Flowmark — To-Do List System

MINI PROJECT REPORT SUBMITTED TO MAHATMA GANDHI UNIVERSITY,
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
BACHELOR OF SCIENCE IN COMPUTER APPLICATIONS

BY,
Aswanidev vs
Register Number:230021078090

DEPT. OF COMPUTER SCIENCE
B.V.M HOLY CROSS COLLEGE CHERPUNKAL,
KOTTAYAM 686 584
August 2025

Department of Computer Science
BVM HOLY CROSS COLLEGE CHERPUNKAL

CERTIFICATE

Certified that the report entitled **Flowmark — To-Do List System** is a bonafide record of the mini project work done by Aswanidev vs(reg no:230021078090) under our Guidance and supervision and is submitted in partial fulfillment of the Bachelor degree in Computer Applications,awarded by Mahatma Gandhi University, Kerala.

HOD
Seena S Nair

Project Guide
Nova Emmanuel

Submitted for Project Evaluation on:

External Examiner

DECLARATION

I hereby declare that the mini project work entitled **Flowmark – To-Do List System** submitted in partial fulfillment of the requirements for the award of the Bachelor of degree in Computer Applications from BVM Holy Cross College, Cherpunkal , is record of bonafide work done under the guidance of Mrs. Nova Emmanuel.

Name:Aswanidev vs

Reg.No:230021078090

Place: Cherpunkal

Date:

ACKNOWLEDGEMENT

Dedicating this mini project to the Almighty God whose abundant grace and mercies enabled successful completion,we would like to express our profound gratitude to all the people who had inspired and motivated us to make this mini project a success. We would like to place our deep sense of gratitude to Mrs.Seena S Nair (Head of the department of Computer Applications) for her guidance in carrying out this project work. We are profoundly grateful to Mrs.Nova Emmanuel as the Project guide for her valuable guidance, suggestions and assessment throughout the project.We also extend our sincere thanks to all other faculty members of the department of Computer Applications for their assistance and encouragement throughout the project. Last, but not least we would like to thank our friends for their co-operation and encouragement.

ABSTRACT

Flowmark is a PHP & MySQL powered To-Do list web application that enables registered users to manage personal tasks efficiently in a structured manner. It supports two interchangeable UI skins (UI1 — compact list style, UI2 — card/dashboard style) with a simple file-based UI switcher, giving users flexibility in choosing the interface that best suits their workflow preferences. Core features include secure user authentication, complete task CRUD (Create, Read, Update, Delete) functionality, filtering and search options, task metadata such as priority and status, due-date management, and a real-time dashboard summarizing task statistics (total, completed, pending, overdue, upcoming).

The system emphasizes modularity by separating presentation (UI1/UI2) and shared backend, promoting reusability and maintainability while allowing further UI expansion. This architecture also facilitates collaborative development where multiple UI designs can coexist without duplicating backend logic. Flowmark has been designed to follow security best practices, including prepared statements, password hashing, and session management, ensuring safe user data handling.

Beyond task tracking, the project demonstrates how open-source technologies (HTML, CSS, JavaScript, PHP, MySQL, and Git) can be integrated to deliver a lightweight yet scalable productivity tool. The report documents the full software development lifecycle — from requirements and analysis to design, implementation, testing, deployment, and maintenance — and highlights considerations for security, usability, and extensibility. Future improvements may include notification services, mobile app integration, and collaborative features for teams.

Table Of Contents

1.Introduction

Task management is a fundamental productivity activity in both academic and professional life. A purpose-built To-Do web application provides persistent storage, easy retrieval, filtering, and a consistent UI across devices — much more effective than scattered notes or ad-hoc lists.

Flowmark is designed to provide a simple, focused task management experience. It is intentionally modular: the **backend** implements business logic (authentication, task CRUD, filtering, dashboard counts) and the **presentation** is implemented as two independent UI skins (UI1 and UI2). This separation allows rapid experimentation with different UX designs without touching backend code.

1.1 Project Overview

Flowmark is a lightweight yet extensible web-based To-Do list management system designed to help users organize and track their daily tasks in a structured, secure, and user-friendly manner. The project was developed as a mini-project using the **LAMP stack (Linux/Apache/MySQL/PHP)** with standard web technologies such as **HTML, CSS, and JavaScript** for the frontend and **MySQL** as the database backend.The system allows users to register, authenticate securely, and manage their personal tasks with details such as title, description, due date, priority, and status. A key distinguishing feature of Flowmark is its **dual user interface design (UI1 and UI2)**. Both UIs provide the same functionality but present tasks differently:

- **UI1** adopts a compact, list-based design for minimalistic task viewing.
- **UI2** uses a modern card-based dashboard layout with more visual elements.

A **UI switcher mechanism** has been implemented via two redirect handler files (`ui1.php` and `ui2.php`) that seamlessly redirect users between UI folders (`ui1/` and `ui2/`) without duplicating backend logic. This approach demonstrates modular design and allows easy experimentation with interface styles.

Flowmark follows software engineering best practices with **modular code organization, database normalization, prepared statements for SQL queries, and password hashing for user security**. The system includes task filtering, searching, and a dashboard summarizing pending, completed, overdue, and upcoming tasks.

The project serves as both a functional productivity tool and an academic case study in **full-stack web application development**, covering all stages of the Software

Development Life Cycle (SDLC): requirement analysis, design, implementation, testing, deployment, and maintenance. It also highlights future scope areas such as notification features, mobile app integration, and collaborative task management.

2.System Specification

2.1 Hardware Specification

- CPU: Intel i3 or equivalent
- RAM: 4 GB (8 GB recommended)
- Storage: 20 GB free

2.2 Software Specification

- OS: Windows / Linux / macOS
- Web Server: Apache (XAMPP/WAMP/LAMP) or Nginx + PHP-FPM
- PHP: 7.4 / 8.x
- Database: MySQL / MariaDB
- Frontend: HTML5, CSS3, JavaScript (ES6)
- Tools Used: Git, VS Code, phpMyAdmin

3.System Study

The **System Study** provides an in-depth examination of the existing problem domain, identifies gaps in current practices, and establishes how the proposed system (Flowmark) addresses those shortcomings. It is divided into three parts: **Existing System**, **Proposed System**, and **Feasibility Study**.

3.1. Existing System

Task management for students and professionals is often handled through:

- Paper-based notes or diaries.
- General-purpose mobile notes applications without structured task metadata.
- Basic spreadsheets or manual lists.

Limitations of Existing System:

- No centralized access across devices.
- Lack of structured metadata such as priority, status, and due date.
- Difficulty in filtering, searching, or tracking overdue tasks.
- No analytics or dashboard summarizing task progress.
- Limited flexibility in customizing the user interface.

3.2. Proposed System (Flowmark)

Flowmark overcomes these limitations by providing a **web-based To-Do list system** with:

- **User Authentication:** Secure login and registration system for per-user task management.

- **Task Management (CRUD):** Users can create, view, update, and delete tasks with details like description, priority, and due date.
- **Filtering and Search:** Tasks can be searched or filtered by keyword, status, or priority.
- **Dashboard:** Summarized counts of total, pending, completed, overdue, and upcoming tasks.
- **UI Flexibility:** Two distinct UIs (UI1 – list-based, UI2 – card-based) with a simple **UI switcher mechanism** (`ui1.php` and `ui2.php` redirect handlers).
- **Security Measures:** Use of password hashing, prepared statements, and secure session management.
- **Extensibility:** Modular design enables adding more UI styles or features without rewriting backend logic.

3.3. Feasibility Study

To validate the practicality of Flowmark, a feasibility analysis was conducted:

- **Technical Feasibility:**
  The project uses open-source, widely available technologies (PHP, MySQL, HTML, CSS, JS) that are supported on almost all platforms. No specialized hardware is required.
- **Operational Feasibility:**
  The system is simple to use, requiring only a web browser. UI1 and UI2 provide flexibility for users with different preferences.
- **Economic Feasibility:**
  Since Flowmark uses free tools (XAMPP, MySQL, VS Code, Git), the cost of development and deployment is negligible, making it highly cost-effective.
- **Schedule Feasibility:**
  The system is lightweight, with limited but essential features, making it possible to complete within a standard mini-project timeline (3–4 weeks).

3.4. **Requirement Specification**

Software Requirement Specification (SRS) is the requirements document that provides the technical specification for the design and development of the software. This document enhances the system's quality of formalizing communication between the system developer and the user and provides the proper information for accurate documentation. A description of each function required to solve the problem is presented in the functional description. The behavioral description section of the specification examines the operation of the software as a consequence of external events and internally generated control characteristics. Validation criteria is perhaps the most important and ironically most often neglected section of the SRS Specification of validation criteria acts as an implicit review of all other requirements. The proposed system has the following requirements.

3.4.1.Functional Requirement

1. User registration and secure authentication.
2. Create tasks (title required), optional description, priority (Low/Medium/High), due date.
3. Read/list tasks, and view task details.
4. Update task fields and mark as complete.
5. Delete tasks with confirmation.
6. Filter and search tasks.
7. Dashboard: total tasks, completed, pending, overdue, upcoming.
8. Toggle UI: from either UI's `todo.php`, a user click calls `ui1.php` or `ui2.php` (parent redirect file) to switch UIs.
9. Responsive UI for mobile and desktop.

3.4.2 Non-Functional Requirements

1. Passwords stored as hashes.

2. Use prepared statements to prevent SQL injection.

3. UX should be responsive and accessible.

4. Maintainable code with clear separation between presentation and backend.

4.System Design

4.1 Introduction

System design is the process of defining the architecture, components, modules, interface and data for a system to satisfy specified requirements. It is a solution to an approach compared to system analysis which translates these “what is” orientation. System requirements into a way of making them operational. The design phase focuses on detailed implementation of the system recommended in the feasibility study. Planning of a system or to replace or complement an existing system. But before this, planning should be done. It must thoroughly understand the old system and determine how computers can make its operations more effective. The importance of system design is the quality. Design is the place where quality is fostered in software development. Design representation of software provides us with that which can be accessed for quality. System design is a transaction from a user-oriented document to a programmer or database personal.it is creative in both art and technology.

4.2. System Flowchart

The classical system flowchart approach to describing and documenting a system will be presented. These system flowcharts are also used in the structured approach that is, from the general to detailed, of the system development life cycle. Because they have been used to describe systems for many years, they are still common in many businesses.

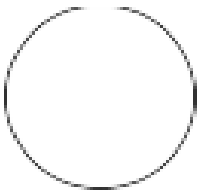
Basic Flow chart Symbols:

Process

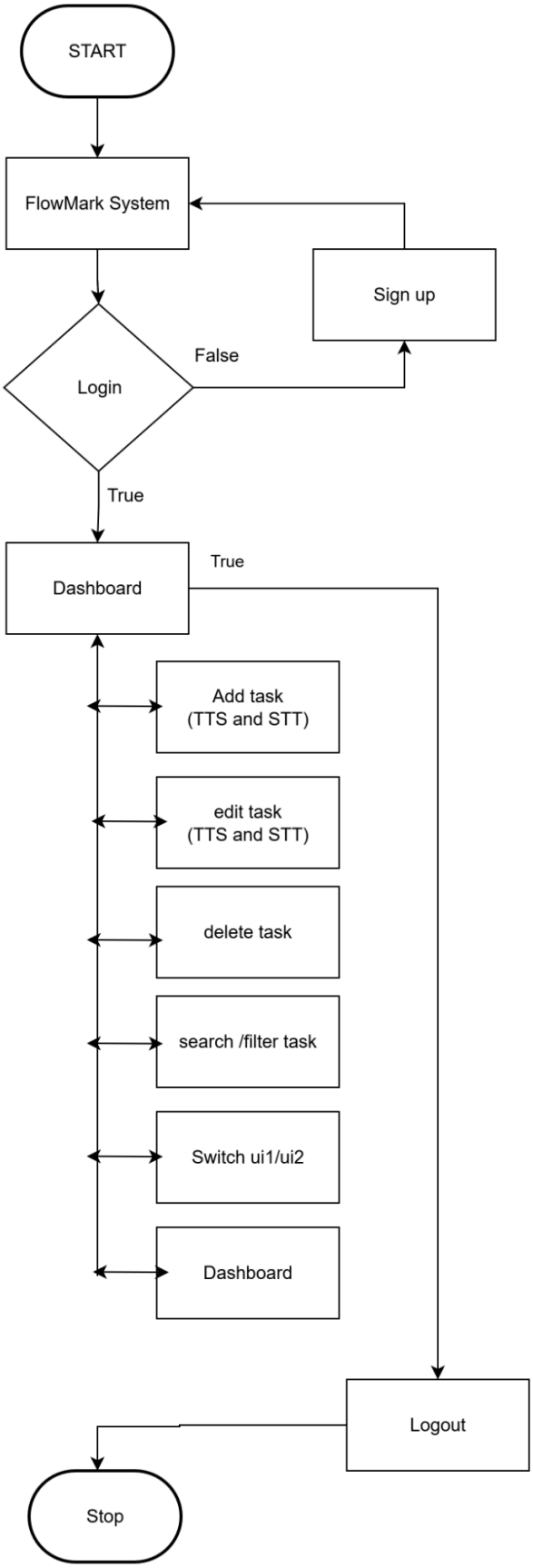
Off page Connector

Data Flow

Input - Output



Connector


START
FlowMark System
Sign up
Login
False
True
Dashboard
True
Add task
(TTS and STT)
edit task
(TTS and STT)
delete task
search /filter task
Switch ui1/ui2
Dashboard
Logout
Stop

4.3.Database Design

The most important aspect of building an application is the design of tables or the database schema. The data stored in the tables must be organized in some manner,which is meaningful. The overall objective in the process of table design has been to treat data as an organizational, resource and as an integrated whole. The organization of data in a database aim to achieve three major objectives, which are given below:

- Data integration
- Data abstraction
- Data independence

Several degrees of normalization have to be applied during the process of table design. The major aim of the process of normalization is to reduce data redundancy and prevent losing data integrity. Data integrity has to be converted at all levels. Pure normalization can access problem related to storage and retrieval of data. During the process of normalization, dependence's can be identified which cause serious problems during deletion and updating. Normalizing also hope in simplifying the structure of the table. The theme behind a database is to handle information as an integrated whole thus making access to information easy, quick, inexpensive and flexible for users.The entire package depends on how the data are maintained in the system.Each table has been designed with a perfect vision. Minor tables have been treated which through takes much space facilitates the process of querying fast and accurate.

PRIMARY KEY

The key is to identify records. Also uniquely notify the not null constraints.

FOREIGN KEY

The key which references the primary key, is the data inserted in the primary key column of the table.

NORMALIZATION

Normalization is the process of efficiently organizing data in a database. There are two goals of the normalization process: eliminating redundant data (for example,storing the same data in more than one table) and ensuring data dependencies make sense (only storing related data in a table). Both of these are worthy goals as they reduce the amount of space a database consumes and ensure that data is logically stored.

First normal form (1NF)

sets the very basic rules for an organized database: Eliminate duplicative columns from the same table. Create separate tables for each group of related data and identify each row with a unique column or set of columns (the primary key).

Second normal form (2NF)

further addresses the concept of removing duplicative data: Meet all the requirements of the first normal form. Remove subsets of data that apply to multiple rows of a table and place them in the separate tables.

Tables:

Login

Field name	type	constraint	description
uid	int	pk	Unique id
username	varchar(50)		Username
email	varchar(50)		User email
pwd	varchar(10)		password

Task

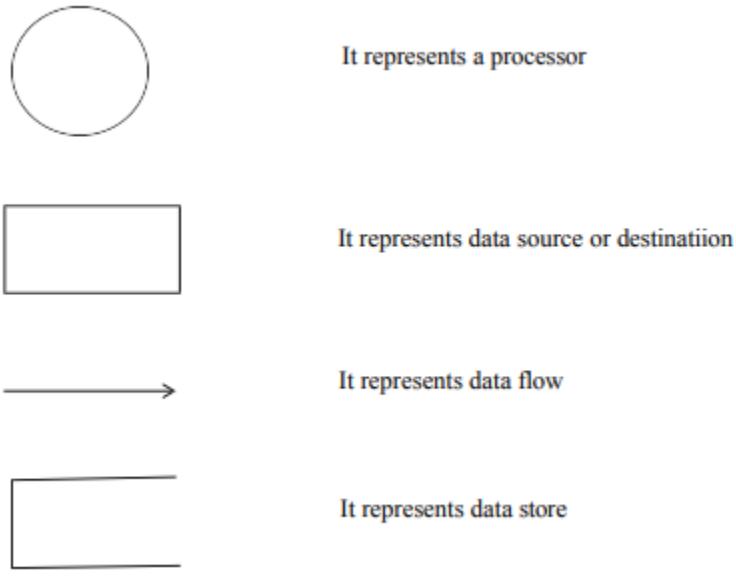
Field	Type	constraint	Description
Taskid	int(11)	ai,pk	Unique id
taskname	varchar(255)		Title of the task
description	Text		Task details
status	enum(not started,in progress,completed")		Task status
created_at	timestamp		Created time
updated_at	timestamp		Updated time
user_id	int(11)	fk	Foreign key(login)
priority	enum(low,medium,high)		Priority type
due_date	date		Due date

4.5.Data Flow Diagram

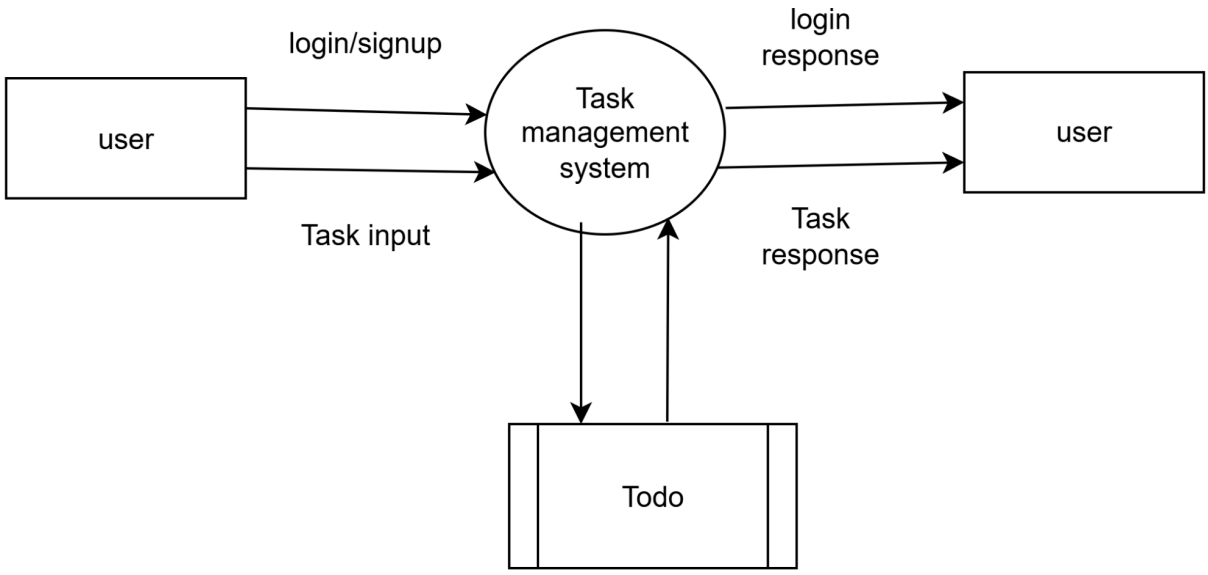
A Data Flow Diagram (DFD) is a graphical representation of the "flow" of data through an information system. A data flow diagram can also be used for the visualization of data processing. Data Flow Diagram is a common practice for a designer to draw a context-level Data Flow Diagram first which shows the interaction between the system and outside entities. A Data Flow Diagram is a network that describes the flow of data and processes that change, or transform, data throughout the system. This network is constructed by using a set of symbols that do not imply a physical implementation. It is a graphical tool for structured analysis of the system requirements. DFD models a system by using external entities from

which data flows to a process, which transforms the data and creates output data-flows which go to other processes or external entities or files. Data in files may also flow to processes as inputs. There are various symbols used in a DFD. Bubbles represent the processes. Named arrows indicate the data flow. External entities are represented by rectangles and are outside the system such as vendors or customers with whom the system interacts. They either supply or consume data are called sinks. Data is stored in a data store by a process in the system. Each component in a DFD is labelled with a descriptive name, Process names are further identified with a number. The Data Flow Diagram shows the logical flow of a system and defines the boundaries of the system. For a candidate system, it describes the inputs(source), all in a format that meets the user's requirements. The main merit of DFD is that it can provide an overview of system requirements, what data a system would process, what transformations of data are done, what files are used, and where the results flow.

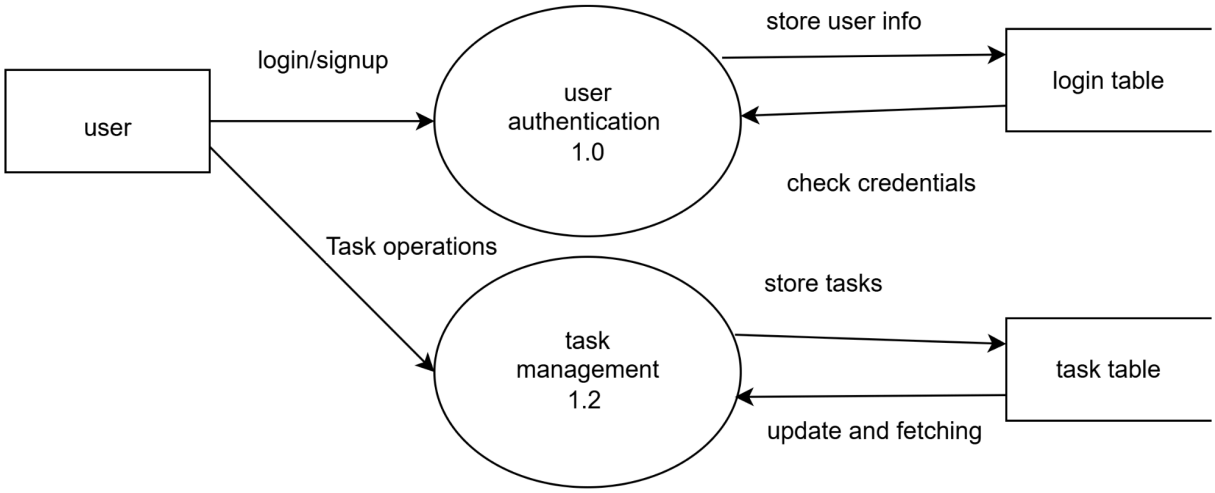
In the normal convention a DFD has four major symbols:



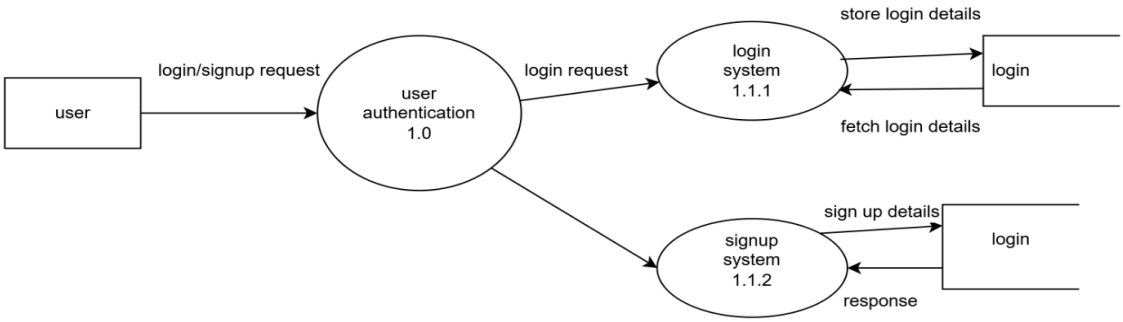
Level-0

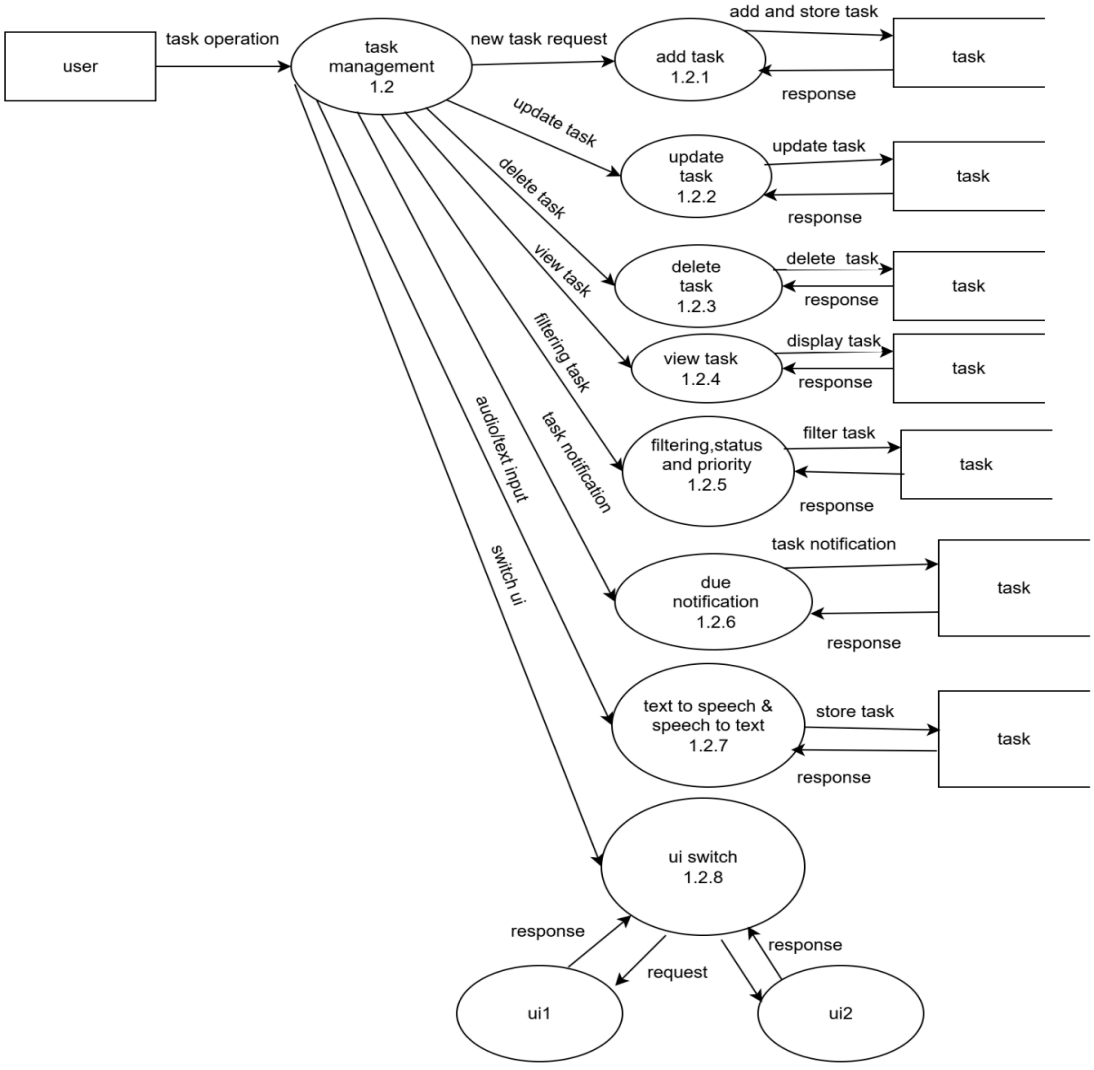


Level-1



Level-2


user
login/signup request
user
authentication
1.0
login request
login
system
1.1.1
store login details
login
fetch login details
signup
system
1.1.2
sign up details
login
response


user
task operation
task
management
1.2
new task request
add task
1.2.1
add and store task
task
response
update task
update
task
1.2.2
update task
task
response
delete task
delete
task
1.2.3
delete task
task
response
view task
view task
1.2.4
display task
task
response
filtering task
filtering,status
and priority
1.2.5
filter task
task
response
task notification
due
notification
1.2.6
task notification
task
response
audio/text input
text to speech &
speech to text
1.2.7
store task
task
response
switch ui
ui switch
1.2.8
response
request
response
ui1
ui2

4.5.Input Design

Overview

This section describes every input the user provides to Flowmark, the UI control used to collect it, client-side and server-side validation, typical error messages, and accessibility considerations.

---

1. Registration (Signup) — User inputs

Purpose: Create a new user account.

Fields & Controls

- `username` — `<input type="text">` — required, max 100 chars.
- `email` — `<input type="email">` — required, unique.
- `password` — `<input type="password">` — required, min 8 chars.
- `confirm_password` — `<input type="password">` — must match `password`.
- Submit — `<button type="submit">Sign up</button>`.

Client-side validation

- Required fields not empty.
- `email` matches email regex / HTML5 `type="email"`.

- `password` length >= 8 and `confirm_password === password`.
- Inline error messages near fields.

Server-side validation

- Trim inputs, validate email format, verify email uniqueness.
- Hash password using `password_hash()` before storing.
- Return structured errors (e.g., `{ field: "email", msg: "Email already in use" }`).

Common error messages

- "Please enter your name."
- "Please enter a valid email address."
- "Password must be at least 8 characters."
- "Passwords do not match."
- "This email is already registered."

Sample HTML snippet

```
<form method="post" action="/auth/signup.php">
  <label for="username">Full name</label>
  <input id="username" name="username" type="text" required
maxlength="100" />
  <label for="email">Email</label>
  <input id="email" name="email" type="email" required />
```

```
  <label for="password">Password</label>
  <input id="password" name="password" type="password" minlength="8"
required />
  <label for="confirm_password">Confirm Password</label>
  <input id="confirm_password" name="confirm_password"
type="password" required />
  <button type="submit">Sign up</button>
</form>
```

---

2. Login — User inputs

Purpose: Authenticate an existing user.

Fields & Controls

- `email` — `<input type="email">` — required.
- `password` — `<input type="password">` — required.
- Submit — `<button>Login</button>`.
- Optional: "Remember me" checkbox.

Validation

- Client: non-empty checks.
- Server: fetch by email, verify with `password_verify()`, on success set session.

Error messages

- "Invalid email or password."
- "Account locked after multiple failed attempts. Try again later."

---

3. Add Task — User inputs

Purpose: Create a new task.

Fields & Controls

- `taskname` — `<input type="text">` — required, max 255 chars.
- `description` — `<textarea>` — optional.
- `priority` — `<select>` — options: Low, Medium, High.
- `due_date` — `<input type="date">` — optional.
- `add_task` — `<button type="submit">Add Task</button>`.
- Optional: STT button (mic icon) to dictate title/description.

Client-side validation

- `taskname` required and trimmed.
- `due_date` valid date and (optional rule) not in the past.
- Show inline messages for invalid fields.

Server-side validation

- Re-validate all fields, ensure `taskname` non-empty, sanitized.
- Verify `user_id` from session to prevent spoofing.
- Use prepared statements for DB insert.

Error messages

- "Task title is required."
- "Invalid due date."
- "Unable to save task. Please try again."

STT (Speech-to-Text) usage

- Mic button triggers `webkitSpeechRecognition` or `SpeechRecognition`.
- On `result`, populate `taskname`/`description` fields (still run validation before submit).
- If recognition fails: show "Voice recognition failed — please type or try again."

Sample HTML (simplified)

```
<form method="post" action="tasks/todo.php">
  <input name="taskname" required maxlength="255" placeholder="Task title">
  <button id="micBtn" type="button">🎤</button>
  <textarea name="description" placeholder="Description"></textarea>
```

```
  <select
name="priority"><option>Low</option><option>Medium</option><option>
High</option></select>
  <input name="due_date" type="date">
  <button type="submit" name="add_task">Add Task</button>
</form>
```

---

4. Edit Task — User inputs

Purpose: Update existing task.

Fields & Controls

- Same as Add Task + hidden `taskid` input.
- Prepopulate fields with existing values.
- Submit to update endpoint.

Server-side checks

- Verify `taskid` exists and belongs to `session user`.
- Perform update via prepared statement.

Errors

- "Unauthorized to edit this task."
- "Failed to update task."

---

5. Delete Task — User input

Purpose: Remove a task.

Controls

- Delete button triggers confirmation modal: "Are you sure?"
- On confirm, send POST with `taskid`.

Server checks

- Verify user ownership before `DELETE`.

Messages

- "Task deleted successfully." or "Failed to delete task."

---

6. Search & Filter — User inputs

Purpose: Locate tasks.

Controls

- `search` text input.
- `status` select (All / Not Started / In Progress / Completed).
- `priority` select.
- Date range inputs.

Behavior

- Submit queries build parameterized SQL `WHERE` clauses.
- Support STT for filling search box via mic.

Errors

- "No matching tasks found."

---

7. UI Switching — User input

Controls

- Toggle link/button in navbar: links to `/ui1.php` or `/ui2.php`.
- No form data; optional query string preservation.

Behavior

- Clicking calls parent-level redirect file which redirects to other UI's `todo.php`.
- Optionally set cookie `preferred_ui` for persistence.

---

8. TTS (Text-to-Speech) Trigger — User input

Controls

- Button "Read Tasks" or similar.

Behavior

- On click, front-end fetches tasks and invokes `speechSynthesis.speak()` with a generated string.
- Provide Play / Pause / Stop controls.

Fallback

- If API unsupported, show textual transcript.

---

9. Profile & Account Inputs

Actions

- Update username.
- Change password (requires current password verification).
- Logout
- Delete account (confirm + password).

Security

- Re-authenticate for critical operations.

4.6.Output Design (User)

Overview

This section describes all outputs presented to the user after their inputs: pages, messages, read-aloud text, exports, and accessibility considerations.

---

1. Dashboard — User output

Purpose: Provide an overview after login.

Displayed items

- Summary counters: Total tasks, Completed, Pending, Overdue, Upcoming.
- Action buttons: Add Task, Read Tasks (TTS), Switch UI, Export.
- Recent tasks list/cards (UI-specific layout).
- Search/filter controls.
-

Sample textual output

You have 8 tasks — 3 Completed, 4 Pending, 1 Overdue.
Upcoming: 'Project report' due 2025-08-30

---

2. Task List / Task Card — User output

Elements for each task

- Title (link to edit/detail)
- Description (truncated)

- Priority tag
- Status badge
- Due date
- Action buttons: Edit, Delete, Mark Complete

Visual cues

- Overdue tasks highlighted in red.
- Completed tasks dimmed or struck through.

---

3. Add/Edit/Delete Confirmation — User output

Success messages (non-intrusive)

- "Task added successfully."
- "Task updated successfully."
- "Task deleted."

Error messages

- "Unable to save tasks. Please try again later."
- "You are not authorized to edit this task."

4. Search & Filter Results — User output

Outputs

- List or cards matching criteria.
- "No results found" message when empty.

Pagination

- Show page numbers and controls for large results.

---

5. UI Switch — User output

Behavior

- After redirect, user sees the other UI's dashboard (same data, different presentation).
- Optionally show a temporary message: "Switched to UI2 (card view)."

---

6. TTS Output — User output (spoken)

When user clicks "start recording"

- Front-end composes a readable summary string:
    - Example: "You have eight tasks. Task one: Buy groceries, due August twenty-eighth, priority high..."
- Use short sentences, explicit separators, and optional pause between tasks.

Controls

- Play / Pause / Stop buttons with clear labels .

Fallback

- If `speechSynthesis` not available, show textual transcript: "TTS not supported — here's a text version."

---

7. STT Output — User visible results

After speech recognition

- Recognized transcript populates the relevant input fields (`taskname`, `description`, or `search`).
- Show small confirmation: "Recognized: ‘Buy groceries tomorrow’ — [Use] [Retry]"
- If recognition low-confidence: indicate uncertainties and allow manual edit.

Errors

- "Voice recognition failed. Please try again."

5.System development

5.1.Introduction

System development is the process of defining, designing, testing and implementing a new software application or program. The Assignment Submission and Evaluation System undertaking encompasses meticulous steps, starting with requirement gathering and data collection, ensuring a systematic approach to achieving data consistency throughout the entire system development process.

1. Requirement Gathering:

Stakeholder Collaboration:

• Engaging with key stakeholders, including students,teachers, and administrators, to understand their unique needs and expectations.
• Conducting interviews, surveys, and workshops to gather insights into the desired functionalities and features.

Functional and Non-Functional Requirements:

• Identifying both functional requirements (features and capabilities) and non-functional requirements (performance, security, and usability).
• Prioritizing requirements based on criticality and impact on the user experience.

Use Cases and User Stories:

• Developing detailed use cases and user stories to capture the system's interactions from different user perspectives.
• Ensuring that every requirement aligns with the overarching goals of the Assignment Submission and Evaluation System.

2. Data Collection and Preparation

Data Sources Identification:

• Identifying the sources of data, including student information, assignment details, and grading criteria.
• Collaborating with existing educational databases or integrating with external systems for seamless data flow.

Data Schema Design:

• Designing a comprehensive data schema that reflects the relationships between different entities, such as students, assignments, grades, and user roles.
• Ensuring normalization to eliminate redundancy and maintain data consistency.

Data Validation and Cleansing:

• Implementing robust validation mechanisms to ensure the accuracy and integrity of the collected data.
• Incorporating data cleansing processes to handle inconsistencies or errors in the initial dataset.

3. Ensuring Data Consistency:

Referential Integrity:

• Enforcing referential integrity constraints to maintain the consistency of relationships between tables in the database.
• Preventing orphaned records and ensuring that each piece of data has a valid relationship.

5.2.MENU LEVEL DESCRIPTION

Below are the primary navigation/menu levels shown to users. Two UIs (UI1 / UI2) present the same menu logically though visually they differ.

A. Unauthenticated (Public)

- **Home** — project description or landing page.
- **Signup** — create account.
- **Login** — access account.

B. Authenticated (User)

- **Dashboard** — summary counts and recent tasks.
- **Tasks**
    - View All Tasks — list or cards.
    - Add Task — quick add form/modal.

- **Search / Filters** — quick access to search bar and filter menu.
- **Voice Controls**
    - Voice Add — mic button to initiate STT for adding tasks.
    - Voice Search — mic for searching tasks by voice.
    - Read Tasks — TTS control to read dashboard or selected tasks.
- **Settings / Profile** — update profile, change password, preferred UI (if stored).
- **Switch UI** — small link/button in header that calls `ui1.php` or `ui2.php` parent handler.
- **Logout** — terminate session.

5.3.PROCESS SPECIFICATION

Process specification is a method used to document, analyse and explain the decision-making logic and formulas used to create output data from process input data. Its objective is to flow down and specify regulatory/engineering requirements and procedures. High-quality, consistent data requires clear and complete process specifications .

1. Signup (Create Account)

- **Purpose:** Register a new user.
- **Preconditions:** None.
- **Input:** `username`, `email`, `password`, `confirm_password`.
- **Key steps:** client validation → POST to server → server validates email uniqueness → `password_hash()` → INSERT into `users`.
- **Output / Postcondition:** New user row created; redirect to login (or error message).

---

2. Login (Authenticate)

- **Purpose:** Authenticate user and start session.

- **Preconditions:** User registered.
- **Input:** `email`, `password`.
- **Key steps:** fetch user by email → `password_verify()` → on success `session_start()`.
- **Output / Postcondition:** Session established; redirect to dashboard; failure shows error.

---

3. Add Task

- **Purpose:** Create a task for the logged-in user.
- **Preconditions:** Authenticated (`$_SESSION['user_id']`).
- **Input:** `taskname` (required), `description`, `priority`, `due_date`. (Optionally via STT transcript.)
- **Key steps:** client validation → POST → server re-validate & sanitize → prepared `INSERT` into `task`.
- **Output / Postcondition:** Task saved; UI refreshed; success message or error.

---

4. Edit Task

- **Purpose:** Update an existing task.
- **Preconditions:** Authenticated; user owns the task.
- **Input:** `taskid`, updated fields.
- **Key steps:** verify ownership (`task.user_id == session user`) → validate inputs → prepared `UPDATE`.
- **Output / Postcondition:** Task updated; confirmation or error.

---

5. Delete Task

- **Purpose:** Remove a task.
- **Preconditions:** Authenticated; user owns the task.
- **Input:** `taskid` (confirm via modal).
- **Key steps:** confirm → verify ownership → prepared `DELETE`.
- **Output / Postcondition:** Task removed; list refreshed; confirmation or error.

6. Search & Filter

- **Purpose:** Retrieve tasks matching criteria.
- **Preconditions:** Authenticated.
- **Input:** `query`, `status`, `priority`, `date_from`, `date_to`. (STT may fill the query.)
- **Key steps:** validate inputs → build parameterized `SELECT` with active filters → return results .
- **Output / Postcondition:** Filtered task list (or “no results” message).

---

7. UI Switch (UI1 ⇄ UI2)

- **Purpose:** Change presentation layer.
- **Input:** Click link to `ui1.php` or `ui2.php`.
- **Key steps:** parent handler reads query string (optional) → optionally set cookie `preferred_ui` → `header("Location: ui2/tasks/todo.php")` (or reverse).
- **Output / Postcondition:** Browser redirected to other UI’s `todo.php`; same backend data displayed in alternate UI.

---

8. STT (Speech-to-Text)

- **Purpose:** Populate input fields via voice.
- **Preconditions:** Browser supports Web Speech API; mic permission granted.
- **Input:** Spoken audio.
- **Key steps:** start recognition (`SpeechRecognition` / `webkitSpeechRecognition`) → on `result` populate inputs → user edits/validates → submit as normal.
- **Output / Postcondition:** Form fields filled with transcript; fallback to manual input on failure.

---

9. TTS (Text-to-Speech)

- **Purpose:** Read tasks/dashboard aloud.
- **Preconditions:** Browser supports `speechSynthesis`.
- **Input:** User triggers “Read Tasks” or selects items.
- **Key steps:** fetch text summary → create `SpeechSynthesisUtterance` → `speechSynthesis.speak()`; provide pause/stop controls.
- **Output / Postcondition:** Auditory readout; if unsupported, show textual transcript.

---

10. Logout

- **Purpose:** End session securely.
- **Preconditions:** User authenticated.
- **Input:** User clicks logout.
- **Key steps:** `session_unset(); session_destroy();` → redirect to login.

- **Output / Postcondition:** No active session; protected pages require re-login.

6.System Testing

6.1 TESTING METHODS

The purpose of system testing is to identify and correct errors in the candidate system. Testing is an important element of the software quality assurance and represents the ultimate review of specification, design and coding. The increasing visibility of the software as a system element and the costs associated with a software failure are motivated forces for well planned, through testing. Different levels of testing were employed for software to make an error free, fault free and reliable.

UNIT TESTING

Unit testing focuses verification effort on the smallest unit of software designs the module. To check whether each module in the software works properly so that it gives desired outputs to the given inputs. All validations and conditions are tested in the module level in the unit test. Control paths are tested to ensure the information properly flows into, and out of the program unit and out of the program unit under test. In conclusion, all error handling paths are tested. The test data was given to each and every module in all respects and got the desired output. Each module has been then tested to check if it functions properly.

INTEGRATION TESTING

After testing all small units, all the small units are integrated and then the testing process is repeated. The major concerns of integration testing are developing an incremental strategy that will limit the complexity of the entire actions among components as they are added to the system. Developing components as they are added to the system, developing an implementation and integration schedules that will demonstrate the viability of the evolving system. Though each program works individually, they should work after linking them together. This is also referred to as interfacing. Integration testing is a systematic technique for constructing program structure while at the same time; it is also a conducting test to uncover errors.associated with the interface. In the testing, the programs
are constructed and tested in small segments.

VALIDATION TESTING

The validation test can be defined by the following simple definition, that validation succeeds when the software functions in a manner that can be reasonably accepted by the customer.After validation tests have been conducted one of the two.A possible condition exists.The function or performance characteristics are accepted and confirmed to specification.A deviation from specification is uncovered and defining list is created. System validation checks the quality of the software in both simulated (test data) and live environments (live data). First the software goes through a phase called alpha testing, in which errors and failures based on simulated user requirements are verified and studied. In this system, validator fields are there in each and every form. Validators such as Required field validator, compare validator, Regular expression validator etc. are set in this system and thus validation testing is conducted successfully.Validation testing is done in almost all forms. That is, in customer form, registration form, there are many fields where we have to provide only numerical values. Such fields are tested with codes which restrict the entry of characters. Also, there are some fields where numbers should not be entered. Such fields are also provided with code which restricts the entry of numerical values or symbols.

SYSTEM TESTING

The important and essential part of the system development phase, after designing and developing the software is system testing. Theoretically, a newly designed system should have all the parts or sub systems are in working order, but in reality each sub-system into one pool and test the whole system to determine whether it meets the user requirements. This is the last change to detect and correct errors before the system is installed for user acceptance testing. The purpose of testing is to consider all the likely variations to which it will be subjected and then push the system to it.

VERIFICATION TESTING

Verification testing is the method to ensure that the product satisfies the conditions those were imposed at the starting of the development phase. It is the process of evaluating The intermediary works products of a software development life cycle such as verification of documents, design, code and program to check if the developers are on the right track of creating the final product. It is generally done before validation testing and is done without executing the software. It uses static testing techniques or manual activities like inspections, reviews,breakthroughs, and Desk- checking etc. and is a low-level exercise. When verification tests whether the product is built in the right way, validation tests whether the product being built is the right one.

USABILITY TESTING

This testing technique verifies the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component. This testing process is usually performed by end users. It includes the testing of input and output forms. This Cyber spy was given to end users and the software was found to be user friendly.

REGRESSION TESTING

Regression testing is the repeated testing of an already tested program, after modification, to discover any defects introduced or uncovered as a result of the change. It is essential for software to be re-tested from time to time, just to make sure that new fixes don't destroy some other part of the system.Cyber spy has undergone regression testing with every modification and bugs introduced due to changes were removed.

TESTING ENVIRONMENT & TOOLS

Debugging (Black Box Testing & White Box Testing)

Debugging helps to avoid bugs in the program. It helps to ensure that the system produces the desired output and also helps to identify the cases when undesired outputs are produced so that the developers can decide how the unexpected outputs or exceptional cases must be handled. Debugging can be done using two testing strategies namely Black Box Testing and White Box Testing.

Black Box Testing

Black box testing is the Software testing strategy used to test the software without knowing the internal structure of code or program. This means that implementation knowledge is not required to carry out Black Box Testing. This type of testing is carried out by testers. It is also referred to as a functional test or external testing.

White Box Testing

White box testing is the software testing strategy in which internal structure is being known to the person testing the software. Generally, this type of testing is carried out by software developers. Implementation Knowledge is required to carry out White Box Testing. It is also referred to as structural test or interior testing.

6.2 TEST PLAN ACTIVITIES

Testing commences with the test plan and terminates with acceptance testing. A test plan is a general document for the entire project that defines the scope, approach to be taken and the schedule of testing as well as identifies the items for the entire testing process and the personal responsibilities for the different activities of testing. The test planning can be done in parallel with coding and designing activities. The inputs for formatting a test plan are Project plan Requirements document System design documents,The project is needed to make same that the data plan is consistent with the overall quality plan for the project and The testing schedule matches with that of the project plan. The requirements documents and the design documents are the basic documents used for selecting the text units and deciding the approaches to be used during testing.

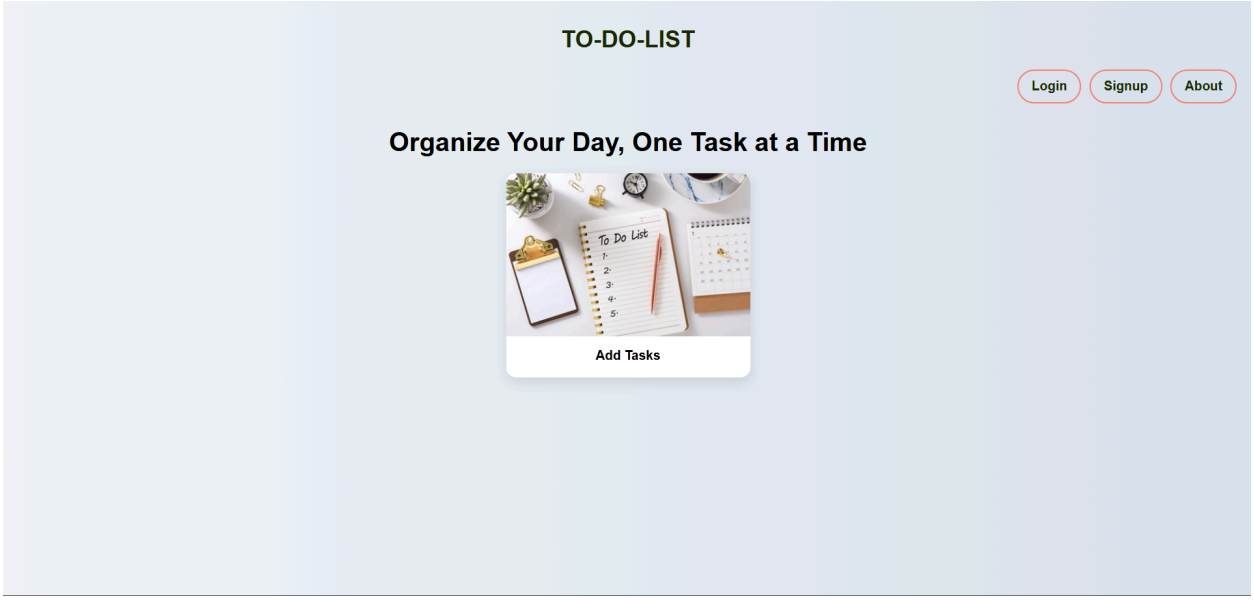
A test plan should contain the following activities:

- Test unit specification
- Features to be tested
- Approach for testing
- Test deliverables
- Personal allocation

One of the most important activities of the test plan is to identify the test units. A test unit is a set of one or more modules, together with associated data, that form a single computer program and that is the object of testing. Test units can occur at any level and contain from a single module to the entire system. Thus, the test unit may be used during the testing activity. The levels are specified in the test plan by identifying the test units for the project. Different units are specified for unit integration and system testing.Features to be tested include all software features and combinations of features that should be tested. Software may include functionality, design constraints and attributes. The test plan, if it is a document separate from the project management plan, typically specifies the schedule that includes the amount of time and effort to be spent on different activities of testing. This schedule should be consistent with the overall project schedule.

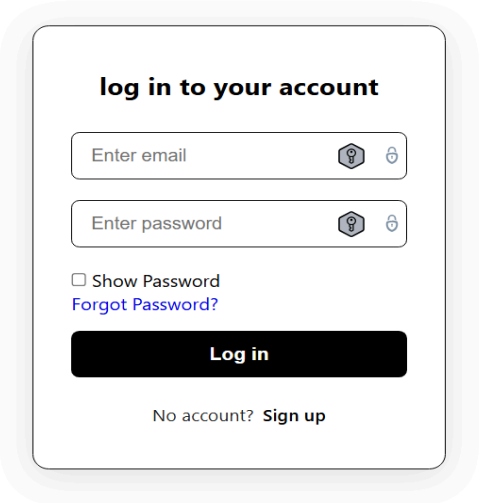
6.3.Screen Layouts

1.Homepage

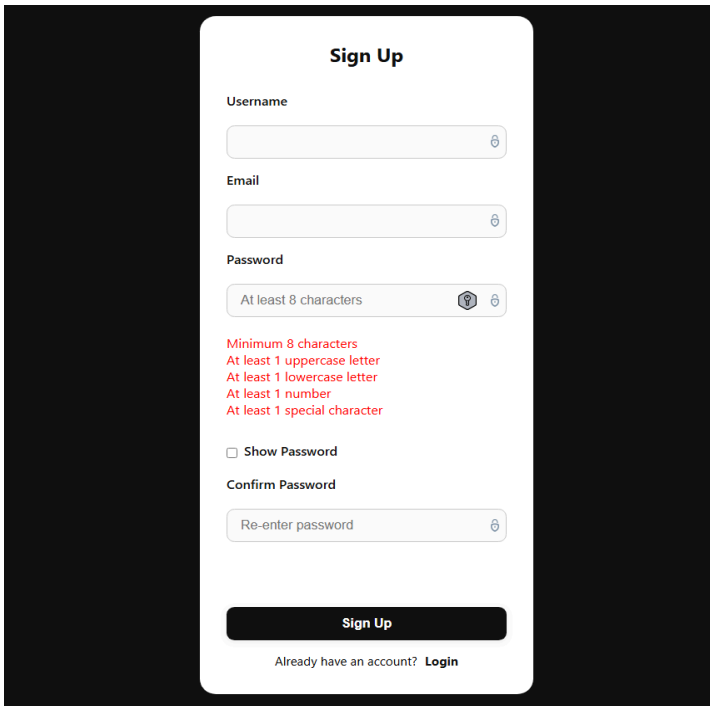


2.Login and signup

2.1 login

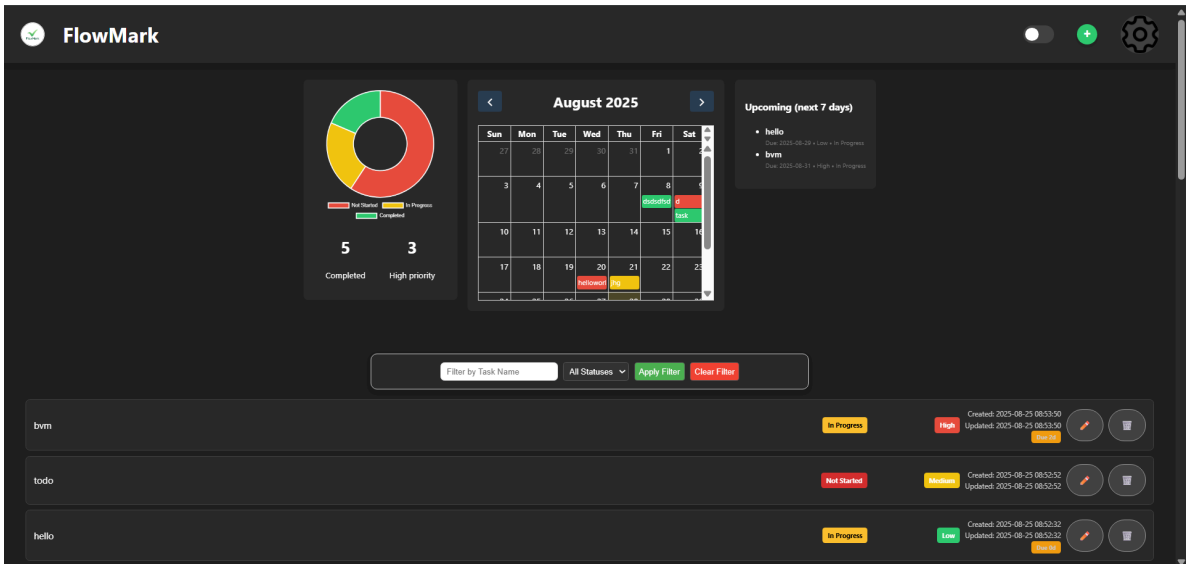


2.2 Signup

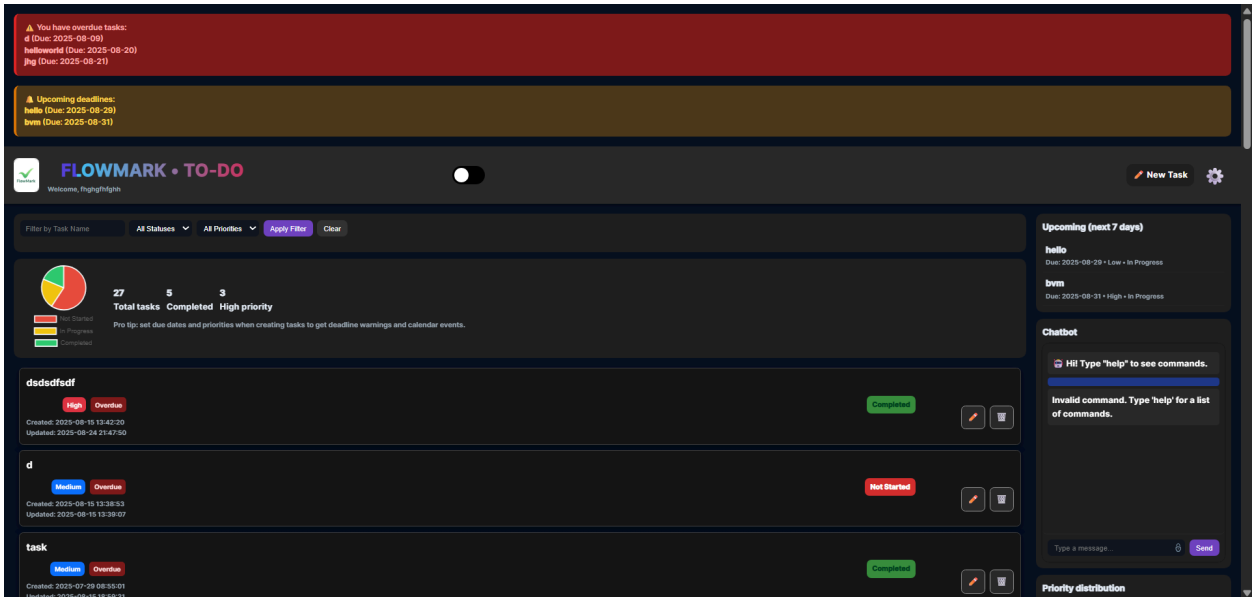


3.Main dashboard

3.1 Ui-1

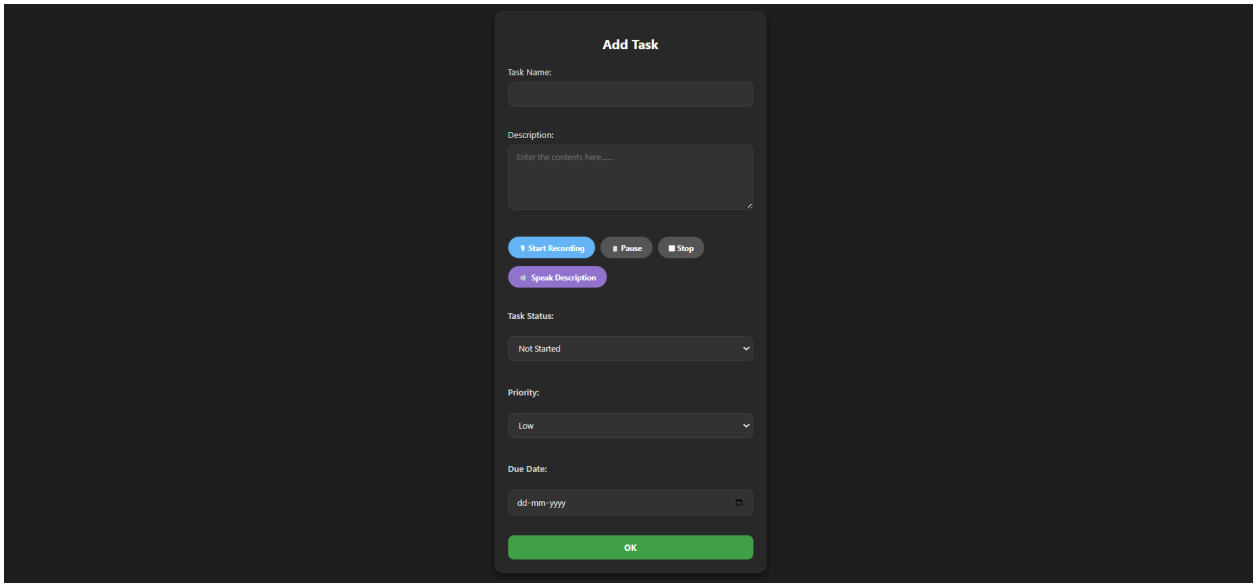


3.2 Ui-2

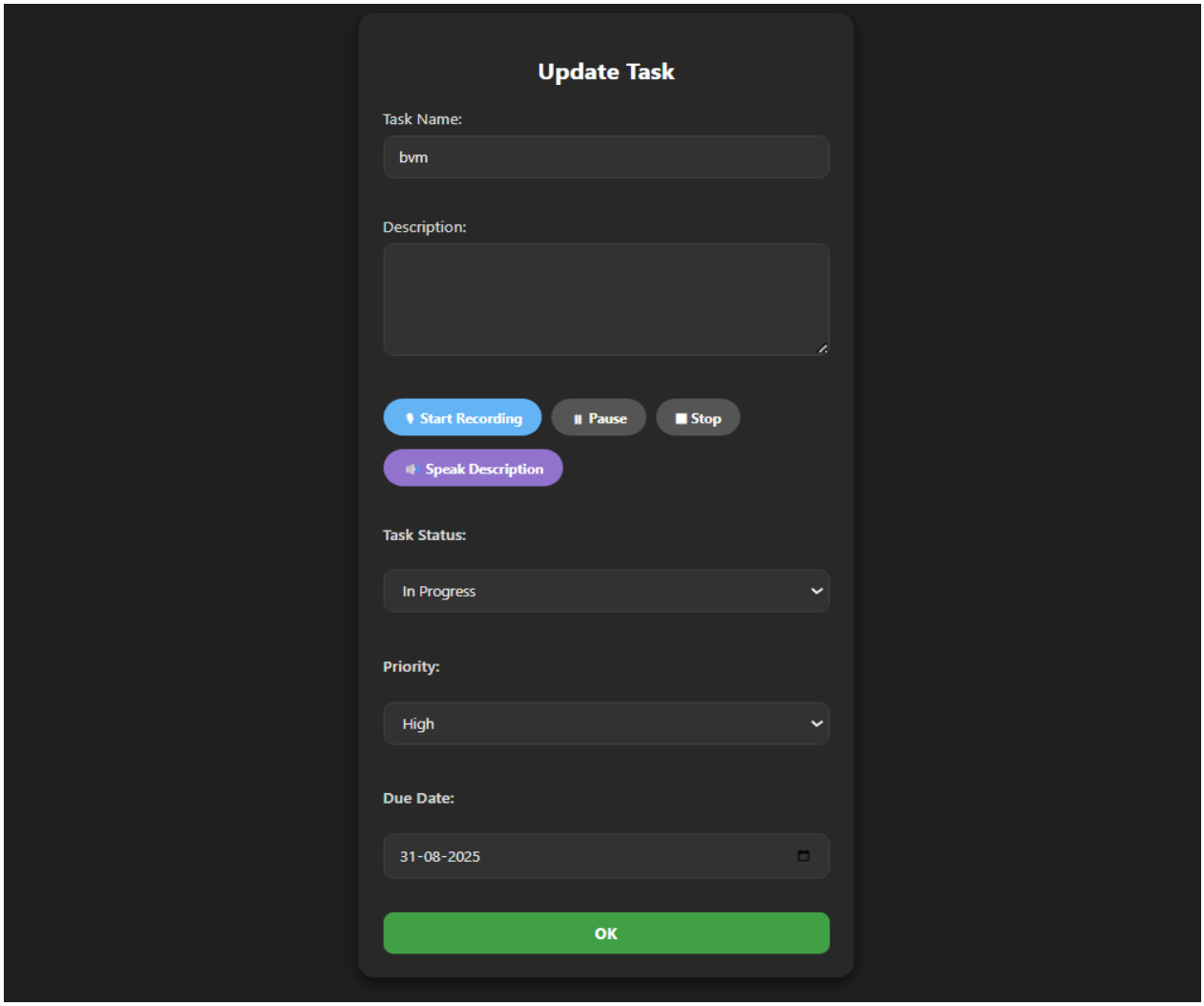


4.Add and update Task

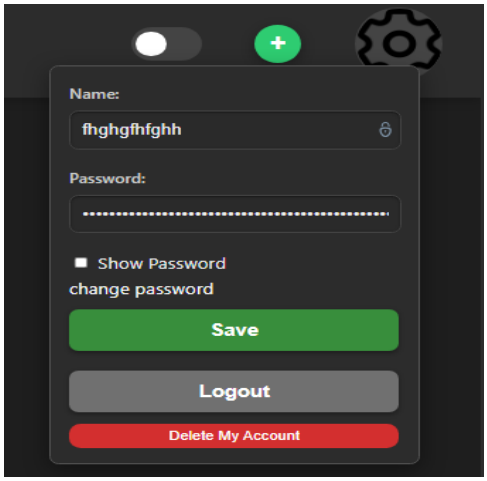
4.1 Add task



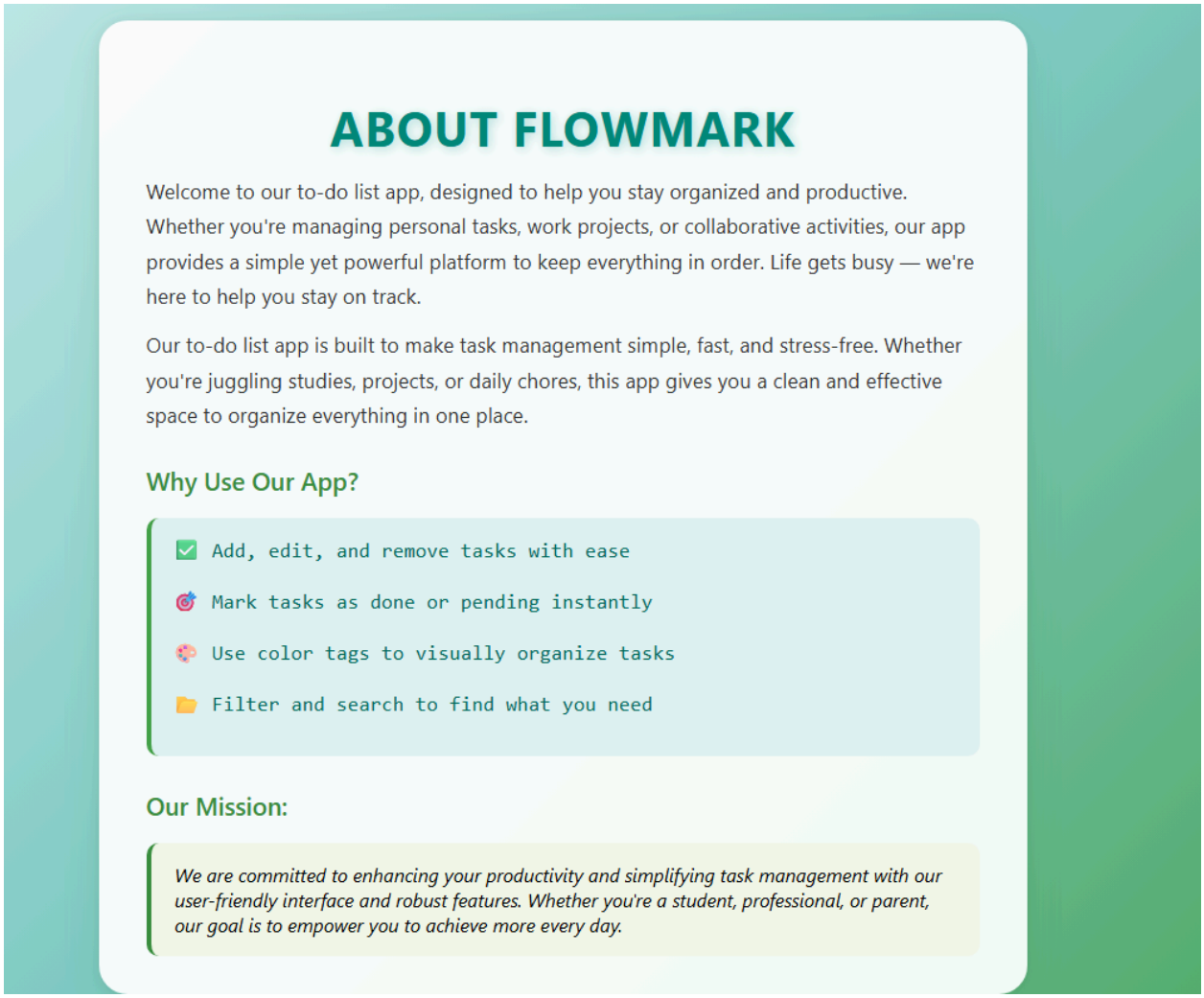
4.2 Update Task



5.Profile setting



6.About



7.System Implementation

System implementation is the process of taking a software system from the design phase to a fully functional and operational state. The following are common activities involved in system implementation:

1. Installation: This activity involves installing the software system on the hardware environment where it will be used.

2. Configuration: This activity involves configuring the software system to meet the specific requirements of the organization or end-users.

3. Data Migration: This activity involves transferring data from the old system to the new system.

4. Testing: This activity involves testing the software system to ensure that it functions as expected and meets the requirements and specifications.

5. Training: This activity involves training end-users and administrators on how to use the new system.

6. Documentation: This activity involves creating documentation for the new system, including user manuals, technical documentation, and support materials.

7. Deployment: This activity involves deploying the new system in the production environment.

8. Maintenance: This activity involves ongoing maintenance and support of the system to ensure that it continues to function properly.

9. Monitoring: This activity involves monitoring the system to identify and resolve any issues that arise.

10. Upgrades: This activity involves upgrading the system to new versions or releases as they become available.

11. Support: This activity involves providing ongoing support to end-users and administrators to address any issues that arise and ensure the system is functioning as intended. Successful system implementation requires careful planning, effective project management, and collaboration among all stakeholders. It is critical to ensure that the system meets the requirements and specifications, is reliable and secure, and is easy to use and maintain.

8.Conclusion And Scope For Future Enhancement

Conclusion

The Flowmark project successfully demonstrates the design and implementation of a functional task management system built on a PHP–MySQL backend with two interchangeable UIs, integrated speech-to-text (STT) and text-to-speech (TTS) features.

The project applied the Prototype Model, enabling rapid development, early user feedback, and iterative refinement of the system.

Core objectives were achieved:

- Secure authentication with session handling.
- Task CRUD (create, read, update, delete) with search and filter capabilities.
- A dual-interface design (UI1 and UI2) accessible via a parent-level UI switcher.
- Voice-based interaction (STT for input, TTS for accessibility).
- Data persistence and integrity maintained through MySQL schema design and validation.

Flowmark demonstrates how web technologies (HTML, CSS, JavaScript, PHP, MySQL) can be combined with browser-native voice APIs to create a lightweight yet feature-rich productivity application. The prototype is reliable for individual use cases and suitable as a foundation for future academic or professional projects.

Scope for future enhancement

While Flowmark meets its initial objectives, there are several opportunities for further improvement and expansion:

1. User Experience Enhancements
    - Persist UI preference in the database (`preferred_ui`) and restore on login.
    - Provide drag-and-drop task reordering and bulk actions (mark complete, delete multiple).
    - Add theme customization (dark/light mode).
2. Advanced Task Features
    - Support recurring tasks and reminders with notification integration.
    - Implement task prioritization with color-coded categories or Kanban view.
    - Allow file attachments or subtasks for complex workflows.
3. Collaboration Features

- Extend the system for group use with shared task lists.
- Real-time updates using WebSockets (multi-device sync).

4. Voice Features Expansion
    - Integrate offline STT/TTS engines for browsers without Web Speech API.
    - Support natural-language parsing (e.g., “remind me tomorrow at 5 PM”).
    - Add multilingual STT/TTS for global users.
5. Security and Performance
    - Implement CSRF protection and two-factor authentication.
    - Add password reset with secure tokens.
    - Optimize queries and caching for larger datasets.
6. Deployment & Scalability
    - Containerize the system with Docker for portability.
    - Host on cloud platforms (AWS, GCP, or DigitalOcean) for wider use.
    - Add automated backup and monitoring tools.

9.Biblography

1.Open Ai gpt 4o,4.1

2.Gemini 2.5 flash and Gemini 2.5 pro

3.youtube
brocode,coding2go