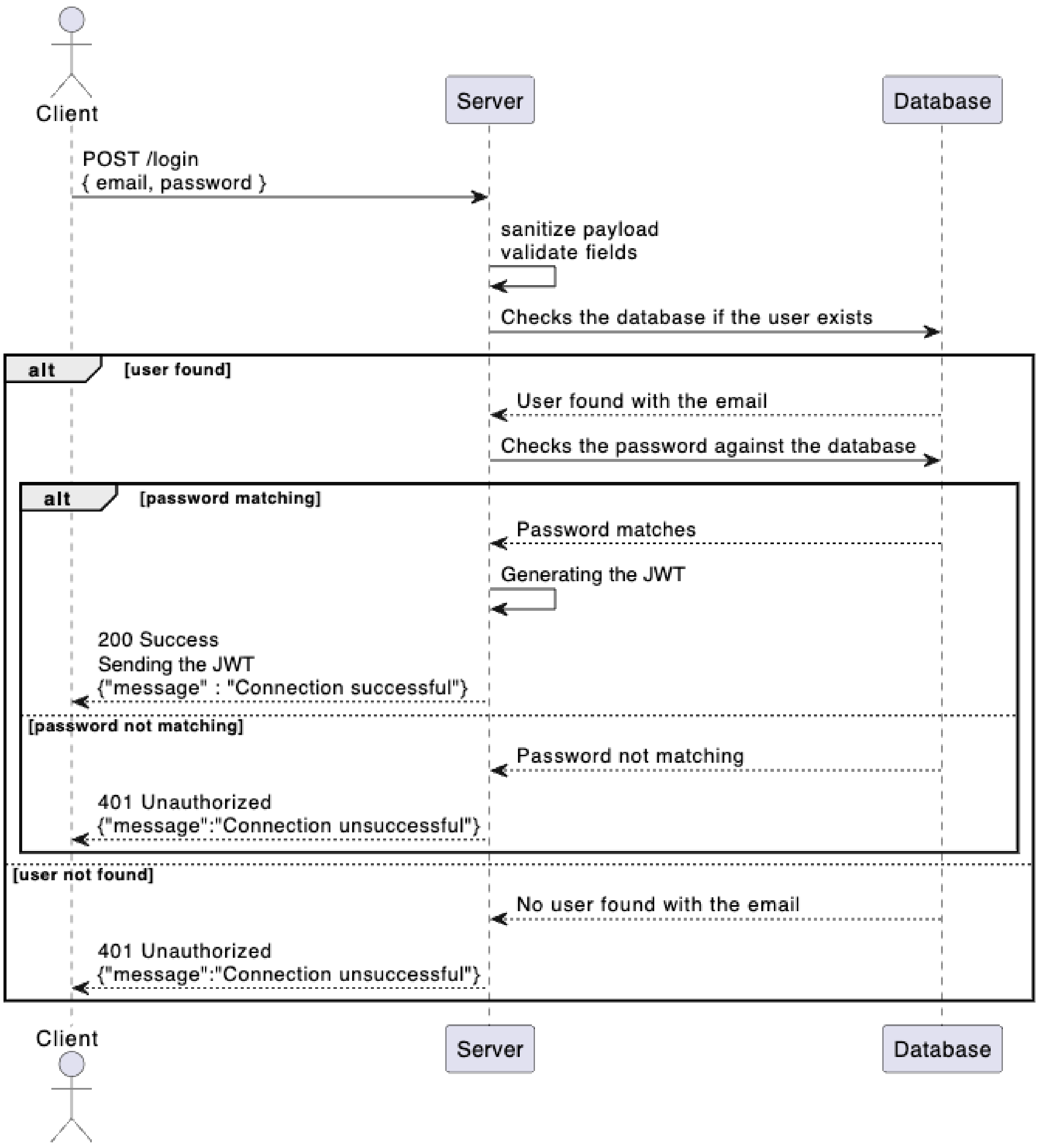
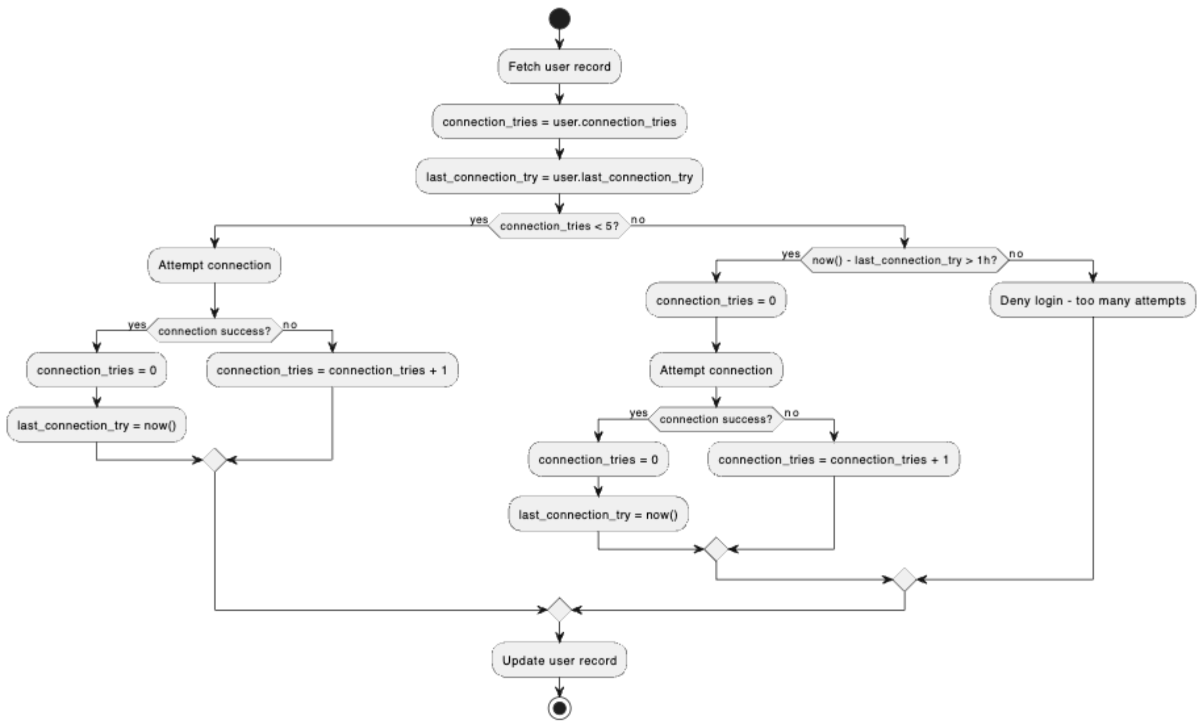


Figure 2: Login request

5.2. login : flow details



Fetch user record
connection_tries = user.connection_tries
last_connection_try = user.last_connection_try
connection_tries < 5?
yes
no
Attempt connection
connection success?
yes
no
connection_tries = 0
last_connection_try = now()
connection_tries = connection_tries + 1
now() - last_connection_try > 1h?
yes
no
connection_tries = 0
Attempt connection
connection success?
yes
no
connection_tries = 0
last_connection_try = now()
connection_tries = connection_tries + 1
Deny login - too many attempts
Update user record

Figure 3: Login decision flow

5.3. register : request

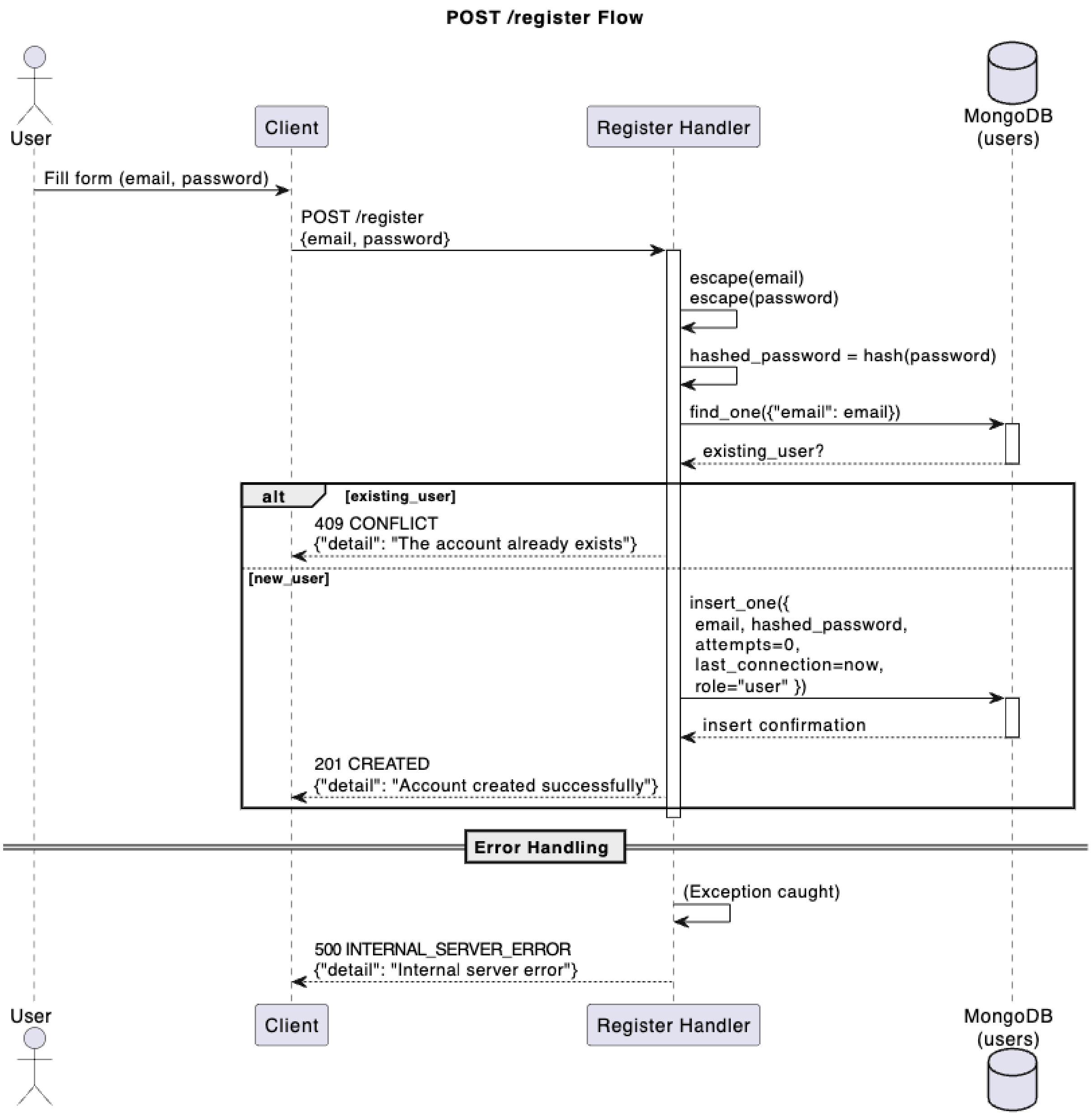

POST /register Flow
User
Client
Register Handler
MongoDB (users)
Fill form (email, password)
POST /register {email, password}
escape(email) escape(password)
hashed_password = hash(password)
find_one({"email": email})
existing_user?
alt [existing_user]
409 CONFLICT {"detail": "The account already exists"}
[new_user]
insert_one({ email, hashed_password, attempts=0, last_connection=now, role="user" })
insert confirmation
201 CREATED {"detail": "Account created successfully"}
Error Handling
(Exception caught)
500 INTERNAL_SERVER_ERROR {"detail": "Internal server error"}

Figure 4: Register request

5.4. post : creation

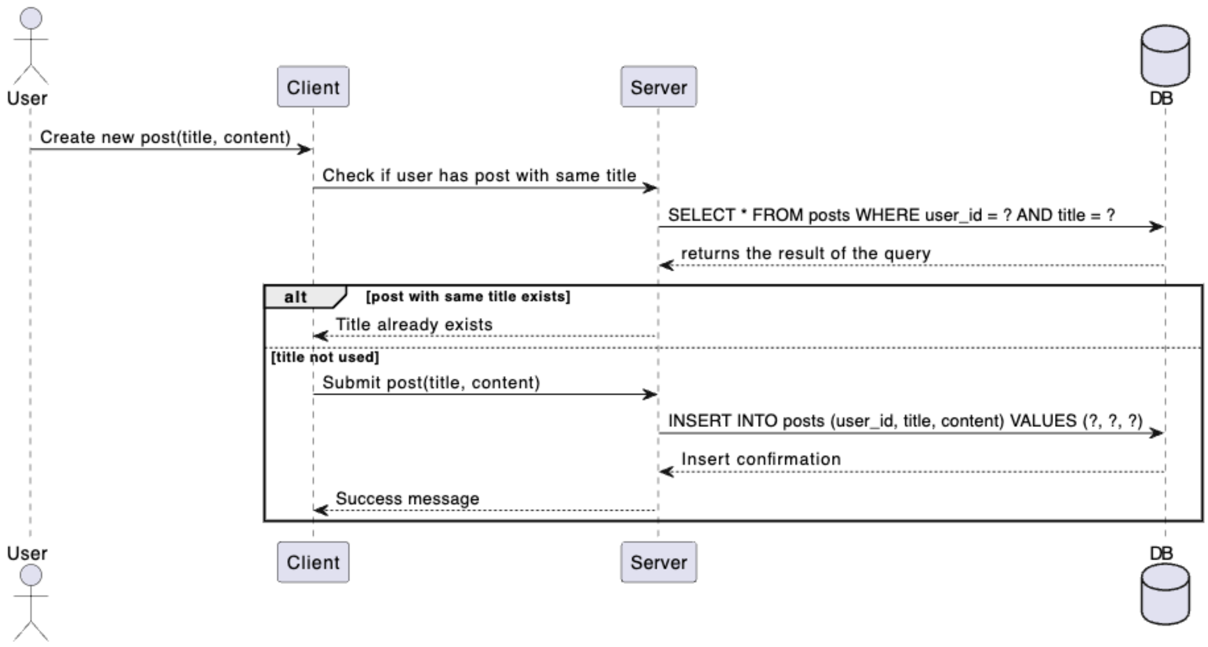

User
Client
Server
DB
Create new post(title, content)
Check if user has post with same title
SELECT * FROM posts WHERE user_id = ? AND title = ?
returns the result of the query
alt
[post with same title exists]
Title already exists
[title not used]
Submit post(title, content)
INSERT INTO posts (user_id, title, content) VALUES (?, ?, ?)
Insert confirmation
Success message

Figure 5: Post creation request

5.5. post : update

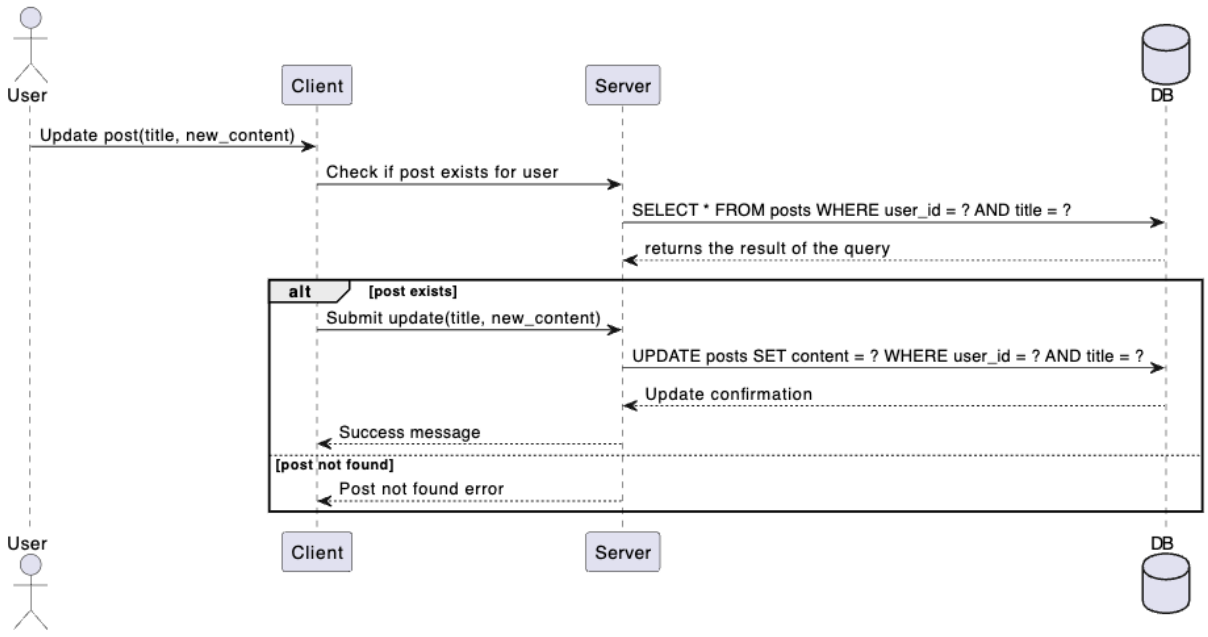

User
Client
Server
DB
Update post(title, new_content)
Check if post exists for user
SELECT * FROM posts WHERE user_id = ? AND title = ?
returns the result of the query
alt
[post exists]
Submit update(title, new_content)
UPDATE posts SET content = ? WHERE user_id = ? AND title = ?
Update confirmation
Success message
[post not found]
Post not found error

Figure 6: Post update request

5.6. post : deletion

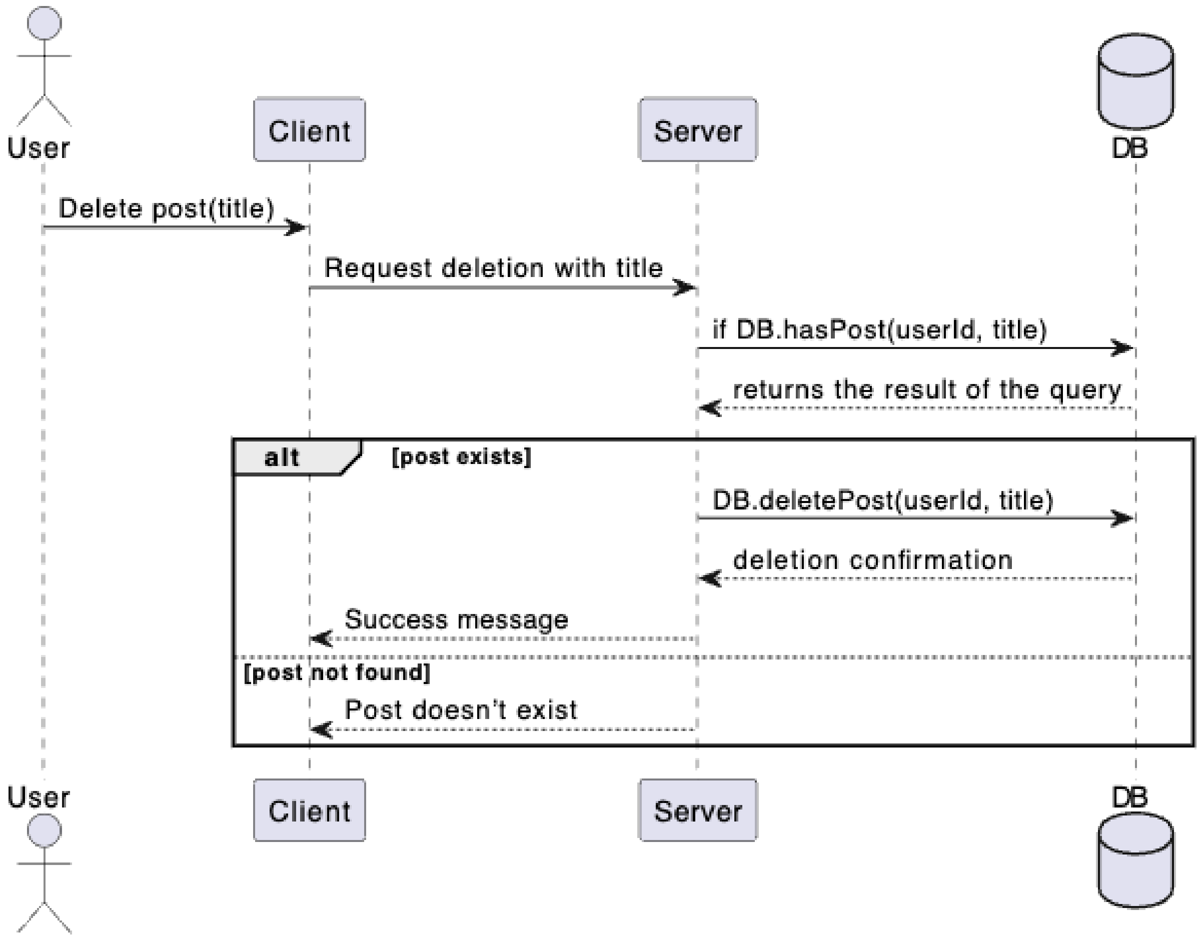
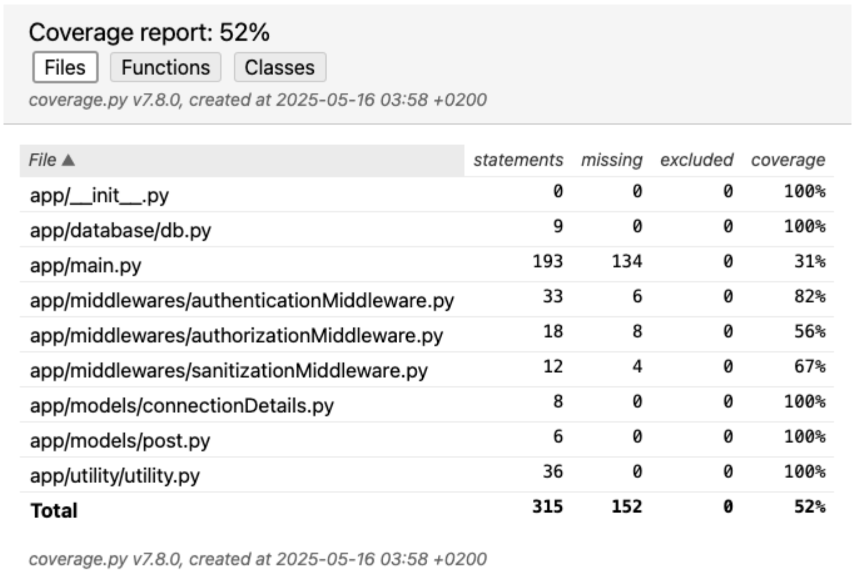


Figure 7: Post deletion request

6. CI/CD

6.1. Code coverage

The usage of `pytest` allows to cover the code by creating tests for each method/function. In this project, because of the time limit, the code coverage is only about `52%`.



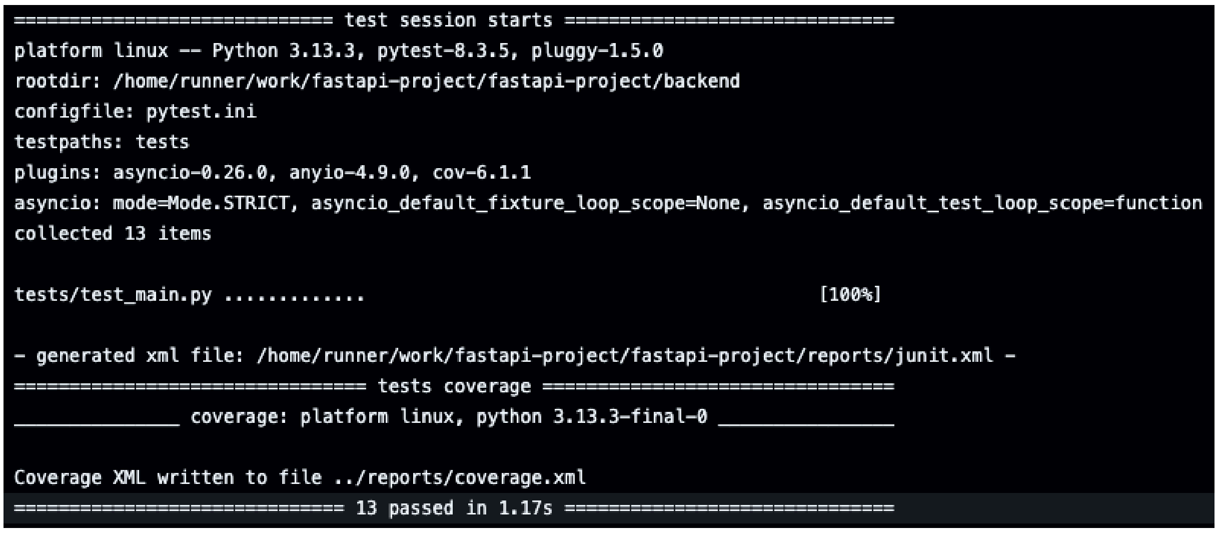
Coverage report: 52%

coverage.py v7.8.0, created at 2025-05-16 03:58 +0200

File ▲	statements	missing	excluded	coverage
app/__init__.py	0	0	0	100%
app/database/db.py	9	0	0	100%
app/main.py	193	134	0	31%
app/middlewares/authenticationMiddleware.py	33	6	0	82%
app/middlewares/authorizationMiddleware.py	18	8	0	56%
app/middlewares/sanitizationMiddleware.py	12	4	0	67%
app/models/connectionDetails.py	8	0	0	100%
app/models/post.py	6	0	0	100%
app/utility/utility.py	36	0	0	100%
Total	**315**	**152**	**0**	**52%**

coverage.py v7.8.0, created at 2025-05-16 03:58 +0200

Figure 8: Code coverage with pytest

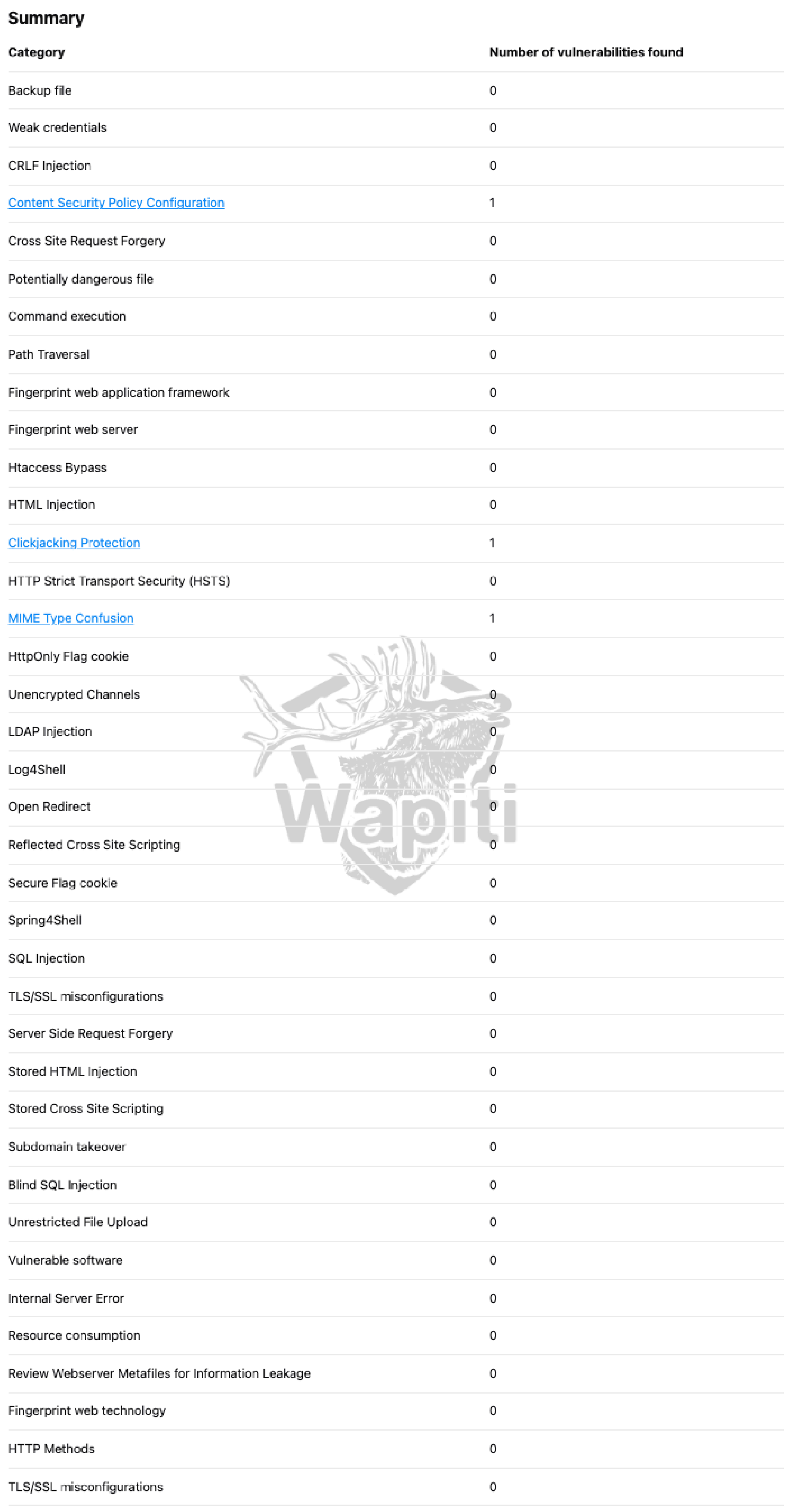


```
============================= test session starts ==============================
platform linux -- Python 3.13.3, pytest-8.3.5, pluggy-1.5.0
rootdir: /home/runner/work/fastapi-project/fastapi-project/backend
configfile: pytest.ini
testpaths: tests
plugins: asyncio-0.26.0, anyio-4.9.0, cov-6.1.1
asyncio: mode=Mode.STRICT, asyncio_default_fixture_loop_scope=None, asyncio_default_test_loop_scope=function
collected 13 items

tests/test_main.py .............                                         [100%]

- generated xml file: /home/runner/work/fastapi-project/fastapi-project/reports/junit.xml -
================================ tests coverage ================================
_______________ coverage: platform linux, python 3.13.3-final-0 ________________

Coverage XML written to file ../reports/coverage.xml
============================== 13 passed in 1.17s ==============================
```

Figure 9: Passed tests in github actions

6.2. Vulnerability scanner

The usage of `wapiti3` allows to scan the web application to check if vulnerabilities are present. Only minor vulnerabilites have been found. Patching them is not done right now, because of the limited time.



Summary

Category	Number of vulnerabilities found
Backup file	0
Weak credentials	0
CRLF Injection	0
Content Security Policy Configuration	1
Cross Site Request Forgery	0
Potentially dangerous file	0
Command execution	0
Path Traversal	0
Fingerprint web application framework	0
Fingerprint web server	0
Htaccess Bypass	0
HTML Injection	0
Clickjacking Protection	1
HTTP Strict Transport Security (HSTS)	0
MIME Type Confusion	1
HttpOnly Flag cookie	0
Unencrypted Channels	0
LDAP Injection	0
Log4Shell	0
Open Redirect	0
Reflected Cross Site Scripting	0
Secure Flag cookie	0
Spring4Shell	0
SQL Injection	0
TLS/SSL misconfigurations	0
Server Side Request Forgery	0
Stored HTML Injection	0
Stored Cross Site Scripting	0
Subdomain takeover	0
Blind SQL Injection	0
Unrestricted File Upload	0
Vulnerable software	0
Internal Server Error	0
Resource consumption	0
Review Webserver Metafiles for Information Leakage	0
Fingerprint web technology	0
HTTP Methods	0
TLS/SSL misconfigurations	0

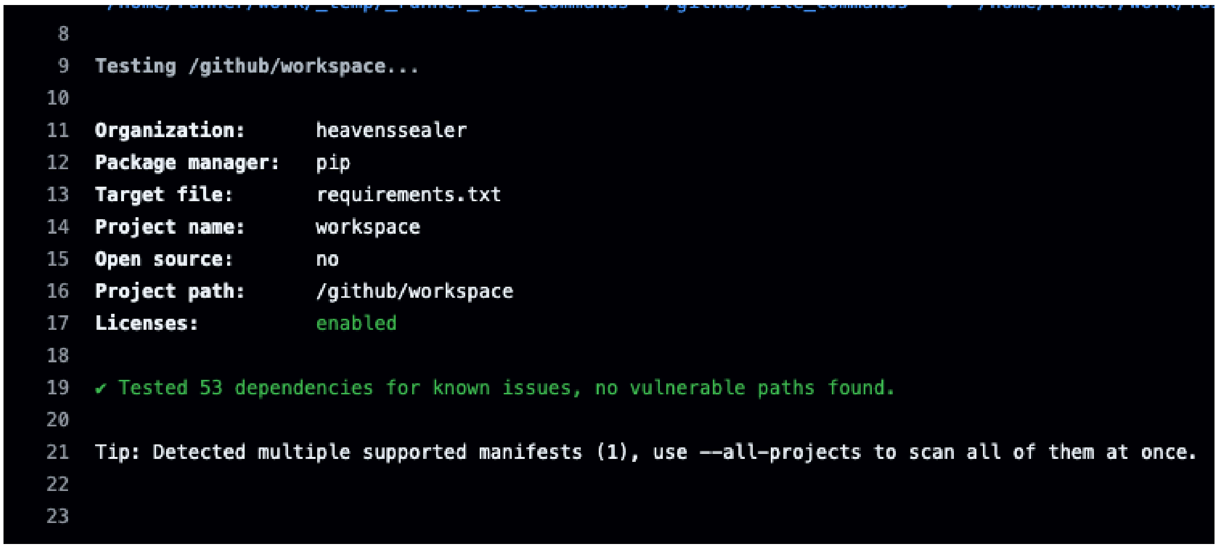
Figure 10: Wapiti scan result

6.3. Github Workflow

Github actions have been set up. At each `pull request` and `push`, the Github Actions trigger `Snyk` to scan the code, and then `Sonarqube`.

6.3.1. Snyk

`Snyk` allows to check for security vulnerabilities in the dependencies.



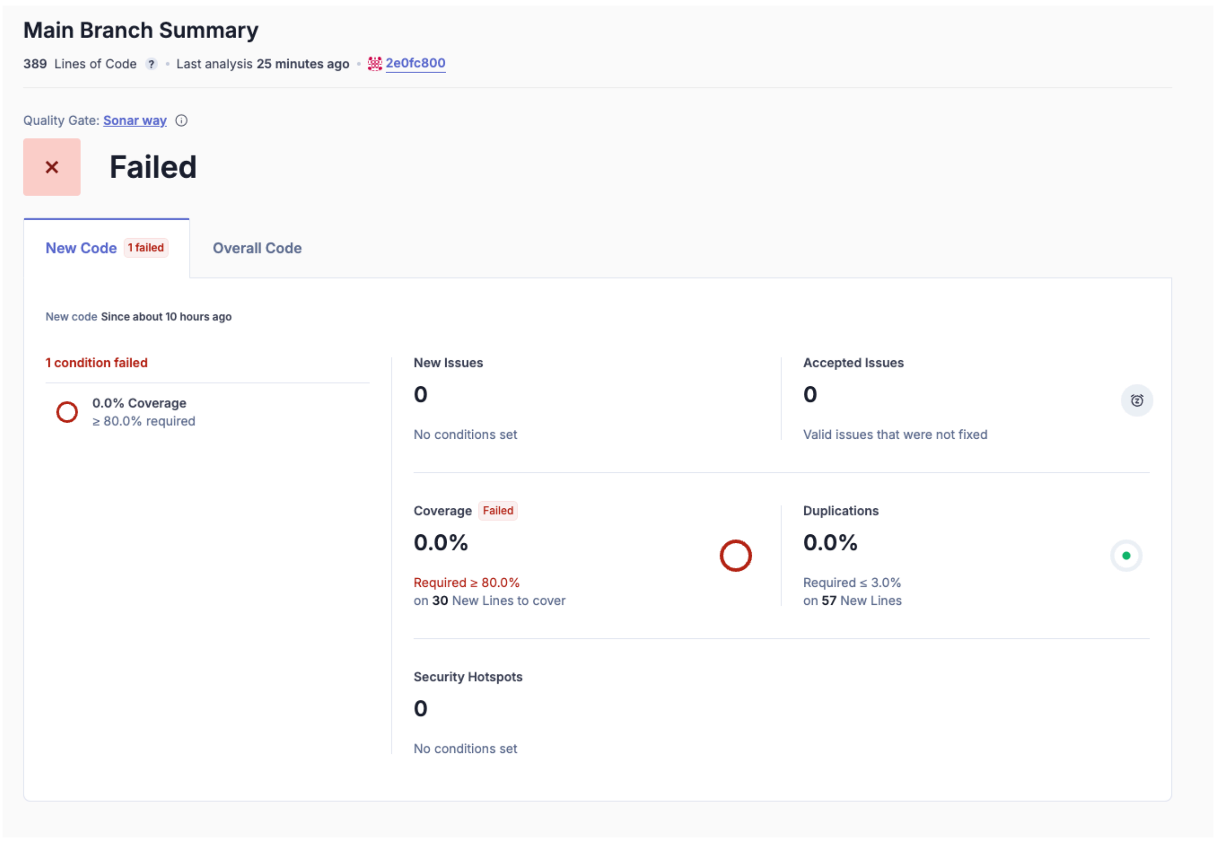
```

Testing /github/workspace...

Organization:      heavenssealer
Package manager:   pip
Target file:       requirements.txt
Project name:      workspace
Open source:       no
Project path:      /github/workspace
Licenses:          enabled

✔ Tested 53 dependencies for known issues, no vulnerable paths found.

Tip: Detected multiple supported manifests (1), use --all-projects to scan all of them at once.


```

Figure 11: Snyk test

6.3.2. Sonarqube

`Sonarqube` checks for multiple indicators :

- Reliability flaws (bugs)
- Security vulnerabilities
- Maintainability issues (code smells, technical debt)
- Quality metrics (coverage, duplications)


Main Branch Summary

389 Lines of Code · Last analysis 25 minutes ago · 2e0fc800

Quality Gate: Sonar way

Failed

New Code 1 failed | Overall Code

New code Since about 10 hours ago

1 condition failed

0.0% Coverage
≥ 80.0% required

New Issues
0
No conditions set

Accepted Issues
0
Valid issues that were not fixed

Coverage Failed
0.0%
Required ≥ 80.0%
on 30 New Lines to cover

Duplications
0.0%
Required ≤ 3.0%
on 57 New Lines

Security Hotspots
0
No conditions set

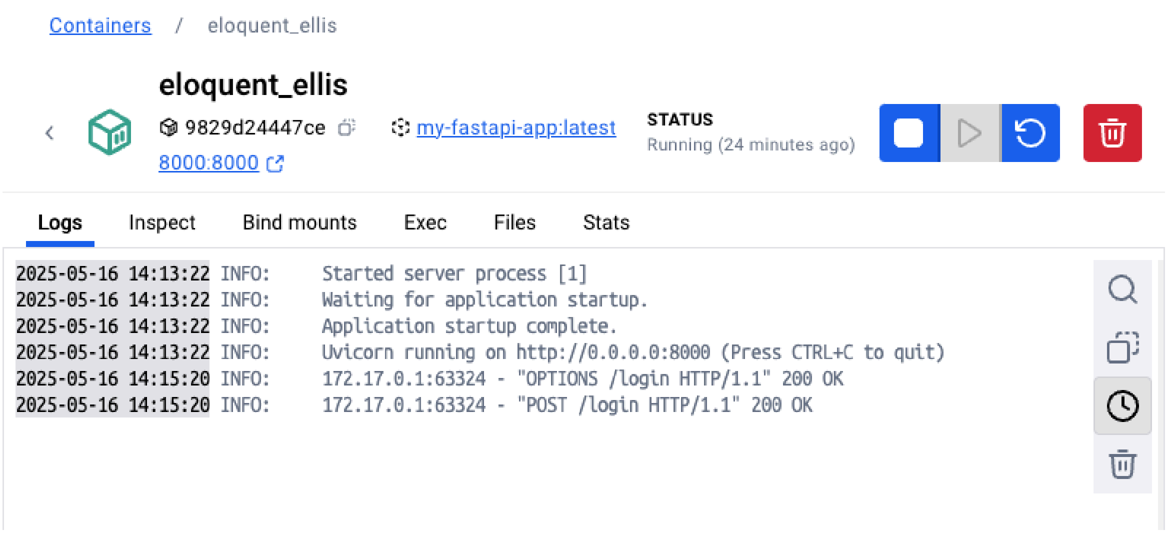
Figure 12: Sonarqube analysis

The coverage is done using pytests, so it is not accounted for here. Despite it showing Failed, the conditions that we check are Duplications and Security Hotspots.

6.4. Dockerization

The server is dockerized, allowing it to be executed from another computer easily. It can also help to host it on cloud services.

The command used to dockerize :
- to build :
  - `docker build -f backend/Dockerfile -t my-fastapi-app:latest backend/`
- to launch the container :
  - `docker run -d -p 8000:8000 --env-file backend/.env my-fastapi-app:latest`

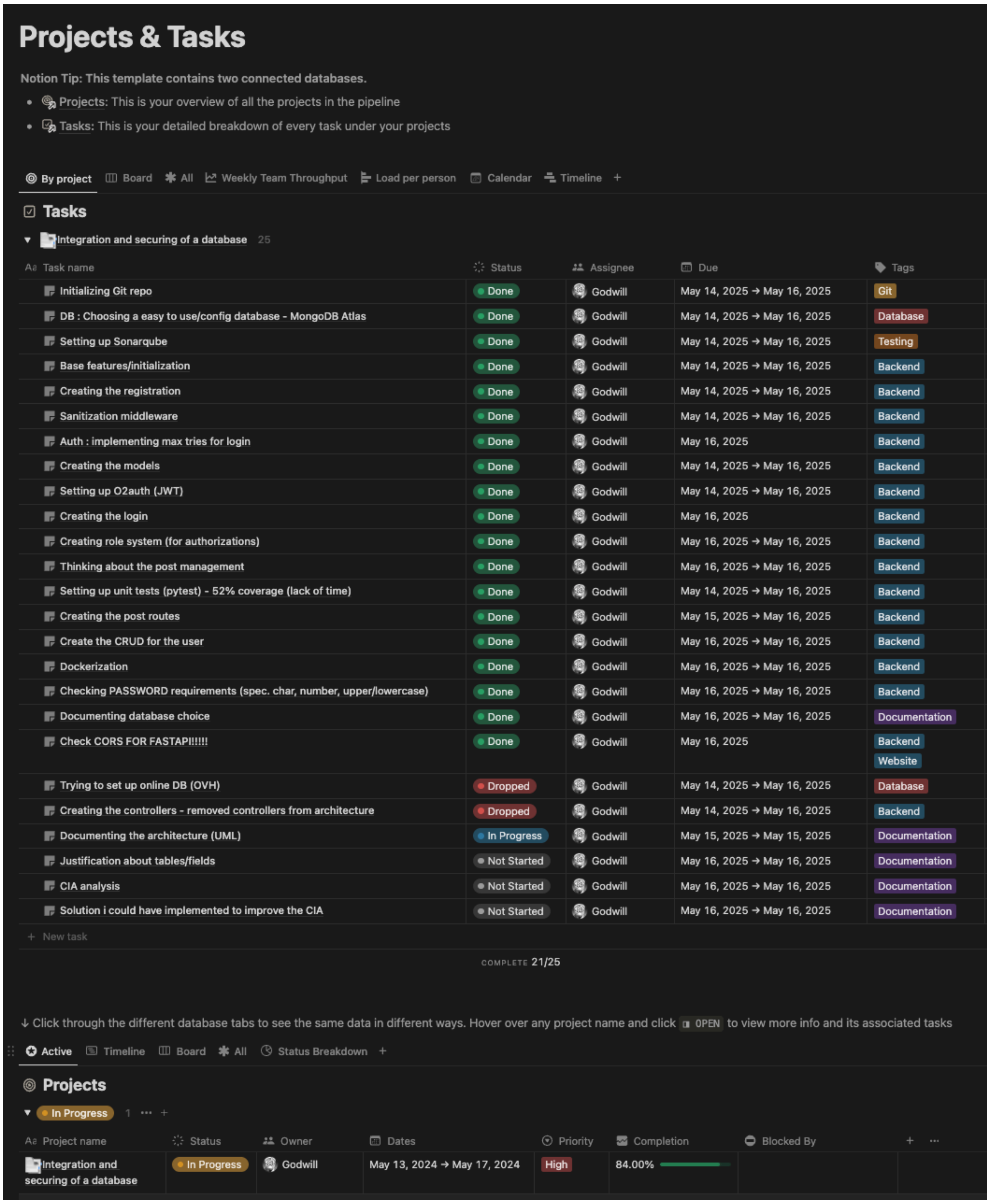


```
2025-05-16 14:13:22 INFO:     Started server process [1]
2025-05-16 14:13:22 INFO:     Waiting for application startup.
2025-05-16 14:13:22 INFO:     Application startup complete.
2025-05-16 14:13:22 INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
2025-05-16 14:15:20 INFO:     172.17.0.1:63324 - "OPTIONS /login HTTP/1.1" 200 OK
2025-05-16 14:15:20 INFO:     172.17.0.1:63324 - "POST /login HTTP/1.1" 200 OK
```

Figure 13: Docker running

7. Project management

`Notion` was used for `agility`, helping with organizing and managing tasks.



Projects & Tasks

Notion Tip: This template contains two connected databases.

- Projects: This is your overview of all the projects in the pipeline
- Tasks: This is your detailed breakdown of every task under your projects

Tasks

▼ Integration and securing of a database 25

Task name	Status	Assignee	Due	Tags
Initializing Git repo	Done	Godwill	May 14, 2025 → May 16, 2025	Git
DB : Choosing a easy to use/config database - MongoDB Atlas	Done	Godwill	May 14, 2025 → May 16, 2025	Database
Setting up Sonarqube	Done	Godwill	May 14, 2025 → May 16, 2025	Testing
Base features/initialization	Done	Godwill	May 14, 2025 → May 16, 2025	Backend
Creating the registration	Done	Godwill	May 14, 2025 → May 16, 2025	Backend
Sanitization middleware	Done	Godwill	May 14, 2025 → May 16, 2025	Backend
Auth : implementing max tries for login	Done	Godwill	May 16, 2025	Backend
Creating the models	Done	Godwill	May 14, 2025 → May 16, 2025	Backend
Setting up O2auth (JWT)	Done	Godwill	May 14, 2025 → May 16, 2025	Backend
Creating the login	Done	Godwill	May 16, 2025	Backend
Creating role system (for authorizations)	Done	Godwill	May 16, 2025 → May 16, 2025	Backend
Thinking about the post management	Done	Godwill	May 16, 2025 → May 16, 2025	Backend
Setting up unit tests (pytest) - 52% coverage (lack of time)	Done	Godwill	May 14, 2025 → May 16, 2025	Backend
Creating the post routes	Done	Godwill	May 15, 2025 → May 16, 2025	Backend
Create the CRUD for the user	Done	Godwill	May 16, 2025 → May 16, 2025	Backend
Dockerization	Done	Godwill	May 16, 2025 → May 16, 2025	Backend
Checking PASSWORD requirements (spec. char, number, upper/lowercase)	Done	Godwill	May 16, 2025 → May 16, 2025	Backend
Documenting database choice	Done	Godwill	May 16, 2025 → May 16, 2025	Documentation
Check CORS FOR FASTAPI!!!!!	Done	Godwill	May 16, 2025	Backend Website
Trying to set up online DB (OVH)	Dropped	Godwill	May 14, 2025 → May 16, 2025	Database
Creating the controllers - removed controllers from architecture	Dropped	Godwill	May 14, 2025 → May 16, 2025	Backend
Documenting the architecture (UML)	In Progress	Godwill	May 15, 2025 → May 15, 2025	Documentation
Justification about tables/fields	Not Started	Godwill	May 16, 2025 → May 16, 2025	Documentation
CIA analysis	Not Started	Godwill	May 16, 2025 → May 16, 2025	Documentation
Solution i could have implemented to improve the CIA	Not Started	Godwill	May 16, 2025 → May 16, 2025	Documentation

+ New task

COMPLETE 21/25

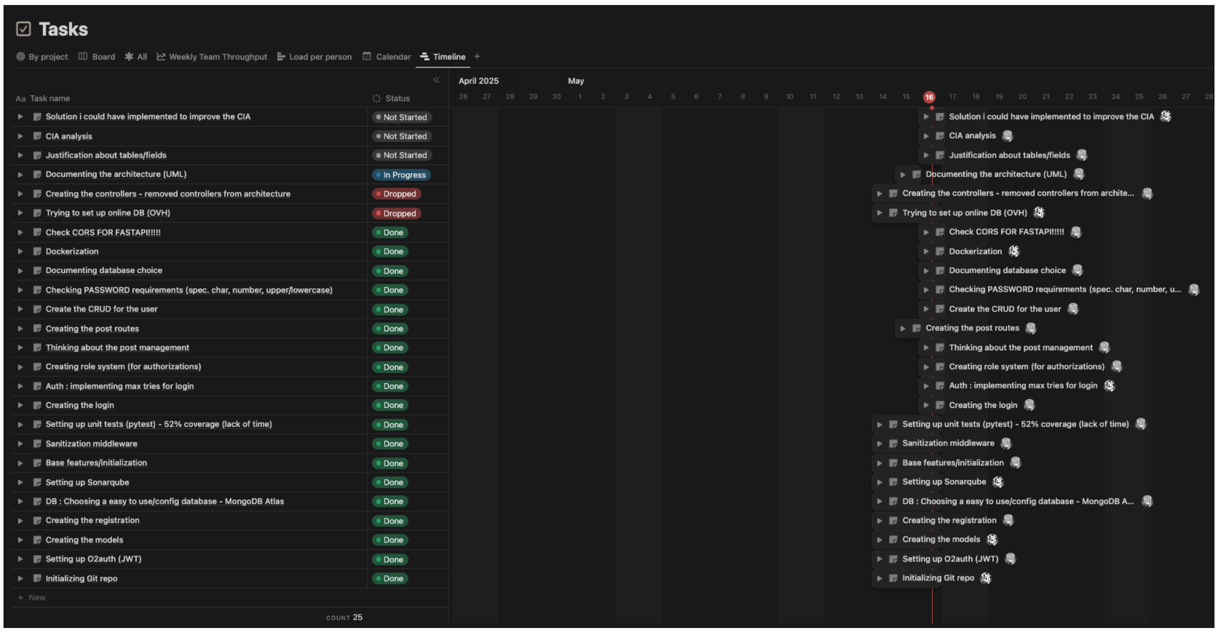
↓ Click through the different database tabs to see the same data in different ways. Hover over any project name and click OPEN to view more info and its associated tasks

Projects

▼ In Progress 1

Project name	Status	Owner	Dates	Priority	Completion	Blocked By
Integration and securing of a database	In Progress	Godwill	May 13, 2024 → May 17, 2024	High	84.00%	

Figure 14: Project management with Notion


Tasks

By project · Board · All · Weekly Team Throughput · Load per person · Calendar · Timeline

Task name	Status
Solution i could have implemented to improve the CIA	Not Started
CIA analysis	Not Started
Justification about tables/fields	Not Started
Documenting the architecture (UML)	In Progress
Creating the controllers - removed controllers from architecture	Dropped
Trying to set up online DB (OVH)	Dropped
Check CORS FOR FASTAPI!!!!!	Done
Dockerization	Done
Documenting database choice	Done
Checking PASSWORD requirements (spec. char, number, upper/lowercase)	Done
Create the CRUD for the user	Done
Creating the post routes	Done
Thinking about the post management	Done
Creating role system (for authorizations)	Done
Auth : implementing max tries for login	Done
Creating the login	Done
Setting up unit tests (pytest) - 52% coverage (lack of time)	Done
Sanitization middleware	Done
Base features/initialization	Done
Setting up Sonarqube	Done
DB : Choosing a easy to use/config database - MongoDB Atlas	Done
Creating the registration	Done
Creating the models	Done
Setting up O2auth (JWT)	Done
Initializing Git repo	Done

COUNT 25

Figure 15: Tasks with Notion

8. Improvements

To improve the project, there is a certain number of elements that could have been improved/implemented:

- TLS/SSL : generating a real certificate instead of a self signed one.
- Improved code coverage : code coverage for more than 80% is the industry norm.
- Responsive/Reactive design : the frontend could be made using React or other libraries/frameworks, to enhance user experience.
- More robust input sanitization: although the inputs are sanitized in the backend, there might be better ways to improve the robustness of the server.

9. Installation

To make the project work on your machine, because of the certificate constraints, you will have to generate your own.

First, create a `san.cnf` file and add the following lines to it :

```
[req]
req_extensions = v3_req
distinguished_name = req_distinguished_name
prompt = no

[req_distinguished_name]
CN = localhost

[v3_req]
subjectAltName = @alt_names

[alt_names]
DNS.1 = localhost
IP.1  = 127.0.0.1
IP.2  = ::1
```

Then, execute this command to generate the certificate and the key :

```
openssl req -x509 -nodes -days 365 \
  -newkey rsa:2048 \
  -keyout local_key.pem \
  -out local_cert.pem \
  -config san.cnf \
  -extensions v3_req
```

Then, go to `backend/certs/` and add the generated certificates `local_key.pem` and `local_cert.pem`.

Then, you can just build the docker image and run it.

Finally, to run the frontend, go to the `frontend` directory and run :

`python3 -m http.server 8080`

This will allow you to access the web application from `http://localhost:8080/index.html`.

You should then be good to go.

Index of Figures