On Monday at 10 AM, our team met to work on the sign-in and sign-up backend as planned in our previous discussion. We started by ensuring that every team member had the necessary tools installed, including:

* **Visual Studio Code**
* **Python**
* **Django Framework**
* **Docker**

Once the work environment was set up for everyone, we moved on to defining the database models required for user authentication. The project was initially configured to use **PostgreSQL**, but due to local compatibility issues, we switched to **SQLite** for development. We agreed on the following attributes:

* id (Primary Key)
* first\_name
* last\_name
* uwe\_email (restricted to UWE domain, unique)
* date\_of\_birth
* gender
* username (unique)
* signup\_time
* last\_signin\_time
* role (user/administrator)
* password (hashed for security)

We then discussed and implemented the necessary functions for authentication:

* **Email Verification**: Ensures only UWE emails can be used for sign-up.
* **Password Hashing**: Securely stores user passwords.
* **One-Time Password (OTP) via Email**: An OTP is sent to the user’s registered UWE email for verification before account creation.
* **OTP Expiration**: The OTP expires after a set time to prevent misuse.
* **Google App Password Setup**: Since we needed to send emails from a system account, we had to configure a one-time app password through Google, which required additional research.
* **Secure OTP Storage**: OTPs are stored in a hashed format to prevent unauthorized access.
* **Uniqueness Check**: Prevents duplicate usernames and emails.
* **Password Reset Functionality**: Allows users to reset forgotten passwords.
* **Error Handling for User Creation Before OTP Verification**: Initially, users were being added to the database before completing OTP verification, so we modified the flow to ensure they were only added after successful verification.

For the user flow, we structured it as follows:

1. The user selects either **Sign In** or **Sign Up**.
2. If signing up:
   * The user enters their details.
   * An OTP is sent to their UWE email for verification.
   * Upon successful OTP validation, the user is registered and saved in the database.
   * The user can now sign in.
3. If signing in:
   * The system verifies the provided credentials.
   * If valid, the user is granted access.
   * If invalid, an error message is displayed.

**Challenges Faced:**

* **Attendance Issues**: Some group members had other commitments on the meeting day, which slowed progress. As a result, parts of the work had to be completed individually at home.
* **Database Change**: The initial plan to use PostgreSQL had to be abandoned due to local setup issues, requiring adjustments to use SQLite.
* **Google App Password Requirement**: Setting up email sending required configuring a one-time app password in a Google account, which was unexpected and required additional research.
* **User Being Added Before OTP Verification**: Initially, users were registered before verifying their OTP. We adjusted the logic to ensure users were only stored after completing OTP verification.
* **OTP Expiration Timing**: Implementing a secure expiration mechanism was necessary to ensure OTPs couldn't be reused indefinitely.
* **Storing OTP Securely**: OTPs had to be stored in a hashed format instead of plaintext to prevent security vulnerabilities.

**Test Cases:**

To ensure system reliability, we implemented the following test cases:

1. **Successful Sign-Up**: Verify that a user can sign up with a valid UWE email and correct OTP.
2. **Sign-Up with Invalid Email**: Ensure non-UWE emails are rejected.
3. **Duplicate Email/Username Check**: Test that existing emails and usernames cannot be used again.
4. **OTP Expiration**: Verify that an expired OTP cannot be used.
5. **OTP Mismatch**: Ensure that entering an incorrect OTP prevents registration.
6. **Password Hashing**: Confirm that passwords are not stored in plaintext.
7. **Successful Sign-In**: Validate that registered users can log in with the correct credentials.
8. **Incorrect Sign-In**: Ensure wrong username or password prevents login.
9. **Password Reset**: Check that users can successfully reset their password via email verification.
10. **Database Consistency**: Ensure users are only saved in the database after OTP verification.

**Work Distribution:**

* **Two members** worked on implementing the authentication functions.
* **Two members** focused on setting up the database and integrating it with the backend.
* **One member** was responsible for error handling and writing test cases to validate functionality.

By the end of the session, we had successfully set up the foundational components of the authentication system, ensuring that user registration and login were functioning as expected. Our next steps will involve refining the user interface and performing additional testing to enhance reliability and security.