

# ProManage Web-based Project Management Application

Student ID: 5614928 (Project Partner: 5653630), May 2025

*Software Development and Security (WM145-24), WMG, University of Warwick*

---

## Contents

1. Introduction . . . . .	2
2. Overview of the Application Architecture . . . . .	2
2.1. Front End . . . . .	2
2.2. Back End . . . . .	3
3. Explanation of Back-End Implemented Features . . . . .	5
3.1. Logging In . . . . .	5
3.2. Signing Up . . . . .	6
3.3. Team Member Actions . . . . .	7
3.4. Project Manager Actions . . . . .	8
3.5. Activity Logs . . . . .	11
4. Design Justifications and Security Decisions . . . . .	12
5. Development Process and Issues Encountered . . . . .	13

# 1. Introduction

This project is a task management application with user authentication, role-based access (project manager vs. team member), and activity tracking. This report focuses on the back end, specifically the database components. It explains how tasks, users, and activity logs are managed through Flask and SQLAlchemy.

## 2. Overview of the Application Architecture

### 2.1. Front End

The `/templates/` folder is front end core, containing HTML files to structure the web pages. Separate HTML files were created for the project manager and team member dashboards, as well as the signing-in and signing-up pages. Python Flask links to the HTML file using `render_template()`, calling the correct page when the browser accesses a specific route. HTML uses Flask's `url_for()` to execute other Python routes and their route-specific functions. `/static/` holds a CSS file that improves layout and appearance. For instance, it styles `flash()` messages differently for errors and general information.

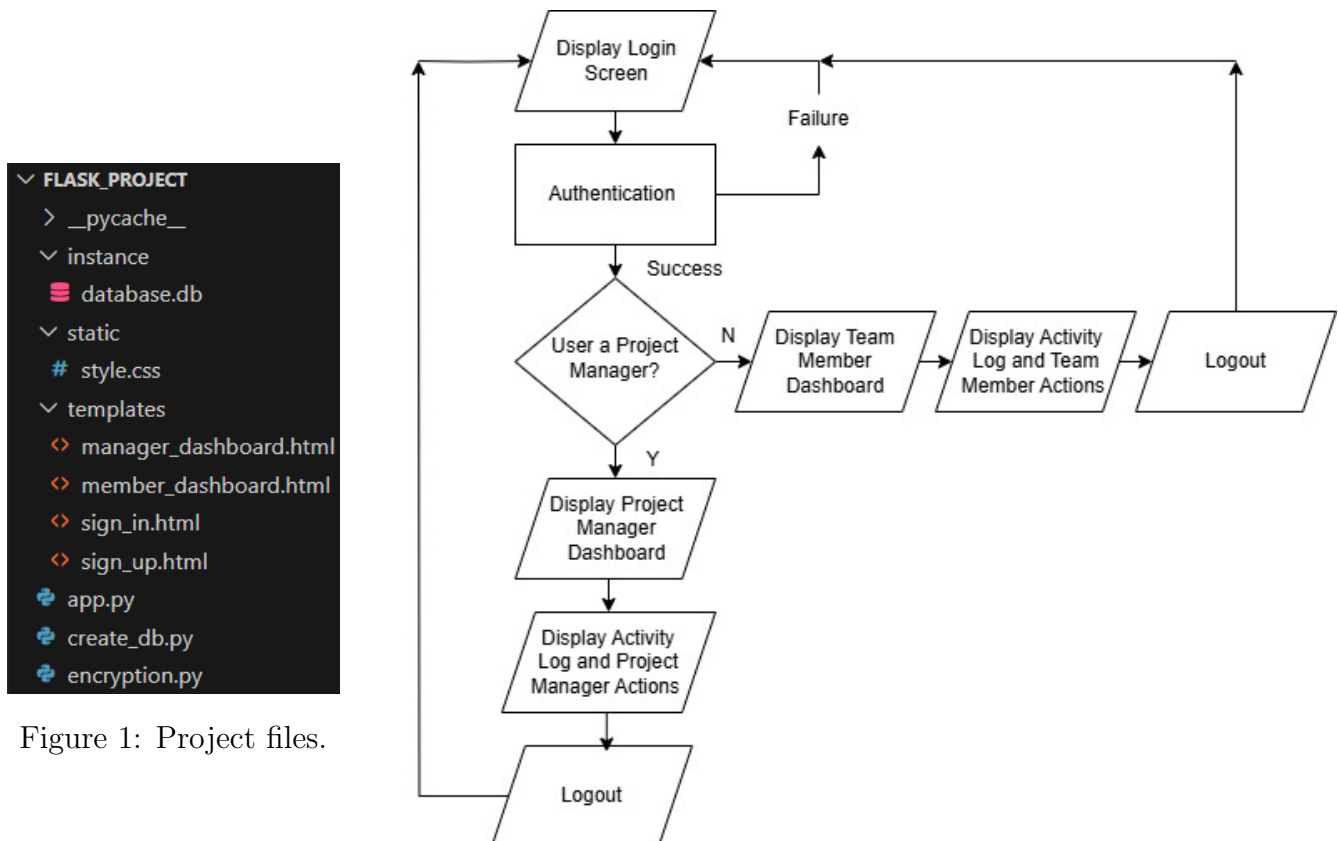


Figure 1: Project files.

Figure 2: Main front end logic.

## 2.2. Back End

The back end is composed of three Python files implemented in Flask, a lightweight web framework for routing between pages and rendering HTML templates.

The main file, `app.py`, is responsible for:

- Creating ‘app’, the Flask object.
- Defining User, Task, and Activity models (database tables).
- Setting up routes and linking them to HTML templates.

`create_db.py` generates `database.db` (the database file) if it does not exist and must be run before `app.py` when running the application for the first time. It creates two default users: a project manager and team member. `encryption.py` generates a unique encryption key using the cryptography library. `app.py` imports this key, storing it in the user’s database row.

The database is created with SQLAlchemy and stored in an SQLite file. SQLAlchemy uses Python classes (models) to represent database tables with attributes such as columns (fields) and foreign keys which define relationships. For example, the User class represents user objects with fields: ID, username, password hash, role, and encryption key. SQLAlchemy uses ORM methods like `.add()`, `.get()`, and `.commit()` to interact with the SQLite database through Python code.

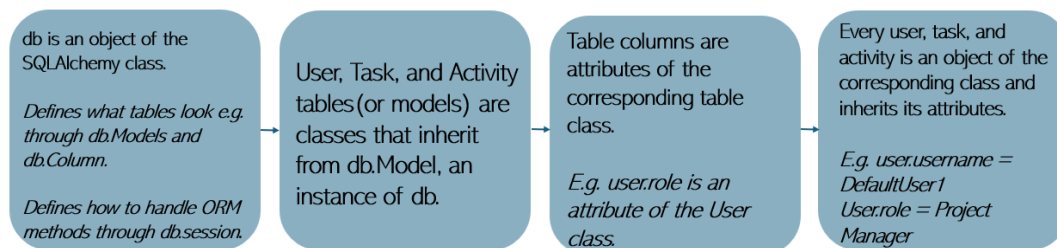


Figure 3: SQLAlchemy’s database logic summarised.

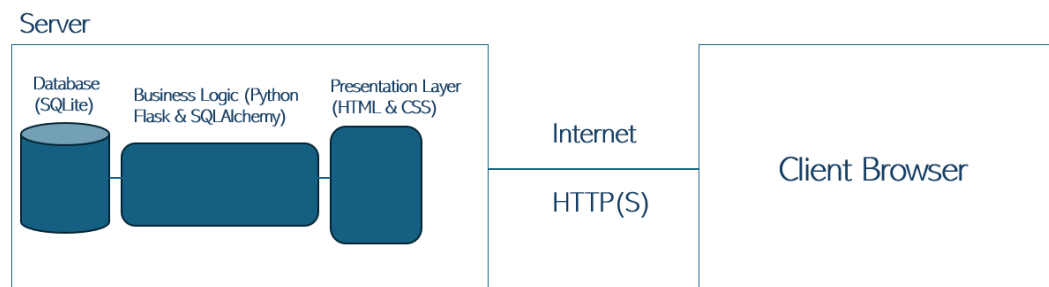


Figure 4: Application architecture diagram.

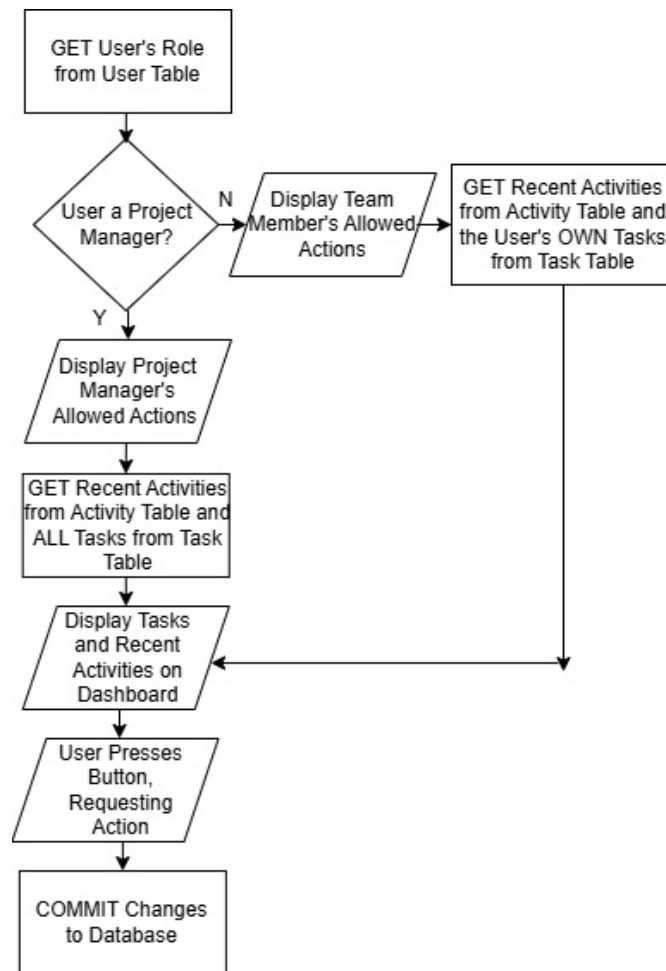


Figure 5: Main back end logic.

### 3. Explanation of Back-End Implemented Features

Back end features involve retrieving and updating table rows. To update rows, the user presses a button, calling a route in app.py that updates the relevant data. For instance, a project manager editing a team member updates the member's row in the User table. Web pages display database data by receiving them as variables from Flask. Flask queries the table and sets the result equal to a variable. It passes the variable to the corresponding HTML template as an argument in `render_template()`. The client's browser uses GET requests to retrieve data from the Flask server, and POST requests to send.

ID	Username	Password Hash	Role	Encryption Key
1	DefaultUser1	script:32768:8:1\$m0M7b36 KjdoBtuD\$3bf02d5f079104 ...	Project Manager	mLQFnT9iMZZ 54wylDWVL8S 6ckzNkRGuzrh 3xtlmN0c=
2	DefaultUser2	script:32768:8:1\$OZxussCbb 6SG3KdN\$75d99e469b9...	Team Member	3yzZD2- FgShszuFToMlg wkofCb4XGrjAG S3tSEBZBog=

Figure 6: Example of the User table.

#### 3.1. Logging In

When logging in, the user enters their username, password, and role for authentication. The `/login` route queries the User table using `.filter_by(username=form.username.data)` and `.first()` to find the entered username. If found, the inputted password is hashed, compared to the stored password hash, and the role is also compared. If they match, the user is redirected to their dashboard. If not, access is denied.

## ProManage

Sign In

**Invalid username or password.**

**Username**

**Password**

**Role**

Project Manager ▾

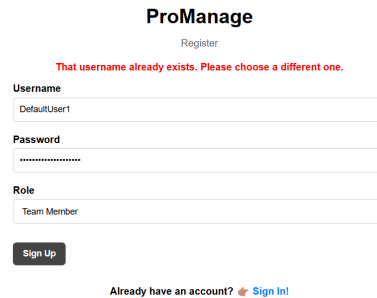
Sign In

Don't have an account? 🖱️ [Sign Up!](#)

Figure 7: Invalid Password Entered.

## 3.2. Signing Up

When creating an account, app.py queries the User table to check whether the username exists. If not, the password is hashed, an encryption key is generated, and, along with the username and role, they are all stored in a new row in the table using `.add()`.



**ProManage**  
Register

That username already exists. Please choose a different one.

**Username**  
DefaultUser1

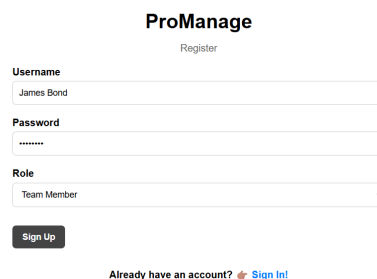
**Password**  
.....

**Role**  
Team Member

Sign Up

Already have an account? [Sign In!](#)

Figure 8: Signing up with a username currently in use.



**ProManage**  
Register

New user 'James Bond' created.

**Username**  
James Bond

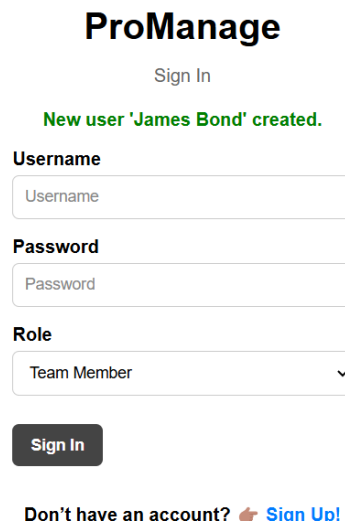
**Password**  
.....

**Role**  
Team Member

Sign Up

Already have an account? [Sign In!](#)

Figure 9: Signing up as a new user.



**ProManage**  
Sign In

New user 'James Bond' created.

**Username**  
Username

**Password**  
Password

**Role**  
Team Member

Sign In

Don't have an account? [Sign Up!](#)

Figure 10: The username is not taken so the user was successfully created.

### 3.3. Team Member Actions

The `/dashboard` route ensures team members can only see their assigned tasks by querying the `Tasks` table with their user ID using `.filter_by(assigned_to=current_user.id)` and `.all()`. Team members can update task status by pressing 'Update Status'. This sends the task ID and new status to the `/update_task` route, updating the `Task` table.

#### ProManage

---

Welcome, DefaultUser2 (Team Member)

**My Tasks**

Task	Status
Create login page	Not Started
Update database	Not Started

Update Status

**Progress Summary**

Not Started: 2  
Completed: 0  
In Progress: 0  
Awaiting Review: 0

**Recent Activity**

- Task 'Create login page' assigned to DefaultUser2 by DefaultUser1.
- Task 'Update database' assigned to DefaultUser2 by DefaultUser1.

Figure 11: DefaultUser2's dashboard.

#### ProManage

---

Welcome, DefaultUser2 (Team Member)

Task status updated.

**My Tasks**

Task	Status
Create login page	Completed
Update database	In Progress

Update Status

**Progress Summary**

Not Started: 0  
Completed: 1  
In Progress: 1  
Awaiting Review: 0

**Recent Activity**

- Task 'Create login page' assigned to DefaultUser2 by DefaultUser1.
- Task 'Update database' assigned to DefaultUser2 by DefaultUser1.
- DefaultUser2 updated task 'Create login page' to 'Completed'.
- DefaultUser2 updated task 'Update database' to 'In Progress'.

Figure 12: DefaultUser2 updated their tasks.

### 3.4. Project Manager Actions

Project managers can assign tasks by submitting a form to the `/assign_task` route, including a task description, assignee's name, and status. The User table is queried to confirm the assignee exists. If so, the new task is added to the Task table by `.add()`.

**Assign Task**  
**Task**  
  
**Assign To**  
  
**Status**

**Assigned Tasks**

Task	Person	Status
Create login page	DefaultUser2	Completed
Update database	DefaultUser2	In Progress

**Activity Log**

- Task 'Create login page' assigned to DefaultUser2 by DefaultUser1.
- Task 'Update database ' assigned to DefaultUser2 by DefaultUser1.
- DefaultUser2 updated task 'Create login page' to 'Completed'.
- DefaultUser2 updated task 'Update database ' to 'In Progress'.

Figure 13: Submitting the task form by pressing 'Assign Task'.

**Activity Log**

- Task 'Create login page' assigned to DefaultUser2 by DefaultUser1.
- Task 'Update database ' assigned to DefaultUser2 by DefaultUser1.
- DefaultUser2 updated task 'Create login page' to 'Completed'.
- DefaultUser2 updated task 'Update database ' to 'In Progress'.
- Task 'Design a flowchart' assigned to James Bond by DefaultUser1.

Figure 14: Task successfully assigned to James Bond.



Managers can add, edit, and delete team members. To add, a form is submitted and a new User object is created with `.add()`. To edit, the form sends the user ID, and `app.py` uses `.get()` to find and update the user's record in the User table. To delete, the user ID is sent, their tasks are removed, and the user is deleted with `.delete()`.

### Activity Log

- Task 'Create login page' assigned to DefaultUser2 by DefaultUser1.
- Task 'Update database ' assigned to DefaultUser2 by DefaultUser1.
- DefaultUser2 updated task 'Create login page' to 'Completed'.
- DefaultUser2 updated task 'Update database ' to 'In Progress'.
- Task 'Design a flowchart' assigned to James Bond by DefaultUser1.

### Add Member

**Name**

**Role**

**Add**

Figure 15: Adding a new user.

Spiderman was added.

### Assigned Tasks

Task	Person	Status
Create login page	DefaultUser2	Completed
Update database	DefaultUser2	In Progress
Design a flowchart	James Bond	Not Started

### Activity Log

- Task 'Create login page' assigned to DefaultUser2 by DefaultUser1.
- Task 'Update database ' assigned to DefaultUser2 by DefaultUser1.
- DefaultUser2 updated task 'Create login page' to 'Completed'.
- DefaultUser2 updated task 'Update database ' to 'In Progress'.
- Task 'Design a flowchart' assigned to James Bond by DefaultUser1.
- Spiderman was created by DefaultUser1.

Figure 16: New user was successfully added.

Edit Members

Name	Role	Actions	
DefaultUser2	Team Member	Update	Delete
James Bond	Team Member	Update	Delete
Spiderman	Project Manager	Update	Delete

Figure 17: Changing Spiderman to a project leader by pressing ‘Update’.

Spiderman was updated.

Assigned Tasks

Task	Person	Status
Create login page	DefaultUser2	Completed
Update database	DefaultUser2	In Progress
Design a flowchart	James Bond	Not Started

Activity Log

- Task 'Create login page' assigned to DefaultUser2 by DefaultUser1.
- Task 'Update database ' assigned to DefaultUser2 by DefaultUser1.
- DefaultUser2 updated task 'Create login page' to 'Completed'.
- DefaultUser2 updated task 'Update database ' to 'In Progress'.
- Task 'Design a flowchart' assigned to James Bond by DefaultUser1.
- Spiderman was created by DefaultUser1.
- admin was deleted by DefaultUser1.
- 'Spiderman' was updated by DefaultUser1.

Figure 18: Spiderman was successfully edited.

James Bond was deleted.

### Assigned Tasks

Task	Person	Status
Create login page	DefaultUser2	Completed
Update database	DefaultUser2	In Progress

### Activity Log

- Task 'Create login page' assigned to DefaultUser2 by DefaultUser1.
- Task 'Update database ' assigned to DefaultUser2 by DefaultUser1.
- DefaultUser2 updated task 'Create login page' to 'Completed'.
- DefaultUser2 updated task 'Update database ' to 'In Progress'.
- Task 'Design a flowchart' assigned to James Bond by DefaultUser1.
- Spiderman was created by DefaultUser1.
- admin was deleted by DefaultUser1.
- 'Spiderman' was updated by DefaultUser1.
- James Bond was deleted by DefaultUser1.

Figure 19: Deleting James Bond.

### Edit Members

Name	Role	Actions	
<input type="text" value="DefaultUser2"/>	<input type="text" value="Team Member"/>	<input type="button" value="Update"/>	<input type="button" value="Delete"/>
<input type="text" value="Spiderman"/>	<input type="text" value="Project Manager"/>	<input type="button" value="Update"/>	<input type="button" value="Delete"/>

Figure 20: James Bond no longer appears as a user.

### 3.5. Activity Logs

All users can see recent activity in their Activity Log. The /dashboard route passes the HTML page a query of the full Activity table through .all(), displaying the latest activities on the page.

## 4. Design Justifications and Security Decisions

The back and front ends are both coded in `app.py` for simplicity and efficiency. HTML methods (e.g. GET, POST) and database interactions are handled in the same file as the HTML template rendering. This allows changes to be made faster by avoiding switching between files and reducing the need to configure imports from other files.

The Task table uses the foreign key ‘`user_id`’, linked to the User table’s ID field. This simplifies assigning and tracking tasks. For example, when a team member updates a task’s status, their user ID is searched up in the Task table to find the corresponding task entry.

Sessions hold the current user’s information, avoiding the need to login to every page the user visits and manually query the User table. Flask’s LoginManager handles the session. After logging in, the `/login` route calls `login_user(user)`, setting Flask’s `current_user` to the current user’s row in the User table. `current_user` holds the user’s attributes e.g. `current_user.id`, `current_user.role`, etc. This data is stored on the client’s browser as a session cookie. When the user navigates through pages, their data is saved in the session. Since `current_user` behaves like a global variable, it can be accessed in any function or HTML template to identify and retrieve the current user’s details.

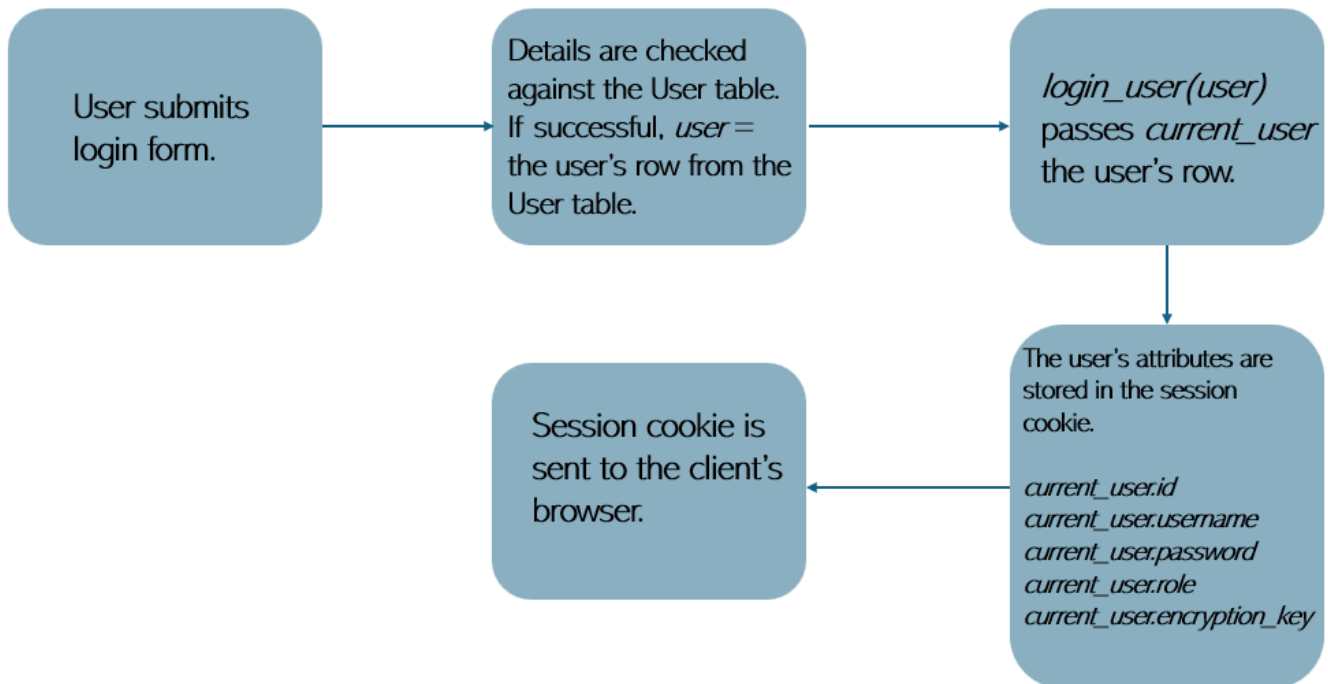


Figure 21: Flask sessions summarised.

In real life, HTTPS encrypts all data between browser and server, protecting user data like credentials from interception. Every user has a unique encryption key stored in their User table row. When a team member submits a file, it would be securely sent via HTTPS and encrypted on the server using `current_user.encryption_key` before being stored in the File table.

Role-based access control prevents team members from actions that only project managers are allowed. Routes which affect other users like `/edit_member` ensure `current_user.role = 'Project manager'` before accessing the database. `/dashboard` ensures team members can only see their tasks while managers can see all tasks.

Passwords are stored as hashes with `generate_password_hash()`. When logging in, the password is compared to the hash in the User table with `check_password_hash()`. These functions use salting to prevent rainbow table attacks.

Flask's `@login_required` decorator ensures only authenticated users can access certain routes, preventing unauthorised access to pages and actions.

## 5. Development Process and Issues Encountered

Back-end Creation Steps:

- Database and tables had to be created first since all routes access them.
- Authentication with `/register` and `/login`.
- `/dashboard` (core route for all users).
- Routes for specific actions: `/assign_task`, `/update_task`, `/edit_member`, etc.

Deleting a user as project manager caused errors because of the foreign key linking the User and Task tables. Every task is assigned to a user via `user.id`. When trying to delete a user with assigned tasks, the database threw an error as it violated the relationship. This was resolved by deleting all the user's tasks before deleting the user.

Activity Log session issues were solved by replacing the concept with an Activity database table. User activities were only stored in the user session, but because sessions are user-specific, different users could not see each other's actions. An Activity table allows all users to see activities because they are stored as strings inside Activity's message field and are retrieved through ORM.