Tunis Business School

This is the final project for Web Service class

Loss and Damage Funds

REST-API

Web Service Project

By

Manel Saddouki

Professor : Dr. Montassar Ben Messaoud

Tunis Business School

BA/IT

Date of submission: January 27, 2024

DECLARATION

I, Manel Saddouki confirm that the work presented in this report is my own. Where information has been derived from other sources, I confirm that this has been indicated in the reference section.

_____

Manel Saddouki

ABSTRACT

This project is part of the Web Service class, focusing on the assigned theme of 'Politics.' The objective is to design and implement an API centered around a political topic.

For my project, I chose to develop an API that manages funds allocated by countries to address specific issues such as the effect of war, responses to natural disasters or limiting the impacts of climate change.

As part of this project, I have created a prototype for the front end. Now, we have a user-friendly interface that facilitates interaction with the API. The live version of my web service is accessible at https://lossanddamagefunds.onrender.com/.

The development process involved various libraries and tools:

- **For the backend and development:** Python served as the primary programming language, supported by several libraries such as Flask, Flask-Smorest, Migrate (from Flask-Migrate), JWTManager (Flask-JWT), Secrets, Passlib.hash, SQLAlchemy (from Flask-SQLAlchemy), Flask.Views, and Schemas. For testing, Insomnia and Postman were employed. Plus, Swagger was employed for documentation and other APIs have been integrated.
- **For the frontend and deployment:** HTML, CSS, and JavaScript were the languages used. Gunicorn was the library of choice for server deployment. Render, ElephantSQL, and pgAdmin4 (PostgreSQL) were tools implemented for deployment and production database manipulation.

ACKNOWLEDGEMENTS

During this project, I developed my first API and first dynamic web service. It was an amazing chance to work in different areas: API and web development (backend and frontend)/ data manipulation/ database management/ Security. I have also had the chance to implement several tools and programming languages. Many of them I haven't had the chance to employ in my previous projects. I am grateful to our professor Dr.Montassar for all the new skills I have learned thanks to this class and to this project.

CONTENTS

LIST OF FIGURES

1. MOTIVATION

This project is intended to tackle political concerns. With what is happening lately, the first thing that crosses someone's mind is the several military conflicts that are happening around the world. We have witnessed shocking pictures. Thousands of citizens are being killed, and cities are being destroyed. (Fig. 1.1)

Figure 1.1: War Effect.

With the images of chaos and destruction, a pertinent question arises: How can a country/city build itself after this kind of destructive war? .

Historically, several countries have witnessed such catastrophic events and needed international financial support for reconstruction. We can mention Germany, the WW2; Kuwait after the Gulf War; and Bosnia and Herzegovina after the Bosnian War. In each case, the international community, through organizations like the United Nations and the World Bank, provided financial aid that targeted infrastructure, housing, and economic recovery..

It’s crucial to note that this kind of international support isn’t exclusive to post-war scenarios but extends to various circumstances. COP, as an example, is the annual meeting where countries, that are members of the United Nations Framework Convention on Climate Change (UNFCCC), meet together. They address global climate issues and the financial support they can provide to certain developing countries lacking resources. This year, during the COP28, UAE for instance announced that it would invest thirty Billion dollars. Norway said it would donate fifty million dollars to Brazil’s Amazon Fund.

International financial funds have always been a way to help countries overcome the critical periods they go through and work on rebuilding themselves or addressing resource shortages.

This report is a detailed technical report of an API that is intended to effectively manage international Funds. With all the hope that all the conflicts will end soon.

2. DATABASE STRUCTURE

2.1. OVERALL ARCHITECTURE

LOSS AND DAMAGE FUNDS is now a publicly available web service. Its backend is built using the REST API library Flask in Python. For the front end, a very simple and user-friendly interface was built with HTML/CSS/Javascript. Users, specifically members of the funds' committee, can access the link https://lossanddamagefunds.onrender.com/ register to the platform, and then log in (we can restrict the signup/login for specific users to keep the platform restricted to committee members and maintain exclusivity). Successful login will lead to a Menu where they can choose the manipulation that needs to be done. Users can either add donors to the list of countries that will donate or add funds by specifying the necessary information. The backend of our API provides a range of additional functionalities that can be seamlessly integrated into our front end as needed.

2.2. DATABASE STRUCTURE

2.2.1. Development database

The API is implemented based on five different databases:

- **Users:** Members of the Loss and Damage Funds committee that will be allowed to manage the funds
- **Donors:** List of countries that will contribute and provide financial aid
- **Affected countries:** List of countries that will receive the funds
- **Funds:** List of funds (with value, type)

- **Funds-Donors:** List of the relationship between the donors and the funds

There are two types of relationships (Fig. 2.1):

1. **Relationship between affected countries and funds:** One to many. A country can have multiple funds, but a fund is intended for one specific country (to avoid management confusion).
2. **Relationship between the list of donors and funds:** Many to many. Countries can give multiple funds, and a fund can be provided by multiple countries (e.g., a Fund provided by the EU).

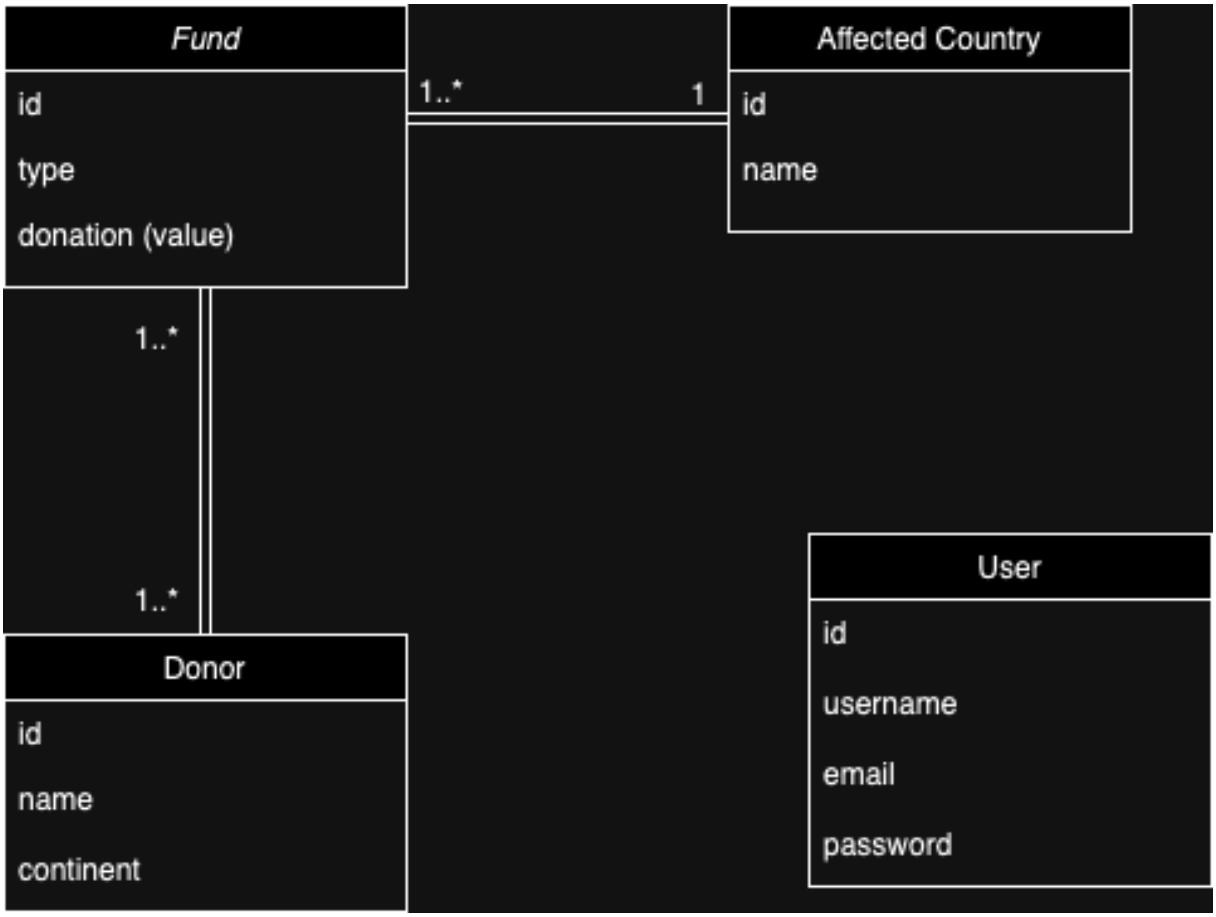


Figure 2.1: Diagram with Draw.io.

For development, the databases are structured using SQLAlchemy. It was used to define models for each database (SQLALchemy Models), store our data, and data manipulate or query the data.

2.2.2. Database Migration

For database migration and to be able to change the structure of our models without the need to rebuild our tables, Alembic is used. With this library, it is possible to 'migrate' the changes (flask db migrate), and then upgrade the database (flask db upgrade) to update the tables. All the versions of the migrations that have been made, are automatically saved in a folder called 'migrations'. This way it is also possible to downgrade to a previous version.

2.2.3. Production Database

- **ElephantSQL PostgreSQL:** For production, ElephantSQL is employed. It is cloud cloud-hosted database that is free and easy to manipulate. It enables us to deploy PostgreSQL databases in the cloud without managing the underlying infrastructure. Now, when the app runs, it gets initiated by accessing the link of ElephantSQL which is defined as environment variable.
- **PgAdmin 4:** It is an open-source database management tool for PostgreSQL databases. It provides a user-friendly interface that facilitates data querying, and manipulation. It was also employed for data visualization as shown in the appendix A section.

3. API ENDPOINTS, TESTING AND DOCUMENTATION

3.1. API ENDPOINTS

3.1.1. Users

- `"/users"`: GET a list of all users.
- `"/register"`: POST to Register a new user.
- `"/login"`: POST to Authenticate and authorize the user.
- `"/refreshtoken"`: POST to Refresh authentication token.
- `"/user/<int:user_id>"`: GET or DELETE user based on their ID.

3.1.2. Donors

- `"/donor"`: GET a list of all donors or POST a new one.
- `"/donor/<string: name>"`: HEAD/GET (check if the donor exists/ get the ID of a donor based on its name).
- `"/donor/<int:donor_id>"`: GET/PUT/DELETE donor information.
- `"/UpdateContinent/<int:donor_id>"`: PATCH update donor's continent.

3.1.3. Affected Countries

- `"/affected"`: GET a list of all affected countries or POST a new one.

- `"/affected/<string: name>"`: GET the ID of an affected country based on its name.
- `"/affected/<int:affected_id>"`: GET/DELETE affected country information.

3.1.4. Funds

- `"/funds"`: GET a list of all funds (with donors and affected countries associated with them).
- `"/fund/<int:affected_id>"`: GET a list of funds for a specific affected country or POST a new fund for a specific country.
- `"/donor/<int:donor_id>/fund/<int:fund_id>"`: POST to link a fund to a donor or DELETE the established link.
- `"/funds/<int:fund_id>"`: GET/DELETE a fund based on its ID.
- `"/fundtype/<int:fund_id>"`: PATCH to modify a fund's type.

Notes

- **PATCH method:** The PATCH method is used to apply partial modifications to a resource. Key differences between PATCH and PUT:
  - PUT replaces the entire resource, while PATCH applies partial updates.
  - PUT is for partial updates and is typically used for full replacements.
  - PATCH is more when only specific fields need updating.
- **HEAD:** The primary purpose of the HEAD method is to retrieve metadata about a specific resource identified by a URI without the actual content or to check if it actually exists. In this API, the HEAD method was used to check if a certain donor country exists in the database.

3.2. HTTP STATUS CODES USED

1. **2xx Success:**
   - **200 OK:** The request was successful.

- **201 Created:** The request was successful, and a new resource was created.

2. **4xx Client Errors:**

   - **400 Bad Request:** The server could not understand the request.
   - **401 Unauthorized:** The request requires user authentication.
   - **403 Forbidden:** The server understood the request, but it refuses to authorize it.
   - **404 Not Found:** The requested resource could not be found on the server.
   - **405 Method Not Allowed:** The method specified in the request is not allowed for the resource identified by the request URI.

3.3. TESTING WITH INSOMNIA

Insomnia is the tool used for testing the API endpoints. Four separate folders are established for each database and its endpoints.

Environment variables are used to define the URL (localhost then after deployment switched to render.com URL). Plus, it is used to get the access token as a variable from the login endpoint and provide it to the other secured endpoints.

3.4. DOCUMENTATION VIA SWAGGER

Swagger, now known as the OpenAPI Specification, was used for documenting our API. It describes our RESTful APIs and offers clear guidelines on endpoints, request/response formats, and usage examples. You can access it at https://lossanddamagefunds.onrender.com/swagger-ui.

4. IMPLEMENTATION OF OTHER APIS

- We started by implementing Restcountries API that gives the list of all the countries. This list was supposed to be the default list for donors. Yet, the list was huge so we removed it for easier manipulation. If the list of donors is limited, it would be easier to create it by ourselves. Yet, if it is long, the implementation of this API would be useful. The relevant code for its implementation is kept in the Appendix B section.

- Implementation of Mailgun API to send a confirmation email once the user is registered (Fig. 4.1). Mailgun API is a free API, and simple to integrate. It is possible to add restrictions so that only users with specific emails can register (customize the emails for the users of this API) Yet, this API is limited With the free tier, it takes time to send a confirmation email. Also, there is a limit to the number of emails sent.

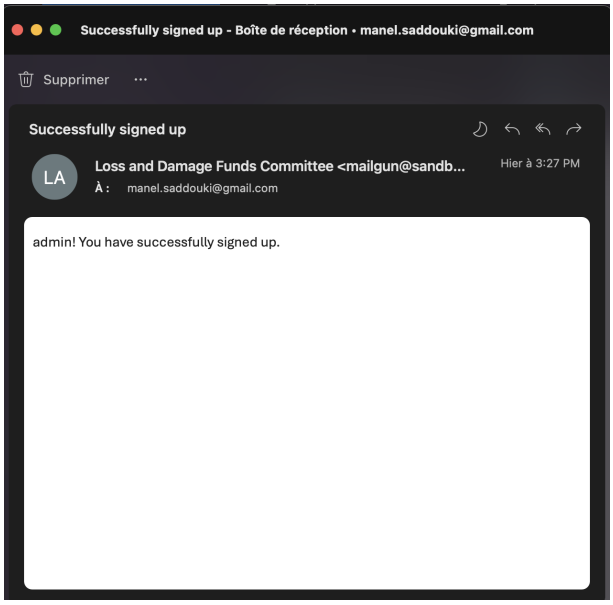


Figure 4.1: Registration Email.

5. FRONTEND IMPLEMENTATION - USER INTERFACE (UI)

FRONTEND TECHNOLOGIES

5.0.1. HTML (Hypertext Markup Language)

HTML was used to define the structure and layout of the web pages. It provides the foundational markup for content.

5.0.2. CSS (Cascading Style Sheets)

CSS played a crucial role in styling and visual presentation. It was utilized to enhance the aesthetics and overall appearance of the web pages.

5.0.3. JavaScript

JavaScript was employed to make the web service dynamic and interactive. It facilitates user interactions and links the frontend UI with the backend API code.

METHODOLOGY OF THE IMPLEMENTATION

To facilitate front-end development, publicly available HTML and CSS templates were used. These templates were adapted to meet the specific requirements of the API.

Each web page has its dedicated HTML, CSS, and JavaScript code. This approach simplifies maintenance and enhances code organization.

Flask's `render_template` function was used to dynamically render HTML templates within the Flask application. This allows for the integration of frontend components.

6. API DEPLOYMENT WITH RENDER

The API is supposed to be deployed via Heroku. However, since November 28, 2022, Heroku no longer offers its users a free tier.

The alternative was Render. The deployment was free and simple. The web service is created by linking it to the ElephantSQL and by configuring the environment variables.

While we used the Flask development server during development for quick testing and debugging, Gunicorn is preferred for production deployments. That’s why we have used Gunicorn for better stability, and performance during the production phase.

Render provide also full security. It automatically provides SSL/TLS certificates. Yet the limitation that has been noticed is that, With the free tier, the website shuts down after a period of inactivity.

Now, the web server is live and publicly available! Here are some examples of how it looks like:

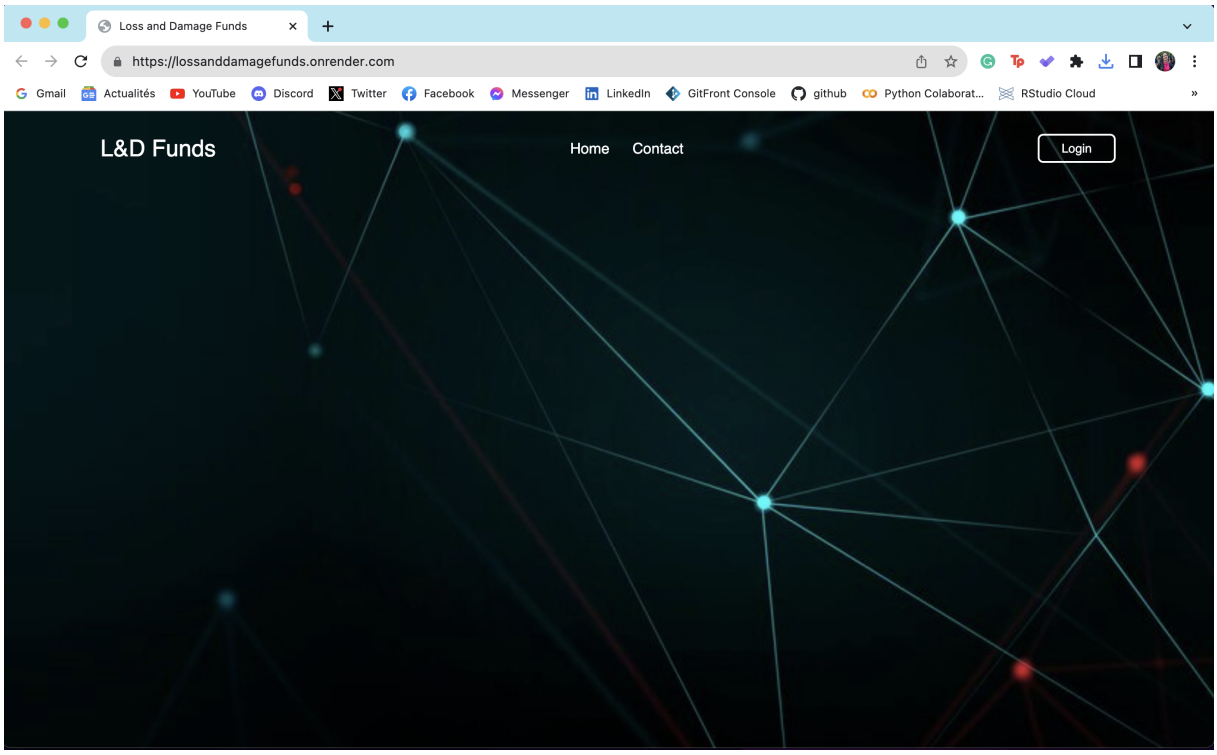


Figure 6.1: Main Page.

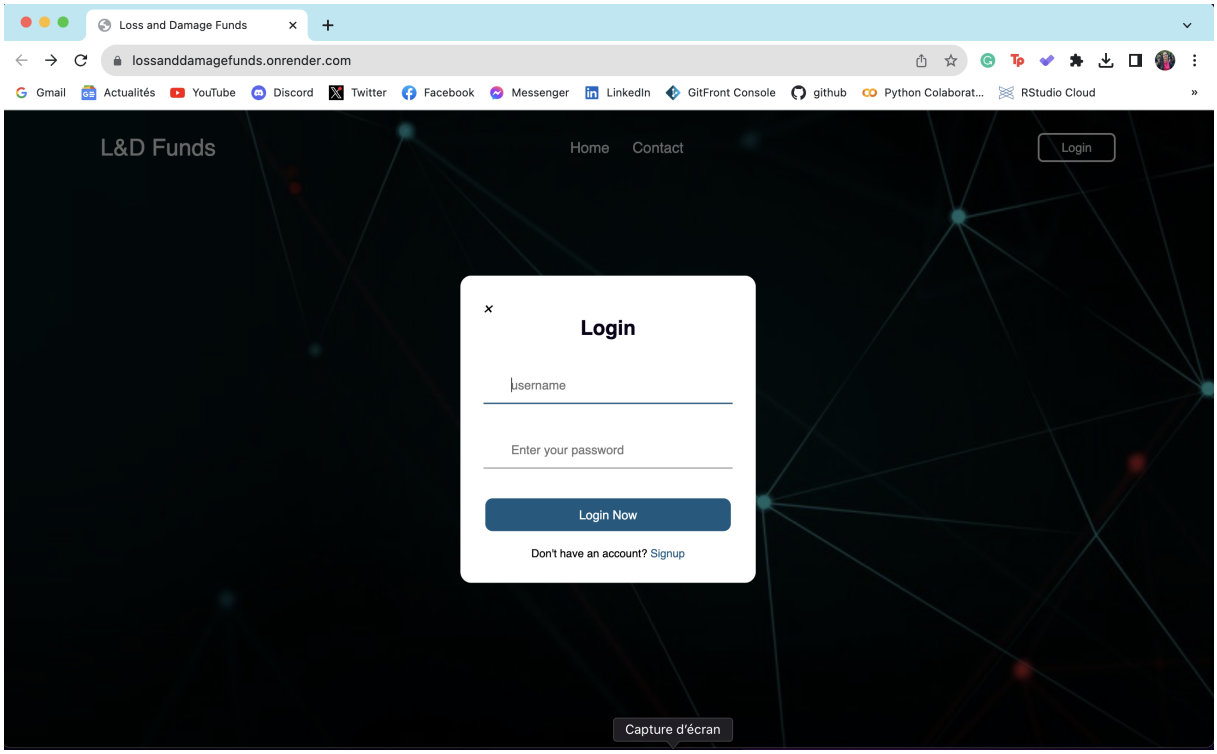


Figure 6.2: Login and Signup Sections.

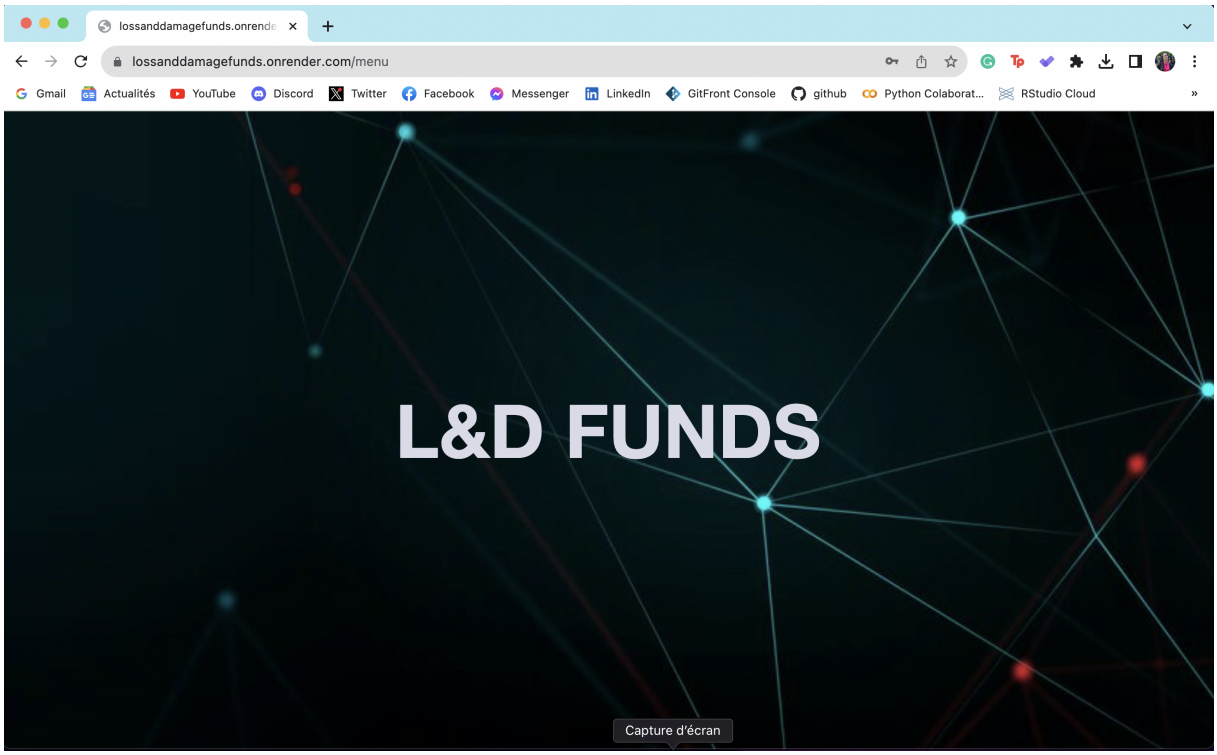


Figure 6.3: Main Menu.

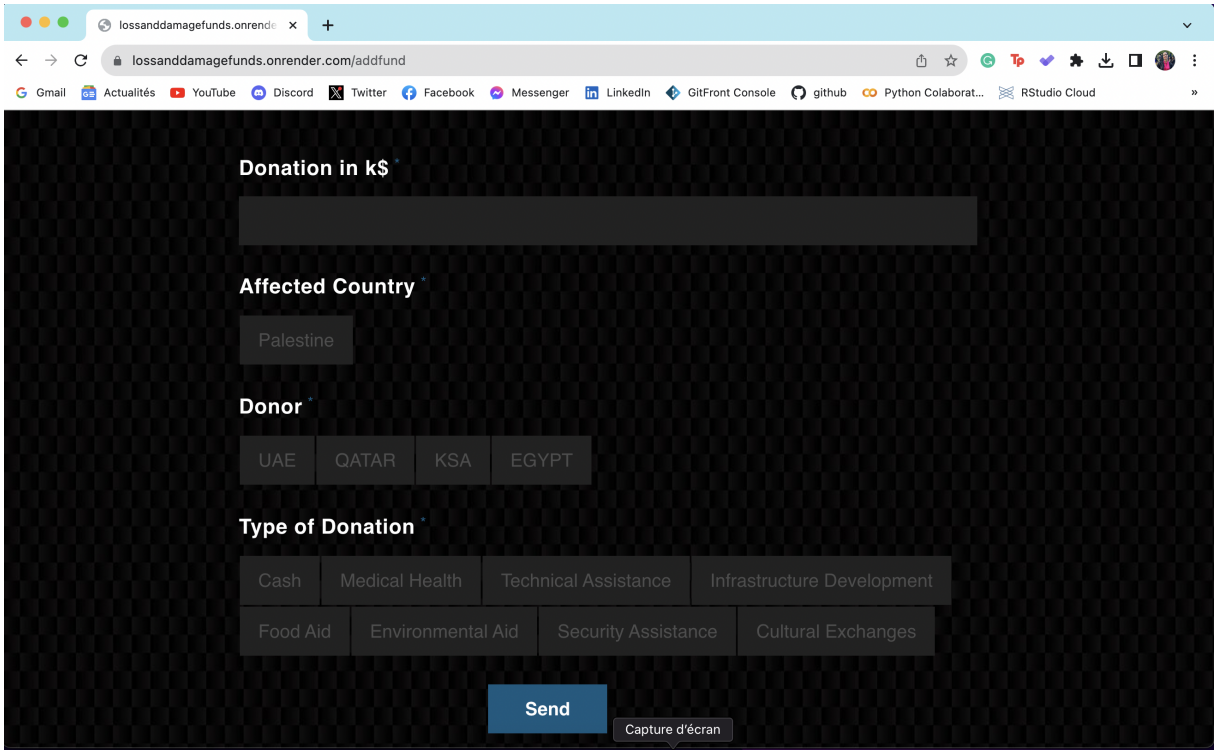


Figure 6.4: Add Fund Section.

7. SECURITY MEASURES

7.1. PASSWORD HASHING

Utilized the passlib.hash library to hash user passwords securely. So when a new user registers, only the hashed password is saved in the database.

7.2. ENDPOINT SECURITY: ACCESS AND REFRESH TOKENS

Secured endpoints using JSON Web Tokens (JWT) to generate access tokens and refresh tokens upon successful user login.

Configured the expiration time for access tokens to 30 minutes and refresh tokens to 30 days.

Users are provided with an access token each time they successfully log in. If the access token expires (and refresh token usage is allowed for that endpoint), another endpoint is invoked to create a non-fresh token for the user, allowing authentication without their intervention.

After 30 days (expiration date for refresh tokens) or when a fresh token is required, users must re-authenticate.

7.3. HTTPS: SSL/TLS WITH RENDER

HTTPS (Hypertext Transfer Protocol Secure) is the secure version of HTTP, and it uses SSL/TLS to encrypt data transmitted between a web browser and a web server.

Render automatically supplies SSL/TLS certificates for domains and manages their renewal. HTTPS is enabled by default, encrypting communication between users and the server.

8. CONCLUSION AND FUTURE IMPROVEMENTS

8.1. CONCLUSION

Our web service is now live. This API can be used for different purposes. It can even manage the funds collected by non-profit organizations. The prototype is implemented for a political concern, yet it can also be useful for any charity event.

8.2. FUTURE OUTLOOK

Future improvements for the Loss and Damage Funds API may include:

- Implement JWT in the frontend
- Improve the frontend; Add other sections that may be needed by the user
- Improve the format of the registration email
- Add logout section

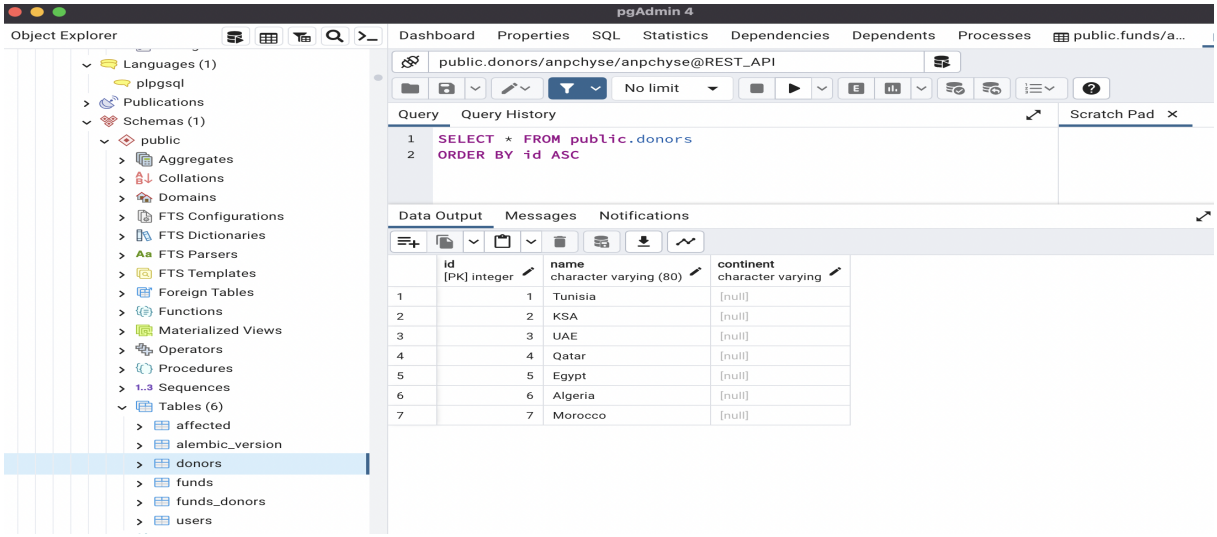
REFERENCES

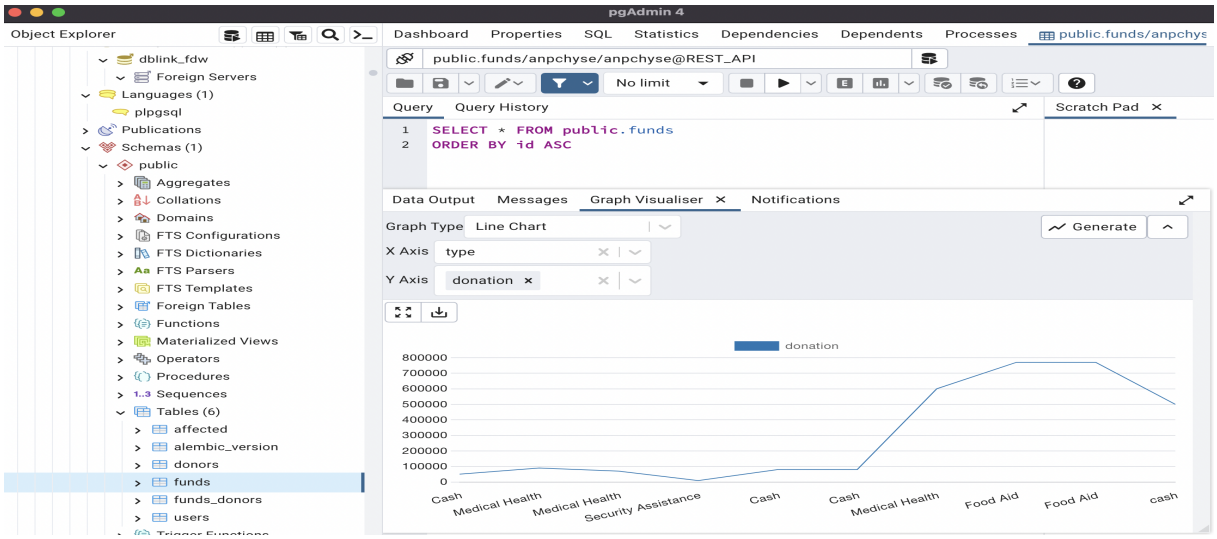
1. Flask Migrate:
    - https://flask-migrate.readthedocs.io/en/latest/
2. APIs Integration:
    - https://restcountries.com/#rest-countries
    - https://rapidapi.com/ajayakv/api/rest-countries/
    - https://documentation.mailgun.com/en/latest/api_reference.html#mailgun-api-reference
3. JWT:
    - https://flask-jwt-extended.readthedocs.io/en/stable/basic_usage.html
    - https://flask-jwt-extended.readthedocs.io/en/stable/refreshing_tokens.html
4. Templates used in HTML/CSS:
    - https://www.codingnepalweb.com/website-login-registration-form-html-css-javascript/?fbclid=IwAR35KmmM2UtcS_1xtJrh4fs3cp6frWSyKiqLflxwu__l16xc9CswiKuPUBQ
    - https://codepen.io/cassandraPaige/pen/BayGLaY
    - https://codepen.io/LewisN/pen/ojJeVO/
    - https://codepen.io/mican/pen/dRWxZe?editors=1111

APPENDIX

A. APPENDIX A:

In Section 2.2.3, we discussed how PgAdmin4 can be used for data querying and visualization:





B. APPENDIX B:

In Section 4, we discussed the implementation of the Restcountries API. Here is the relevant code: :

```python
@blp.response(201, DonorSchema)
def post(self):
    # Make a request to Restcountries API to get the list of country names
    restcountries_url = "https://restcountries.com/v3.1/all"
    try:
        response = requests.get(restcountries_url)
        response.raise_for_status()  # Check for HTTP errors
        countries_data = response.json()
        country_names = [country['name']['common'] for country in countries_data]
    except requests.exceptions.RequestException as e:
        abort(500, message=f"Failed to fetch country names from Restcountries API: {str(e)}")

    # Add the country names to the Donor database
    added_countries = []
    for country_name in country_names:
        if not DonorModel.query.filter(DonorModel.name == country_name).first():
            donor = DonorModel(name=country_name)
            db.session.add(donor)
            added_countries.append(country_name)

    # Commit the changes to the database
    db.session.commit()

    return {"message": f"Added countries: {', '.join(added_countries)}"}, 201
```