Project Spec

Description:

This project aims to build a personal portfolio website using the Flask framework. The website will showcase the admin's professional projects, personal information, and an image gallery. It will feature two primary user roles:

1. **Admin:** The admin will have the ability to add, edit, or delete content for projects and resume directly through the website interface.
2. **Viewers:** Regular visitors can view the portfolio content and be able give rating or some kind of feedback on the projects.

Key Features:

- **Admin Control Panel:** Admin can log in to manage the content seamlessly.
- **Content Management:** Admin can add, edit, or remove projects, descriptions, and images.
- **Viewer Interface:** Users can explore the portfolio content, limited to viewing and reading.
- **Flask for Backend:** Flask serves as the backend framework, handling routing, rendering templates, and managing content.
- **Database:** SQLite (or a simple relational database) will store user credentials, projects, and images efficiently.
- **Front-End:** Utilizes HTML, CSS, and JavaScript for interactive elements; Bootstrap is considered for responsive design.

External Mechanisms:

- **Flask:** Manages routing and server-side logic effectively.

GUI:

- **Admin Panel:** A user-friendly web-based GUI for managing content.
- **Viewer Interface:** A public-facing website where visitors can explore the portfolio.

Task Vignettes:

1. **Task 1: Admin Login**
   Vignette: The admin navigates to the login page, enters their username and password, and logs into the home page. From the home page, they can see a list of all existing projects and images, with options to add, edit, or delete content.
   Technical Details:
   - User credentials stored in the database.
   - Passwords can be encrypted for security.
2. **Task 3: Viewing Projects (Viewer Flow)**
   Vignette: Admin and viewer visits the portfolio website and can see the Projects which are added through the UI. Clicking on a the image of the project redirects to the link if given by the admin.
   Technical Details:
   - Each project has a unique URL endpoint (e.g., /project/<id>).
   - Pagination is implemented for larger portfolios.
3. **Task 4: Admin Adding Projects via UI**
   Vignette: The admin uses the control panel to add multiple projects at once through a bulk upload feature. A form allows the admin to upload multiple images and enter descriptions for each project.
   Technical Details:
   - Multiple files can be stored in a single entry (image, pdf, links) when admin submits details.
   - Each project is stored in the database with associated images.
4. **Task 5: Deleting Projects**
   Vignette: The admin navigates to the delete section of the dashboard, enters the project ID of the project they wish to delete, and confirms the deletion. The project is removed from both the database and the projects local folder.
   Technical Details:
   - Deletion confirmation dialog to prevent accidental removals.
   - Backend logic to remove project files from the file system.
5. **Task 6: Logout Feature**
   Vignette: After managing content, the admin should be able to click the logout button on the homepage, which securely ends the session and redirects them to the homepage as a viewer.
   Technical Details:

- Session management using logout.
- Redirects to the homepage upon logout.

6. **Task 7: UI/UX Improvements**
   Vignette: The admin updates the project adding and deleting sections with CSS styling, enhancing the overall look and feel of the admin panel to make it more user-friendly.
   Technical Details:
   - CSS updates for better layout and responsive design.
   - Use of Bootstrap for responsive UI elements.
7. **Task 8: Admin Resume Update Form**
   Vignette: The admin can update their resume directly from the UI through a dedicated form. The admin uploads a new resume file, which is then stored securely and replaces the previous version.
   Technical Details:
   - File upload handling for resumes.
   - Database entry updated with the new resume file path.

Technical Flow:

- **Data Flow:** Inputs are gathered at various stages, such as the admin inputting details about projects and images. Flask will handle CRUD operations with the database, while the viewer interface will retrieve and display data from the database on the front end.
- **Blocks:**
  - **Content Management:** This block manages functions for adding, updating, or deleting content in the database.
  - **Frontend:** Contains HTML templates for the user interface.
- **Data Types:**
  - Each project can be stored as text field(name and description), JPEG images(project_image) and files(pdf).
  - All projects together can be structured as a list of lists (List of JPEGs, strings).

User Interaction:

- **Admin:** Access to an admin panel with input forms for adding and deleting content.
- **Viewer:** Simple navigation options to view projects and images.

Self-Assessment:

The original sketch primarily described the portfolio itself, but the final specification now includes an admin interface and elaborates on the content management system. I'm moderately confident in building the GUI, although I consider myself an intermediate coder.
I may need assistance with managing and setting up the database and would appreciate suggestions on the best database choice to use. I am familiar with SQL and have researched SQLite and MongoDB; guidance on which database to use would be helpful. Additionally, I might require help with the admin login section.

Link to the Git Repository:

https://github.com/deekshareddy57/PortfolioWebsite_HCI584/tree/main