Backend Implementation (Python/Flask)-

**Setup and Routes**

- Firstly , we need to initialize Flask application and set up necessary routes.

- Then, will create new files in the backend for handling authentication and verification:

- Registration: ‘backend/routes/auth.py’

- Email Verification: ‘backend/routes/verify.py’

- Login: ‘backend/routes/login.py’

Database Model

- Then , will have to implement the User model with attributes including username, email, password\_hash, and is\_verified.

Registration Endpoint

- In ‘backend/routes/auth.py’, (newly created file) will handle user registration.

- In this, store user data with hashed password and a unique verification token.

Email Service

- Next, will implement an email sending utility in ‘backend/utils/email\_service.py’.

- Then, need to use an SMTP server or an email service like SendGrid to send verification emails.

Verification Endpoint

- In ‘backend/routes/verify.py’, will have to create an endpoint to handle the verification of the token received from the user's email.

Login Endpoint

- In ‘backend/routes/login.py’, will need to ensure users can only log in if their email is verified (‘is\_verified’ is ‘True’).

Frontend Implementation (React.js)

Registration Form

- Will utilize the existing form at ‘src/pages/auth/RegisterPage.js’.

Registration Handling

- On form submission in `RegisterPage.js`, will make an API call to the backend registration endpoint.

- Then , will display messages based on the response received.

Verification Page

- Create a new component ‘src/pages/auth/VerifyEmailPage.js’ to handle email verification.

- This component will interact with the backend's verification endpoint.

Login Form

- Use the existing form at ‘src/pages/auth/LoginPage.js’.

- Modify as necessary to interact with the backend's login endpoint.

Login Handling

- Manage user login through ‘LoginPage.js’, with proper handling of authentication status.

Email Verification Flow

1. User Registration: The user registers via ‘RegisterPage.js’.

2. Backend Processing: The backend receives the data, creates a user entry, and sends a verification email.

3. User Email Verification: The user clicks the verification link in their email, redirecting them to ‘VerifyEmailPage.js’.

4. Verification Confirmation: The frontend page confirms the token with the backend, which then marks the user as verified.

5. User Login: Post-verification, the user logs in through ‘LoginPage.js’.

**For Password reset –**

Implement Frontend Form Submission

- In ‘ResetPassword.js’, will need to add a method to handle form submission.

- This method will capture the user's email and send it to backend API.

Set Up Backend API for Password Reset Request

- Then , will need to create a new file in backend, ‘backend/routes/password\_reset.py’.

- will need to implement an endpoint ‘/request-reset’ to handle incoming password reset requests.

- This endpoint will validate the email, generate a secure token, and store it temporarily.

Send Password Reset Email

- In the same backend file, will implement logic to send an email to the user with a password reset link.

- This link will contain the secure token as a query parameter.

Create a Password Reset Page

- Will create a new React component, ‘PasswordResetPage.js’.

- This page will have a form where users can enter their new password.

- It will also extract the token from the URL.

Implement Backend Password Reset Logic

- In ‘password\_reset.py’,will add another endpoint ‘/reset-password’ to handle the actual password reset.

- This endpoint will verify the token and update the user's password.

Connect Password Reset Page with Backend

- In ‘PasswordResetPage.js’, will implement form submission logic to send the new password and token to the backend's ‘/reset-password’ endpoint.