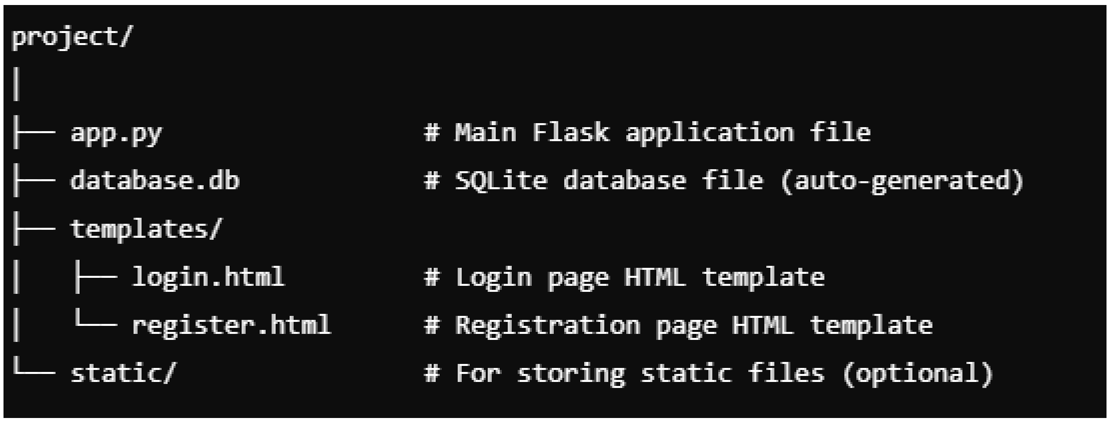
Authentication System Documentation

Overview

This is a simple Flask-based authentication system with user registration and login functionality. It uses SQLite for data storage and SQLAlchemy for managing the database and user-related operations. This system consists of user registration, login validation, and a basic dashboard display upon successful login.

Project Structure



```
project/
|
├── app.py                  # Main Flask application file
├── database.db             # SQLite database file (auto-generated)
├── templates/
|   ├── login.html          # Login page HTML template
|   └── register.html       # Registration page HTML template
└── static/                 # For storing static files (optional)
```

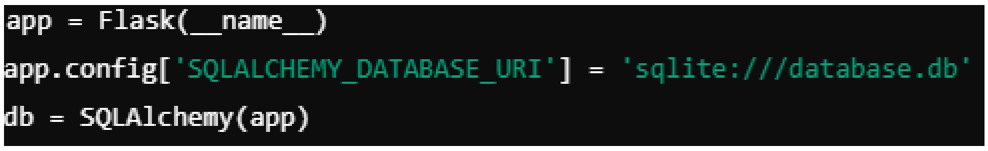
1. **Flask Application**: The main application file contains routes for user registration, login, and dashboard.
2. **Database Model**: The `User` class is the SQLAlchemy model representing users in the system, stored in a SQLite database.
3. **Routes**: There are multiple routes defined to handle the registration, login, and dashboard functionality.

---

1. Flask Application Configuration

The application uses **Flask** as the web framework and **SQLAlchemy** for database operations. Here's how the application is set up:

python



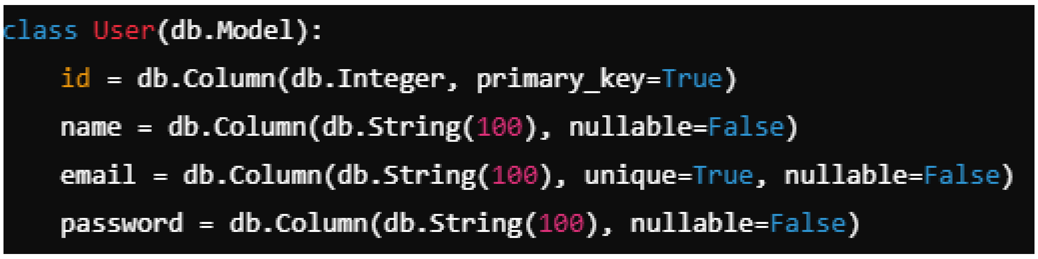
```python
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
db = SQLAlchemy(app)
```

`SQLALCHEMY_DATABASE_URI`: This is the URI used by SQLAlchemy to connect to the SQLite database. The database file is named `database.db` and is stored in the root directory.

`db`: This initializes the SQLAlchemy object which will be used to interact with the database.

2. Database Model: User

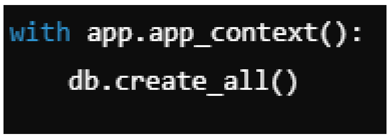
The application uses SQLAlchemy ORM to map the `User` class to a database table. The `User` model defines the structure of the user data stored in the database.



```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(100), unique=True, nullable=False)
    password = db.Column(db.String(100), nullable=False)
```

- `id`: This is the primary key for the user table and will automatically increment with each new user.
- `name`: A string field to store the user's name. It is required (non-nullable).
- `email`: A string field to store the user's email address. This must be unique for each user and cannot be null.
- `password`: A string field to store the user's password (non-nullable).

Creating the Database: To create the `users` table and other necessary tables, the following command is executed:

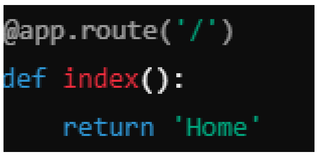


```
with app.app_context():
    db.create_all()
```

3. Routes and Their Functionality

a) Home Route (`/`)

The home route is the entry point for the application. When a user visits the root URL (`/`), this route is triggered.

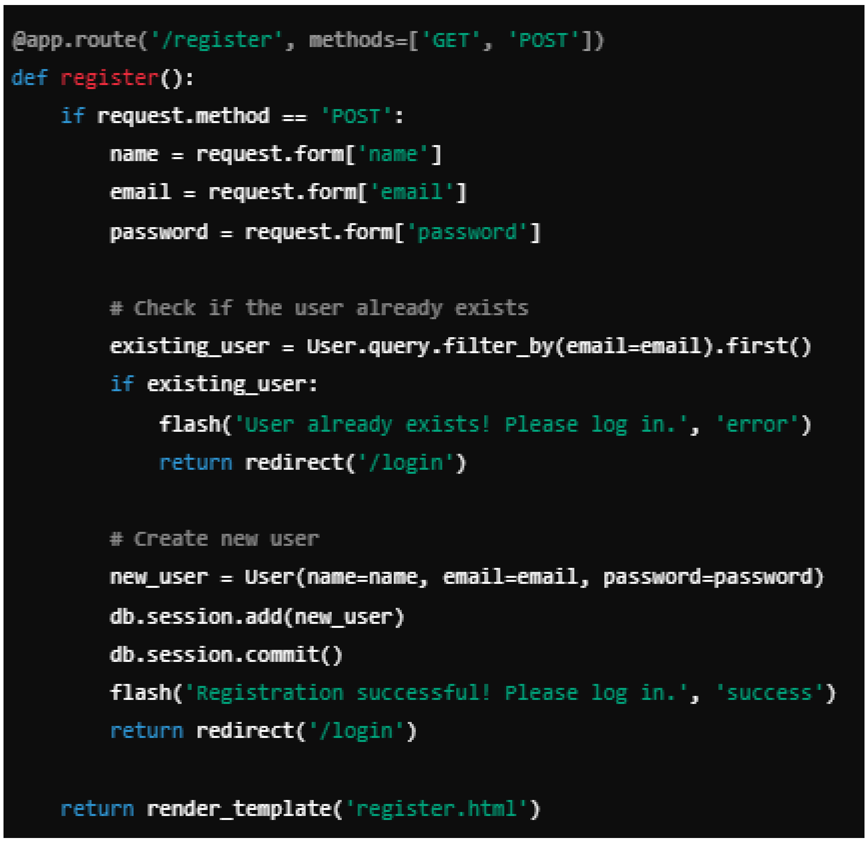


```
@app.route('/')
def index():
    return 'Home'
```

It simply returns the string `'Home'` as a response.

b) Registration Route (`/register`)

The registration route allows users to sign up for the application by submitting their name, email, and password.



```
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        password = request.form['password']

        # Check if the user already exists
        existing_user = User.query.filter_by(email=email).first()
        if existing_user:
            flash('User already exists! Please log in.', 'error')
            return redirect('/login')

        # Create new user
        new_user = User(name=name, email=email, password=password)
        db.session.add(new_user)
        db.session.commit()
        flash('Registration successful! Please log in.', 'success')
        return redirect('/login')

    return render_template('register.html')
```

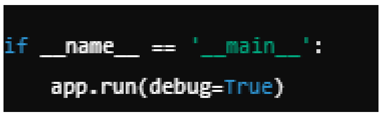
- **GET Request**: When the page is accessed with a GET request, the `register.html` template is rendered to display the registration form.
- **POST Request**: When the form is submitted:
  - The form data (name, email, and password) is retrieved from the request.
  - A check is performed to see if a user already exists with the provided email by querying the `User` table.
  - If the email already exists, an error message is flashed and the user is redirected to the login page (`/login`).
  - If the email is available, a new `User` object is created with the provided data, added to the session, and committed to the database.
  - A success message is flashed, and the user is redirected to the login page.

c) Login Route (`/login`)

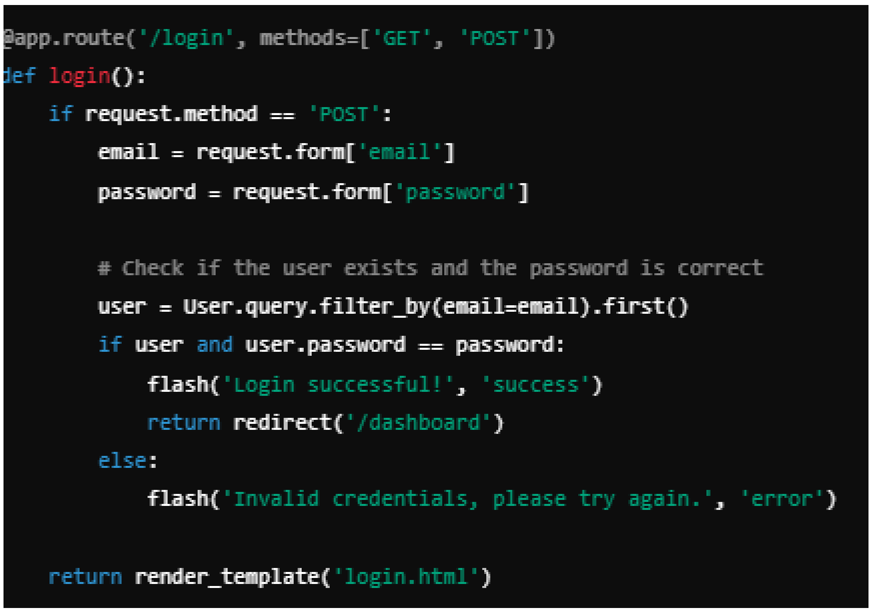
The login route allows users to log into the system using their email and password.

5. Running the Application

To run the application in development mode, the following block of code starts the Flask development server:



```
if __name__ == '__main__':
    app.run(debug=True)
```

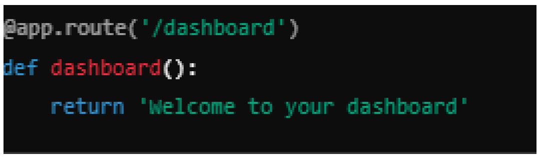


```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']

        # Check if the user exists and the password is correct
        user = User.query.filter_by(email=email).first()
        if user and user.password == password:
            flash('Login successful!', 'success')
            return redirect('/dashboard')
        else:
            flash('Invalid credentials, please try again.', 'error')

    return render_template('login.html')
```

- **GET Request**: When the login page is accessed with a GET request, the `login.html` template is rendered to display the login form.
- **POST Request**: When the form is submitted:
  - The form data (email and password) is retrieved from the request.
  - The system checks if a user exists with the provided email by querying the `User` table.
  - If the user exists and the provided password matches the stored password, the login is considered successful, and a success message is flashed. The user is then redirected to the dashboard page (`/dashboard`).
  - If the credentials are invalid (either the email doesn't exist or the password doesn't match), an error message is flashed.

d) Dashboard Route (`/dashboard`)

The dashboard route is where users are directed after a successful login. In this current implementation, it simply returns a welcome message.



```python
@app.route('/dashboard')
def dashboard():
    return 'Welcome to your dashboard'
```

This route returns the string `'Welcome to your dashboard'` when accessed.

4. Flash Messages

Flash messages are used throughout the application to provide feedback to users based on their actions. These messages are displayed on the user interface to indicate whether actions like registration or login were successful or if there was an error.

Example of a success flash message:

```
flash('Registration successful! Please log in.', 'success')
```

```
flash('Invalid credentials, please try again.', 'error')
```

Flash messages are rendered in the corresponding templates (`login.html` and `register.html`) using the `get_flashed_messages()` method provided by Flask.

5. Running the Application

To run the application in development mode, the following block of code starts the Flask development server:

```
if __name__ == '__main__':
    app.run(debug=True)
```

debug=True: This option allows Flask to automatically reload the server when code changes and provides useful debugging information in the browser.

Summary

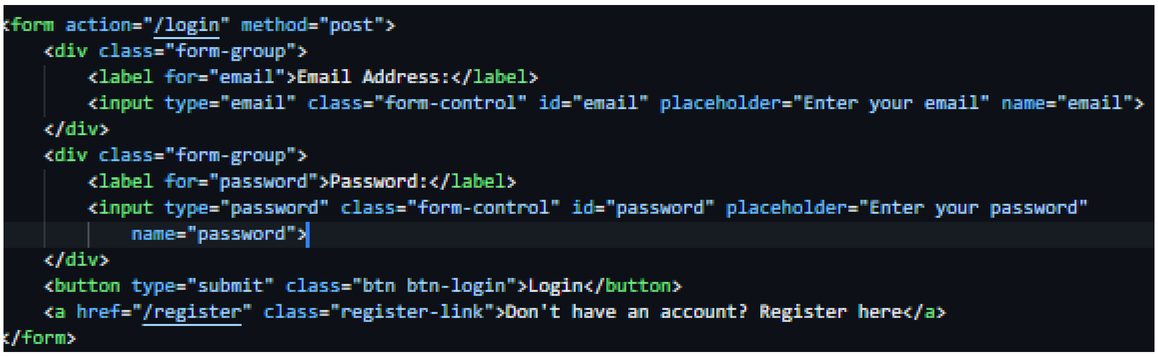
This authentication system consists of basic user registration, login, and a dashboard display. The steps involved in the operation are as follows:

1. **Registration**: Users submit their name, email, and password. If the email is not already in the database, a new user is created and stored in the database.
2. **Login**: Users submit their email and password. The system verifies the credentials and, if valid, directs the user to the dashboard.
3. **Dashboard**: After successful login, users are redirected to a dashboard page.

The application uses Flask, SQLAlchemy for database handling, and SQLite as the database system. Flash messages provide immediate feedback to the user about the success or failure of their actions.

HTML Form Documentation for Flask Login Page

This section provides detailed documentation for the HTML code used in the login form of the Flask application. The form allows users to log in by submitting their email and password. It also provides a link for users who do not yet have an account, redirecting them to the registration page.



```html
<form action="/login" method="post">
    <div class="form-group">
        <label for="email">Email Address:</label>
        <input type="email" class="form-control" id="email" placeholder="Enter your email" name="email">
    </div>
    <div class="form-group">
        <label for="password">Password:</label>
        <input type="password" class="form-control" id="password" placeholder="Enter your password"
            name="password">
    </div>
    <button type="submit" class="btn btn-login">Login</button>
    <a href="/register" class="register-link">Don't have an account? Register here</a>
</form>
```

Explanation of the Code:

1. Form Tag

html

```html
<form action="/login" method="post">
```

- `action="/login"`: The form sends a POST request to the `/login` route when submitted. This URL corresponds to the login route in the Flask application.
- `method="post"`: The form uses the POST method, meaning that the form data will be sent in the body of the HTTP request when the form is submitted.

2. Email Field

html

```html
<div class="form-group">

    <label for="email">Email Address:</label>

    <input type="email" class="form-control" id="email"
placeholder="Enter your email" name="email">

</div>
```

- `<label for="email">Email Address:</label>`: This label describes the input field, making it accessible for users and screen readers. It is associated with the input field through the `for="email"` attribute.
- `<input type="email" class="form-control" id="email" placeholder="Enter your email" name="email">`:

    - `type="email"`: This input field is designed for email addresses. It will automatically validate the input to ensure it follows the correct email format (e.g., `user@example.com`).
    - `class="form-control"`: This class applies Bootstrap's styling to the input field to ensure it has the correct appearance and behavior.
    - `id="email"`: The `id` is used to uniquely identify this field, and it's linked to the label with the `for="email"` attribute.
    - `placeholder="Enter your email"`: The placeholder text gives users a hint about what information is expected in the field.
    - `name="email"`: This name attribute is used to send the email value to the server in the form data when the form is submitted.

3. Password Field

html

```
<div class="form-group">

    <label for="password">Password:</label>

    <input type="password" class="form-control" id="password"
placeholder="Enter your password" name="password">

</div>
```

- `<label for="password">Password:</label>`: This label describes the password input field, linked via the `for="password"` attribute.
- `<input type="password" class="form-control" id="password" placeholder="Enter your password" name="password">`:
    - `type="password"`: This input field is designed for passwords. It hides the input characters to ensure privacy.
    - `class="form-control"`: This applies Bootstrap's styling to the input field, ensuring it is appropriately formatted.
    - `id="password"`: The `id` attribute is used to uniquely identify the password field and links it to the label.
    - `placeholder="Enter your password"`: This placeholder text prompts the user to enter their password.
    - `name="password"`: This name attribute is used to send the password value to the server in the form data.

4. Submit Button

html

```
<button type="submit" class="btn btn-login">Login</button>
```

- `<button type="submit">`: This button triggers the form submission when clicked. The `type="submit"` attribute specifies that the button submits the form.
- `class="btn btn-login"`: The `btn` class applies Bootstrap's button styling, and the additional `btn-login` class is used for custom styling specific to this form's button (this class could be used in custom CSS for the button's appearance).

5. Registration Link

html

```
<a href="/register" class="register-link">Don't have an account?
Register here</a>
```

- `<a href="/register" class="register-link">`: This is an anchor tag that links to the registration page. When clicked, it redirects the user to the `/register` route where they can sign up for a new account.
- The text inside the link, `"Don't have an account? Register here"`, encourages users who are not yet registered to navigate to the registration page.

Form Submission Process

1. **User Input**: The user enters their email and password in the form fields.
2. **Form Submission**: When the user clicks the **Login** button, the form data is sent via a POST request to the Flask route `/login`.
3. **Flask Handling**: The Flask route for `/login` will receive the form data (`email` and `password`), check if the credentials are valid, and then redirect the user based on the validation result (either to the dashboard or back to the login page with an error message).

Additional Notes

Form Validation: While not included in the provided HTML code, you can add client-side validation using HTML5 attributes like `required` to ensure that the user fills out the form fields before submitting. Example:
html
```
<input type="email" class="form-control" id="email" name="email"
required>
```

-

- **Styling**: The form uses the `form-control` class for consistent Bootstrap styling. You can further customize the appearance of the form by applying custom CSS classes, such as `btn-login` and `register-link`, to adjust the look and feel of the form elements.