Next Steps

As for the next steps in the upcoming semester, there are several crucial tasks that need to be completed in order to ensure the smooth functioning of the project. Here is a more detailed description of each task:

1. Enhancing the user authentication process: Currently, the process of signing up and logging in on the webpage is functional until the creation of an access token. The next step is to enable users to navigate to the questionnaire page seamlessly upon obtaining the access token. To achieve this, consider incorporating session cookies for FastAPI, which will help manage user sessions securely. Additionally, use the fastapi-login manager to access protected routes, such as the already implemented route (/questionnaire).
2. Restoring the database from the snapshot: A snapshot containing the AWS RDS database information has been shared with Professor Abhari. This snapshot must be utilized to restore the database in the AWS account that will be used for the project. Carefully follow the AWS documentation to ensure accurate restoration of the database, avoiding any loss of critical data.
3. Displaying survey results and comparison data: In the project's Git repository, there is a Python file designed to populate the database with dummy data. Once a user has completed a survey, they should be presented with their individual results compared against the normalized data. To achieve this, implement a feature that retrieves the relevant data from the database and displays it in a visually appealing and easily understandable format.
4. Improving the website's aesthetics: Lastly, it is essential to enhance the overall appearance and user experience of the website. This can be achieved by refining the CSS, focusing on aspects like layout, color schemes, typography, and responsive design. Make sure that the website is both visually appealing and user-friendly, ensuring a seamless experience for users across different devices and screen sizes.

By addressing these tasks, the project will be better equipped to provide a secure and engaging experience for users during the upcoming semester.

Database Information

PostgreSQL is a powerful and popular open-source relational database management system that can handle a wide range of workloads. Amazon RDS is a fully managed database service that makes it easy to set up, operate, and scale a relational database in the cloud.

In this documentation, you will find information on how to connect to your PostgreSQL database on RDS, how to manage your database and its resources.

**Connecting to the database**

To connect to your PostgreSQL database on Amazon RDS, you can use pgAdmin4, a popular open-source administration and management tool for PostgreSQL.

<https://www.postgresql.org/download/>

OVERVIEW

The database is hosted on Amazon Web Services (AWS) Relational Database Service (RDS) and comprises five tables: Users, Surveys, Questions, Answers, and Reports. The Users table stores user information such as user ID, name, age, gender, and education level. The Surveys table contains survey ID and survey name data. The Questions table, linked to the Surveys table through a foreign key relationship, maintains information on each question, including its ID, associated survey ID, and question text. The Answers table, which has foreign key relationships with both the Users and Questions tables, stores the user's responses to questions as normalized integers ranging from 1 to 7. Finally, the Reports table holds user-generated reports, including report ID, user ID, and report text. This database structure allows for the efficient organization, storage, and retrieval of user responses to survey questions and the generation of user-specific reports.

**CREATE TABLES IN SQL**

1. CREATE TABLE Users (

2.     User\_id INT PRIMARY KEY,

3.     name VARCHAR(255),

4.     age INT,

5.     gender VARCHAR(10),

6.     Education VARCHAR(255)

7. );

8.

9. CREATE TABLE Surveys (

10.     Survey\_id INT PRIMARY KEY,

11.     Survey\_name VARCHAR(255)

12. );

13.

14. CREATE TABLE Questions (

15.     Question\_ID INT PRIMARY KEY,

16.     Survey\_id INT,

17.     Question\_text TEXT,

18.     FOREIGN KEY (Survey\_id) REFERENCES Surveys(Survey\_id)

19. );

20.

21. CREATE TABLE Answers (

22.     Answer\_id INT PRIMARY KEY,

23.     User\_id INT,

24.     Question\_id INT,

25.     Answer INT,

26.     FOREIGN KEY (User\_id) REFERENCES Users(User\_id),

27.     FOREIGN KEY (Question\_id) REFERENCES Questions(Question\_ID)

28. );

29.

30. CREATE TABLE Reports (

31.     Report\_id INT PRIMARY KEY,

32.     User\_id INT,

33.     Report\_text TEXT,

34.     FOREIGN KEY (User\_id) REFERENCES Users(User\_id)

35. );

36.

 INSERT QUESTIONS

1. INSERT INTO Questions (Survey\_ID, Question\_Text) VALUES (1, 'Improve my skills');

2. INSERT INTO Questions (Survey\_ID, Question\_Text) VALUES (1, 'Gain new knowledge/expertise');

3. INSERT INTO Questions (Survey\_ID, Question\_Text) VALUES (1, 'Test my talent/capabilities');

4. INSERT INTO Questions (Survey\_ID, Question\_Text) VALUES (1, 'Keep up with new ideas and innovations');

5. INSERT INTO Questions (Survey\_ID, Question\_Text) VALUES (1, 'Come up with new ideas');

6. INSERT INTO Questions (Survey\_ID, Question\_Text) VALUES (1, 'Have a pleasurable work experience');

7. INSERT INTO Questions (Survey\_ID, Question\_Text) VALUES (1, 'Have fun at work');

8. INSERT INTO Questions (Survey\_ID, Question\_Text) VALUES (1, 'Enjoy what I am doing');

9. INSERT INTO Questions (Survey\_ID, Question\_Text) VALUES (1, 'Feel more satisfied with my work');

10. INSERT INTO Questions (Survey\_ID, Question\_Text) VALUES (1, 'Have better interactions with my colleagues');

11. INSERT INTO Questions (Survey\_ID, Question\_Text) VALUES (1, 'Expand my personal/social network');

12. INSERT INTO Questions (Survey\_ID, Question\_Text) VALUES (1, 'Strengthen my affiliation to my organization');

13. INSERT INTO Questions (Survey\_ID, Question\_Text) VALUES (1, 'Feel I belong to my organization');

14. INSERT INTO Questions (Survey\_ID, Question\_Text) VALUES (1, 'Meet others with whom I share similar interests at work');

15. INSERT INTO Questions (Survey\_ID, Question\_Text) VALUES (1, 'Make a good impression on my colleagues');

16. INSERT INTO Questions (Survey\_ID, Question\_Text) VALUES (1, 'Make a better contribution');

17. INSERT INTO Questions (Survey\_ID, Question\_Text) VALUES (1, 'Improve the impact of my work');

18. INSERT INTO Questions (Survey\_ID, Question\_Text) VALUES (1, 'Control the quality of my work');

19. INSERT INTO Questions (Survey\_ID, Question\_Text) VALUES (1, 'Have positive impact on other's work');

20.

AWS CONNECTION

This is used to connect to the database which is hosted on RDS from within the main.py file.

1. # Connect to the database

2. conn = psycopg2.connect(

3. host='mis-database.cevtznumxb4e.us-west-1.rds.amazonaws.com',

4. port='5432',

5. database='mis-database',

6. user="postgres",

7. password="password123"

8. )

9.

FASTAPI

STATIC

To serve static files such as CSS, JavaScript, and images, FastAPI relies on the Starlette framework's StaticFiles class. Static files are essential for styling and enhancing the look and feel of web applications. FastAPI allows developers to host these files by setting up a designated directory, often named "static", to store and serve these resources. To configure static file serving in FastAPI, developers must import the StaticFiles class from fastapi.staticfiles and mount it to the application instance using the app.mount() method. For our purposes this folder contains the CSS for our HTML pages found in the templates folder

TEMPLATES

FastAPI allows developers to leverage templates, such as those based on the Jinja2 template engine, to create dynamic HTML pages that integrate with backend APIs seamlessly. The templates are separate files with placeholders for data that can be filled in with information received from the backend, enabling the generation of customized content for the user.

MAIN.py

The base URL for this API is http://localhost:8000.

To sign up for the application, users can navigate to /usersignup and fill out the required fields: email, password, name, date of birth, and gender. Once submitted, a new user account will be created.

To log in to the application, users can navigate to /login and enter their email address and password. If the credentials are correct, the user will receive an access token that they can use to access protected resources.

The /protected endpoint is only accessible to authenticated users. To access this endpoint, users must provide their access token in the Authorization header of the request.

Users can fill out the questionnaire by navigating to /questionnaire and submitting their responses to the 19 questions. The questionnaire form also requires the user to provide their first and last name, age, gender, occupation, education, and email address.

The API uses OAuth2 authentication and the pyjwt library to handle access tokens. The fastapi-login library is used to handle user authentication in FastAPI.

The API stores user information and survey responses in a PostgreSQL database. The users table stores information about each user, including their name, age, gender, occupation, education, and email address. The survey\_response table stores the responses to the questionnaire for each user.

Dependencies used by this API include fastapi, psycopg2, bcrypt, pyjwt, fastapi-login, Jinja2, and uvicorn.

To run the API, users should install the required dependencies by running pip install -r requirements.txt in their terminal. Users should then create a PostgreSQL database and update the conn variable in main.py to match their database connection details. The API can then be started by running the command **uvicorn main:app --reload.**

AWS Services

RDS

Amazon Relational Database Service (RDS) is a web service that makes it easy to set up, operate, and scale a relational database in the cloud. RDS provides a managed database service for various popular database engines, such as MySQL, PostgreSQL, Oracle, and Microsoft SQL Server.

In this API, we are using RDS to host a PostgreSQL database. The RDS instance is hosted on Amazon Web Services (AWS) and is accessed through the API using the psycopg2 library.

The conn variable in main.py contains the connection details for the RDS instance, including the host, port, database name, user, and password. When the API needs to interact with the database, it creates a connection using these details and executes SQL queries using a cursor. Once the queries are executed, the API commits the changes to the database and closes the connection.

By using RDS, we can take advantage of AWS's expertise in managing and scaling relational databases. This allows us to focus on developing our API without worrying about the underlying infrastructure. Additionally, RDS provides features such as automatic backups, automatic software patching, and easy scaling, which can help us to keep our database secure, up-to-date, and performant.