Technical Documentation for Enterprise Pro Task Management System V2

File: app.py (Main Application)

The core Flask application handling routing, user sessions, and interactions with the database.

Class: myClass

- **Purpose**: Manages routes, user sessions, and business logic.
- **Key Attributes**:
    - blueprint: Flask Blueprint for modular routing.
    - user_logged_in: Tracks login state.
    - database: Instance of databaseManager for DB operations.
    - list_operation_manager: Instance of listOperationsManager for sorting/filtering.

Key Methods:

1. **index()**
    - **Route**: /
    - **Functionality**:
        - Redirects unauthenticated users to /login.
        - Admins/supervisors are redirected to their respective dashboards.
        - Fetches projects and tasks assigned to the logged-in user.
        - Renders index.html with filtered projects.
2. **login()**
    - **Route**: /login (GET/POST)
    - **Functionality**:
        - Validates LoginForm inputs.
        - Checks credentials using check_password_hash (or plaintext fallback).
        - Sets session variables (user_id, user_role, user_logged_in).
        - Redirects to appropriate dashboards or displays error messages.

3. **create_task()**
    - **Route**: /create_task (POST)
    - **Functionality**:
        - Extracts task details from form data (task-title, task-due-date).
        - Converts dates to DD-MM-YYYY format.
        - Inserts task into the tasks table and assigns it to the current user via assigned_tasks.
4. **sort_tasks(sort_type)**
    - **Route**: /sort_tasks
    - **Parameters**: sort_type (e.g., "due date", "status").
    - **Functionality**:
        - Fetches tasks and uses listOperationsManager.categorise_data() to sort.
        - Renders sorted tasks in tasks.html.
5. **add_user_to_project()**
    - **Route**: /add_user_to_project (GET/POST)
    - **Functionality**:
        - Uses UsersInProjectsForm to link users to projects.
        - Iterates over selected usernames and inserts entries into project_users table.
6. **delete_task(task_id)**
    - **Route**: /delete_task
    - **Parameters**: task_id (ID of the task to delete).
    - **Functionality**:
        - Moves the task to deleted_tasks list (soft delete).
        - Removes the task from the tasks table.

File: forms.py

Handles form creation and validation using Flask-WTF.

Key Classes:

1. **LoginForm**
    - **Fields**: username, password (with DataRequired validation).
    - **Purpose**: Authenticates users.
2. **CreateUserForm**
    - **Fields**: username, password, role (dropdown), team (dropdown).
    - **Purpose**: Registers new users with hashed passwords.
3. **CreateProjectForm**
    - **Fields**: project_title, project_details, project_status, project_review, project_owner.
    - **Validation**: validate_project_owner ensures the owner exists in the database.
4. **EditTaskForm**
    - **Fields**: task_id, task_title, task_details, task_status, task_assigned_date, task_due_date.
    - **Purpose**: Updates task details in the database.

File: search_sort.py

Provides utilities for sorting, filtering, and searching tasks/projects.

Class: listOperationsManager

1. **binary_search(arr, target)**
    - **Functionality**:
        - Performs binary search on a sorted list.
        - Returns the index of target or -1 if not found.
2. **merge_sort(arr)**
    - **Functionality**:
        - Iterative implementation of merge sort.
        - Sorts the list in ascending order.
3. **categorise_data(arr, categories_type)**
    - **Issue**: Currently returns strings (e.g., "title") instead of sorted data. Needs implementation.
4. **filter_data(arr, filter_type)**

- **Issue**: Similar to categorise_data; returns strings instead of filtered data.

File: use_database.py

Manages SQLite database interactions.

Class: databaseManager

- **Key Methods**:
  1. **create_tables()**
     - Creates tables (users, projects, tasks, etc.) with foreign keys.
  2. **add_user(username, password, role, team)**
     - Inserts a new user with a hashed password.
     - **Risk**: Uses string formatting for SQL queries (prone to injection).
  3. **find_user(username)**
     - Fetches a user by username or ID.
  4. **get_all_from_table(table_name)**
     - Returns all rows from a specified table (e.g., tasks, projects).
  5. **update_user(user_id, ...)**
     - Updates user details (password, role, team) using dynamic query building.

HTML Templates

- **index.html**: Displays assigned projects and tasks. Uses Jinja2 loops to render dynamic content.
- **login.html**: Simple login form with client-side validation.
- **admin.html**: Allows admins to create users/projects and modify permissions.
- **tasks.html**: Shows tasks with filtering/sorting options and a form to create new tasks.
- **supervisor.html**: Dashboard for supervisors to view projects and task statuses.

CSS: styles.css

- **Styling**: Consistent theme with blue/white colors.
- **Responsive Design**: Flexbox layouts for project cards and forms.

- **Dynamic Elements**: Styling for task boxes, dropdowns, and buttons.

Key Technical Notes

1. **Security Risks**:
    - SQL queries in databaseManager use string formatting, which is vulnerable to injection. Use parameterized queries instead.
    - check_password_hash allows a fallback to plaintext passwords (e.g., password == thisUser[2]), which is unsafe.
2. **Incomplete Features**:
    - categorise_data and filter_data in listOperationsManager are placeholders.
    - passwordReset route is unimplemented.
3. **Session Management**:
    - User state (user_id, user_role) is stored in class attributes instead of Flask sessions, which may cause issues in multi-user environments.
4. **Database Schema**:
    - Foreign keys (e.g., project_owner in projects) reference users(username), but usernames may change, breaking referential integrity. Use user_id instead.