Faculty of Computer Science and Engineering - Skopje Implementation of Free and Open Source Systems

PROJECT REPORT

**Professor: Student:** Boban Joksimoski Magdalena Petrushevska

June 2022

1. **Purpose and Scope**

This project is designed for the purpose of the course Implementation of Free and Open Source Systems under the Faculty of Computer Science and Engineering, UKIM 2021/2022

The goal was to create an example site using the FastAPI development framework in Python.

The specific topic that I chose to work on is mental health app.

This mental health app aims to help their users easily cope with stress, low mood, depression, and anxiety. To support this I develop 3 key features that will be provided to the users. These are mood diary, mental health forum and medication reminder. Every of these features will be discussed in more details in the following sections.

1. **Features provided by the app**

As it was mention before, in this project I worked on developing 3 main features for mental health app. These are mood diary, mental health forum and medication reminder.

* 1. Mood diary

Mood diary

Allows the user to track his mood constantly by adding on daily basis the record of how does he feels and also can describes the reason why he feels that way. The user has the opportunity to export pdf document containig the mood history of the user. The user can see all the moods records. He can add at any time a new record related to his mood. Features related to the mood diary are only accessible to authenticated users.

* 1. Мental health forum

Allows users to share posts through which they can share their problems, help each other in times of crisis, to share experiences, etc. Every user will be able to start a new forum topic. Anyone can write a comment on that topic. The user can give thumbs up on the comment he likes. Anyone who has written the comment can delete it, edit it. All users are able to see all the posts. But only the authenticated users are able to comment and gives thumps up. Every post owner can see its own posts and appropriately manages it, since he can can delete it, edit it. There is no limit on how many topics a particular

* 1. Medication reminder
  2. As app feature, medication reminder is designed to be used by users that have the need of taking medication on regular basis. This feature allows users to receive SMS messages at the scheduled time.That way they can be sure they never forgot to take appropriate medication. This means that user has the opportunity to create new reminder through filling a form by related data such as the period duration in which he needs to take the medication. Also it required information about the time and quantity of medication that the user should takes. The users are able to add new reminders for as many as needed. Which means there is no limitation of number of reminder that user can make. If the user doesn’t need anymore any of the reminders, he can cancel it by simple deleting it. Also if there are any changes that needs to be make related to a specific reminder, the edit option is available to users. The system is configured so it detects any obsolate reminders, then it simply removes it. That way if the user forgets to delete it, the system takes care of it. Also the system is configured so it can send SMS messages when at the scheduled time to remind the user to take the medication. This process of sending SMS messages is done through integration with Twilio, which is a communcation API for sending SMS. Every SMS message contains the name and the quantity of medication that is needed.

1. **INTERACTIVE FASTAPI DOCS**

If we start the app and search in the browser for localhost:8000/docs, what we will see is the interactive API documentation for our backend. To get the sense of what we are talking, in continuation we can take a look at few screenshots related to that documentation.

**// todo: Slike of docs interative doc**

This documentation is automatically generated as we go along with adding code and new https methods. So each time we add new http method in FastAPI, accordingly the new method is added to the this interactive API documentation. This features of FastAPI is provided by [Swagger UI](https://github.com/swagger-api/swagger-ui).

There is also an alternative API documentation of this, which is provided by [ReDoc](https://github.com/Rebilly/ReDoc) and can be accessed in the browser on localhost:8000/redoc. These two API automatic documentations are pretty much similar.

1. **HOW TO TEST THE APP**

In order to see how the app works in practice, you should download the repo from the given [link](https://gitlab.finki.ukim.mk/181541/mental-health-app). Once it is downloaded, unzip it. Then open the backend and frontend directories in separete windows in your IDE of preference. I work in Visual Studio Code and below are the steps needed to run the app.

Important thing to note here is that the app is not going to function properly if we do not have setup a database. I used PostgreSQL for this particular project, so I will show you the steps needed to create a new database in PostgreSQL.

Note here that I have locally intalled the PostgreSQL database and psql command line.

Open the psql command line, enter the required credentials and then write:



This will create a new database for you.

Next, open the terminal in VSC and run the following commands:



This is needed in order to enter the virtual environment. The reason why we need it is for us to use an isolated **environment** for Python projects. There is already created virtual environment contained in the backend directory so there is no need for you to create new one. Your taks is just to activate it in order to be able to use it.

This virtual environment is used **to manage Python packages for different projects**. Using virtual environement allows you to avoid installing Python packages globally which could break system tools or other projects.

Now in order to create all the needed tables in the database we need to run the code from the file init\_db.py.



This will create all the defined tables based on the models that we crete in the app. If everything is okey so far, you should have the tables created in the database.

Next, run the following command to start the server.



Now you should be able to access the app from your browser.

Next we need to connect also to the frontend part. As we mention before open the frontend directory in separate window in VSC, then open the terminal and run the following:



Now you can chech the user interface for mental health app.

If you download everything, then there should not be a need to install any other dependencies in order to run the app.

**RUNNING SERVER**

The main thing you need to run a **FastAPI** application in a remote server machine is an ASGI server program. The ASGI server **that I used for this project is** uvicorn.

**AUTHENTICATION**

In order to provide user security, I implement authentication using JWT tokens.

**//TODO: DA SE DODADE SLIKA OD KOD STO E IMPLEMENTIRANO ZA AUTHENTICATION ZAKLUCNO SS FRONTEND I BACKEND**

**TECHNOLOGIES/CONCEPTS USED TO BUILD THE APP**

**FastAPI (backend)**

On backend I used FastAPI, modern, fast web framework for building APIs with Python.

**React (frontend)**

On frontend I used React, JavaScript library for building user interfaces.

**PostgreSQL (database)**

The app data is stored in PostgreSQL database, which is powerful, open source object-relational database system.

**Twilio (communication APIs for SMS)**

Works in the background to bridge the gap between web based applications and telephones

**JWT (authentication)**

Open standard used to share security information between two parties.

**PyFPDF (PDF documents generation)**

Library for PDF document generation under Python

**Pydantic (Data validation)**

Pydantic enforces type hints at runtime, and provides user friendly errors when data is invalid.

**SQLAlchemy**

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.

It provides a full suite of well known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language.

**STUCTURE OF THE BACKEND APP**

**Graphical user interface, text, application, chat or text message

Description automatically generated**

The tree structure of the backend app can be seen at the screenshot above.

**\_\_pycache\_\_**

is a directory that contains bytecode cache files that are automatically generated by python

**env**

is a directory that represent the virtual environment that we use in this project. The reason why we need it is for us to use an isolated **environment** for Python projects. Before we can start the uvicorn server, we must activate this enviroment.

**mood\_history\_files**

The mood\_history\_files directory is used to store on the server the generated mood history files from its users.

**.env file**

The .env file is used to configure some environmental variables needed to implement integration with Twilio.

**Database.py**

Database.py file is used to configure the database to be used for storing data. Additionally using file init\_db.py we create the needed tables in the database**.**

**init\_db.py**

We must run the code from this particular file in order to create all the needed tables in the database. This can be done while running the following command in the VSC terminal:



If we do not run this file, the server will start but it will not be able to retrieve, add, remove or update any data in the database because they basically do not exist.

**Main.py**

Main.py is the file where the entire backend logic lies.

**models.py**

Our models are kept in models.py. The models that we have are: User, Post, Comment, MoodOptions, MoodHistory, Reminder. Every of these models have one uniques kewy called id. This way we can differenciate different instances from the same class. Also every model inherits from declarative\_base from sqlalchemy.orm. sqlalchemy.orm functions as object-relational mapper (ORM). **provides an object-oriented layer between relational databases and object-oriented programming languages without having to write SQL queries**. It allows the Python classes to be mapped as appropriate tables in the database.

The screenshot below shows us the way we create a model in FastAPI. Which is then used as a base to create the appropriate table in the database.

**//todo: slika od model za klasu**

**schemas.py**

The DTO classes are kept in schemas.py. These classes are like a **container to encapsulate data and pass it from one layer of the application to another**. In our case we use them to get the form data that are passed from the user interface to the backend, where we use them as a base to create new instances for our models. These classes inherit from pydantic.BaseModel. In this case pydantic allows auto creation of JSON Schemas from models.

**DATABASE CONNECTION**

JSON Schemas from m

As I mentioned before, I used PostgreSQL for storing data. Further I will show the process of connecting to the database.

First of all we need to create a new database. For this project I use locally installed PostgreSQL and psql terminal for running SQL commands.



In VSC open the terminal write the following:





Next we create new file named database.py.

In this file we set the ofiguration needed for the database. What we do here is to create new sqlalchemy engine and we set the DATABASE\_URL on which we will connect to the previously created database.

Text

Description automatically generated

Here the SessionLocal is used to to track Session objects and consider more explicit means of scoping when using application servers which are not based on traditional threads.

Next we create file named models.py and here we define all the classes, models that we are going to need for the app. Later this classes are going to be used as base upon which the tables are created in the database.

Next we create file named init\_db.py. We use this file to create all the tables needed in the database.

Graphical user interface, text, application

Description automatically generated

To do this we