STUDYMATE: A SMART MOBILE TASK MANAGER FOR PEAK STUDENT PRODUCTIVITY

By

Mandy Teoh Jiayi

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION SYSTEMS (HONOURS)
BUSINESS INFORMATION SYSTEMS
Faculty of Information and Communication Technology
(Kampar Campus)

JUNE 2025

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Mr. Yong Tien Fui, for providing me with invaluable guidance, encouragement, and support throughout the course of my final year project. His expertise, constructive feedback, and dedication were fundamental in the successful completion of this project. I am truly grateful for the opportunity to work under his mentorship, and I have learned a great deal from his insights.

A special thank you to my family for their unwavering love, patience, and support, especially during the difficult times when I faced stress and challenges. Their constant encouragement, understanding, and presence by my side were my pillars of strength, and I cannot express enough how much it means to me. They have always been my source of motivation, and I truly appreciate everything they have done for me.

Finally, I would like to thank everyone who indirectly supported me, whether through their advice or being part of my journey.

COPYRIGHT STATEMENT

This Final Year Project proposal is submitted in partial fulfillment of the requirements for the degree of Bachelor of Information Systems (Honours) Business Information Systems at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project proposal represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project proposal may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ABSTRACT

This project aims to address critical challenges in academic task management through the development of an innovative task management system tailored specifically for any level of students who need a personalized assistant with their studies. Many existing productivity tools fall short in key areas such as task prioritization, real-time collaboration, and progress tracking, leading to confusion, miscommunication, and increased stress. This system introduces several advanced features, including a dynamic task prioritization algorithm, real-time collaboration capabilities, and comprehensive project tracking and visualization tools. The system seeks to enhance students' organizational efficiency, improve collaboration, and provide clear visibility into task progress. The project employs a prototyping approach and utilizes modern technologies to develop a robust platform that addresses the unique needs of students in managing their academic workloads.

Area of Study: Mobile Application Development, Productivity and Time Management

Keywords: Dynamic Weighted Task Prioritization Algorithm, Earliest Deadline First (EDF), Task Completion Detection, Mobile Task Management, Student Productivity App

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

EDF	Earliest Deadline First
TCD	Task Completion Detection
CSP	Completion of Structured Processes
MoE	Mixture-of-Experts
STT	Speech-to-Text
ASR	Automatic Speech Recognition
STFT	Short-Time Fourier Transform
MFCCs	Mel-Frequency Cepstral Coefficients
RNN-T	Recurrent Neural Network Transducer
VAD	Voice Activity Detection

CHAPTER 1
Introduction

In today's fast-paced academic environment, students face a multitude of challenges in managing their workload effectively. Task management, which encompasses organizing, prioritizing, and tracking tasks, is crucial for academic success. However, many existing productivity tools fall short in addressing the unique needs of students, resulting in widespread issues such as procrastination, disorganization, and inefficient collaboration.

Procrastination, a tendency to delay tasks in favor of more immediately gratifying activities, significantly impacts students' productivity [1]. This tendency is often exacerbated by the inability to effectively prioritize tasks, which can lead to confusion and a lack of clear direction. Current tools, such as Apple Reminders [2] and Microsoft To-Do [3], offer basic functionalities but fail to address the complexities of task prioritization when multiple tasks share the same priority level. These limitations result in students overlooking urgent tasks or struggling to determine which task to tackle first.

Moreover, real-time collaboration is an integral part of academic life, especially for group projects and collaborative assignments. Despite the importance of effective communication and document management, many existing tools lack robust collaboration features. For example, Microsoft To-Do does not support real-time comments on tasks or the integration of shared document links, leading to fragmented communication and inefficiencies [4]. This fragmentation forces students to rely on multiple platforms for communication and document management, complicating teamwork and reducing overall productivity.

Another critical issue is the lack of visibility into task status and project progress. Tools like Todoist offer basic task categorization but fall short in providing detailed insights into task completion and project status [3]. Without a clear understanding of where they stand on their tasks, students may experience heightened stress, missed deadlines, and last-minute rushes. The absence of advanced tracking and visualization features prevents students from monitoring their progress accurately and managing their workload effectively.

In light of these challenges, there is a pressing need for a task management system specifically designed to address these issues. This proposal aims to develop a comprehensive system that incorporates dynamic task prioritization, seamless real-time collaboration, and advanced

project tracking to improve students' task management efficiency and collaborative effectiveness. By addressing the shortcomings of existing tools, this system will provide students with a more effective solution for managing their academic responsibilities, ultimately enhancing their organizational skills and reducing stress.

1.1 Problem Statement and Motivation

Managing academic responsibilities effectively is a crucial skill for students, yet existing task management tools often fall short in addressing their specific needs. The overwhelming nature of multiple deadlines, projects, and assignments necessitates a structured, intuitive system for task organization and collaboration. However, current tools present three major limitations that hinder students' efficiency and productivity.

Problem Statement #1: Ineffective Task Organization and Prioritization

Effective task organization and prioritization are critical for students to manage their academic workload efficiently. However, students often face challenges when it comes to distinguishing the importance of various tasks, especially when multiple assignments are deemed equally important. This can lead to confusion and a lack of clear direction. Apple Reminders [1] has the functionality to set tasks as low, medium, and high priority; however, issues occur when more than one task has the same priority level. The system will not rearrange or reorganize the tasks based on other applied conditions, such as deadlines. This limitation can result in students overlooking urgent tasks or struggling to determine which task to tackle first.

Problem Statement #2: Limited Real-Time Collaboration and Document Integration

Collaboration is an essential aspect of student life, particularly for group assignments and projects. However, many existing tools lack robust collaboration features, making it difficult for students to stay on the same page. This can result in miscommunication, duplicated efforts, and missed deadlines. Microsoft To Do [2] lacks the ability to add real-time comments on tasks, preventing team members from providing instant feedback, discussing details, or asking questions directly within the task. This limitation forces users to rely on separate communication channels like WhatsApp and WeChat, leading to fragmented conversations and misunderstandings. Additionally, users cannot add shared document links directly into the workspace, requiring them to store and manage task-related documents in separate channels. This disjointed approach complicates task management and hampers efficient collaboration.

Problem Statement #3: Lack of visibility into task status and project progress

A significant issue students face is the lack of visibility into task status and project progress. Without a clear understanding of where they stand on their assignments and projects, students can easily fall behind or become overwhelmed. This lack of insight can lead to last-minute rushes, incomplete work, and heightened stress levels. Currently, tools like Todoist [3] allow for basic task categorization, but they fall short in several critical areas. For example, it does not offer features to categorize each task with detailed statuses such as "in progress" or to show granular completion levels like 50% or 80%. This absence of detailed task status tracking prevents students from monitoring their progress accurately and understanding which tasks require immediate attention.

1.2 Project Objectives

Objective #1: Enhance Task Organization and Prioritization Using Dynamic Weighted Task Prioritizing Algorithm

To improve how students manage their tasks, a sophisticated task management system will be developed that dynamically rearranges and prioritizes tasks based on multiple factors, such as deadlines, dependencies, and workloads. This multifaceted system will not just categorize tasks by priority but will also account for overlapping deadlines and task dependencies, ensuring students tackle the most urgent and critical assignments first. The platform will feature detailed task statuses (e.g. "in progress" or "awaiting review") and offer intuitive visualizations like progress bars and charts to keep users informed about the progress and upcoming deadlines. By providing real-time updates on task status and a clear visual representation of workload, students can better manage their time, avoid task overload, and reduce stress.

Objective #2: Enhancing Real-Time Collaboration with Unified Messaging and File Integration with Smart Search

Effective collaboration is vital, especially in group projects and shared academic assignments. To strengthen teamwork, the system will include real-time commenting directly within tasks, allowing students and collaborators to exchange feedback, ask questions, and resolve issues without relying on external platforms. In addition, a comprehensive messaging feature will be implemented, supporting both text and voice input. Voice recordings will be automatically transcribed into text for accessibility, ensuring that spoken contributions are searchable and

readable by all team members. The system will also provide a powerful search function that scans across all forms of communication in the comment section including text messages, voice messages, attached files, and shared URLs. This enables users to quickly locate important information without manually browsing long discussion threads. Alongside this, shared document links and attachments will be integrated into the workspace, ensuring that all resources and discussions remain centralized. By consolidating real-time communication, transcription, and advanced search capabilities into one platform, the system minimizes fragmentation, reduces information loss, and promotes efficient, transparent collaboration.

Objective #3: Implementing Smart AI Assistant for Daily Planning and Project Oversight

This objective not only focuses on enhancing visibility into task progress and overall project status through real-time tracking and visualization but also integrates a Smart AI Assistant to act as a virtual secretary. The AI assistant leverages user data including tasks, schedules, and deadlines, etc. to suggest personalized daily plans, highlight urgent priorities, and recommend optimal time allocations. It will allow users to interact naturally through text or speech, enabling commands such as “Plan my day” or “Reschedule my meeting”. Next, the project tracking system will continue to provide granular completion levels (e.g., 50% or 80% complete), progress dashboards, and timeline views to help users monitor workload and deadlines. Alongside this, the AI assistant will proactively organize schedules, suggest focus sessions, and remind users of pending obligations, ensuring that their daily workflow is both efficient and manageable. By combining visualization tools with an intelligent personal assistant, users will gain not only awareness of their progress but also actionable guidance to stay organized and productive.

1.3 Project Scope and Direction

The project aims to develop a specialized mobile-based task management system that addresses common challenges faced by students, professionals, and teams in managing their daily workload. This system will provide a tailored solution to these issues by enhancing overall task management and productivity. The system will offer a dynamic task management framework that automatically arranges tasks based on urgency, deadlines, and workload, ensuring users focus on what matters most. It will also include a unified collaboration space with messaging and file-sharing features, supporting both text and voice input. Voice recordings will be transcribed into searchable text, and an integrated search function will allow users to find any

comment, attachment, or URL shared in the workspace. Finally, the system will feature a smart AI assistant that acts like a personal secretary, helping users plan their day, reschedule tasks, and receive personalized recommendations through natural interaction.

Target user:

- Individual students who need a structured approach to managing academic workloads.
- Student groups collaborating on academic assignments and projects.
- Professionals and teams who require better organization, communication, and planning tools.

Since each individual has a unique pace and working style, this system will not impose any standardize performance metrics. Instead, it will serve as a flexible and adaptable tool to help users organize their tasks according to their own preferences.

1.4 Contributions

The proposed task management system introduces several innovative features and advanced technologies aimed at significantly enhancing students' productivity and collaboration in handling academic workloads. By addressing key challenges such as missed deadlines, inefficient teamwork, and disorganized tasks, this platform provides a smart, dynamic, and user-friendly solution that caters specifically to the needs of students, study groups, and possibly university lecturer.

Contribution #1: Dynamic Task Prioritization and Scheduling to Prevent Missed Deadlines, Addressing the First Problem Statement, Ineffective Task Organization and Prioritization

Our system leverages a sophisticated dynamic weighted task prioritization algorithm that adjusts task hierarchies based on multifaceted factors such as deadlines urgency, dependencies, and individual workload which is addressing the first problem statement. Unlike static task management systems, this adaptive algorithm ensures that students focus on the most urgent and important task first, preventing last-minute rushes and helping them stay on track [5]. Addressing the challenge of managing and scheduling tasks efficiently, this advance cost-efficient task prioritization algorithm based on Earliest Deadline First (EDF) principles as studied by Spuri and Buttazzo [6], ensures that tasks with the nearest dues dates receive the

highest priority. By dynamically adjusting to new tasks and deadlines, our system reduces delays and minimizes the risk of missing critical submissions. This approach helps students manage their workload effectively, prevent last-minute rushes, and meet deadlines with reduced stress.

Contribution #2: Seamless Collaboration Framework to Enhance Teamwork Efficiency, Addressing the Second Problem Statement, Limited Real-Time Collaboration and Document Integration

Another major contribution is the seamless collaboration framework that are specifically designed for group projects. This is due to group projects often suffer from miscommunication and inefficient task distribution, which can hinder productivity. To address the problem stated in the second statement, our system includes a real-time task collaboration feature, allowing students to communicate, assign responsibilities, and track progress directly within their workspace. The integration of comment section with chat functionality and document-sharing features ensures that discussions and resources are centralized, eliminating the need for external apps and ensuring all team members stay aligned and productive throughout the collaboration.

Additionally, the collaboration feature will be expanded into a unified messaging system that supports text, voice, file attachments, and shared URLs. Voice recordings will be automatically transcribed into searchable text, ensuring accessibility and convenience. A universal search function will allow users to locate any past message, transcription, or file in the comment section, reducing information loss and enabling faster retrieval of important details.

Contribution #3: Comprehensive Task Tracking and Visualization to Minimized Disorganization, Addressing the Third Problem Statement, Lack of Visibility into Task Status and Project Progress

To enhance visibility into task progress and overall project status, our system includes a comprehensive project tracking and visualization framework. This feature provides granular completion levels and a unified dashboard for monitoring all ongoing projects and tasks. We have incorporated visual aids such as dashboards and progress bars, which offer students a detailed understanding of task interconnections and project timelines. This advanced tracking system aids in better prioritization and helps students manage their projects with greater precision.

Our system also integrates User Device Calendar to streamline task management related to scheduled events. This feature enables users to schedule tasks and events efficiently, while automatic updates ensure that deadlines and important academic activities are never missed. Additionally, our system supports the Completion of Structured Processes (CSP), breaking down complex assignments into smaller, manageable milestones with clear tracking of progress until completion [7].

A novel aspect of our system is the integration of Task Completion Detection (TCD) to enhance task management. This feature automatically tracks and updates task progress, providing students with timely feedback and insights into their workload. By detecting completed tasks in real-time, the system ensures that students stay organized and aware of their progress. This approach helps them maintain efficiency, avoid redundant work, and stay motivated throughout their academic journey [7].

Contribution #4: Smart AI Assistant for Personalized Daily Planning, Addressing the First and Third Problem Statement

Finally, the system introduces a Smart AI Assistant that functions as a virtual secretary. These assistant leverages user data including tasks, subtasks, and schedules to generate personalized daily plans, suggest focus sessions, and highlight urgent priorities. Users can interact naturally through text or voice, asking the assistant to schedule meetings, reschedule deadlines, or recommend optimal study/work blocks. By combining dynamic task awareness with progress visibility and proactive guidance, the AI assistant transforms the platform from a static task tracker into a dynamic productivity partner, empowering users to manage their day with confidence and reduced stress.

1.5 Report Organization

This report is structured to guide readers through the development of our task management system for students. Chapter 1 (Introduction) establishes the overall purpose and direction of the project including background of the study, the problem statement and motivation, project objectives, scope, target users, and expected contributions. Chapter 2 (Literature Review) reviews the technologies adopted in this project, highlights the limitations of previous studies, proposes solutions to overcome them, and compares the proposed system against existing applications. Chapter 3 (Proposed Method/Approach) describes the methodology employed in the development of the system, including system design diagrams. It also explains the prototyping methodology used, discusses implementation challenges and issues encountered during development, and presents the project timeline. Chapter 4 (System Evaluation and Discussion) provides the evaluation and testing of the system through methods such as black-box testing, survey-based client satisfaction analysis, and performance benchmarks. The chapter also discusses the results obtained, highlights the challenges faced, and evaluates the extent to which the stated objectives have been achieved. Finally, Chapter 5 (Conclusion) summarizes the project's overall achievements, reflects on the success of the objectives, and discusses future work and potential enhancements to improve the system further.

CHAPTER 2
Literature Reviews

2.1 Review of Technologies

2.1.1 Hardware Platform

The hardware for this project includes a laptop, serving distinct roles in the development process. The laptop is the primary device used for writing, running, and testing the software, providing the necessary computing power to handle the integrated development environment (IDE), database management, and simulation tools. It allows developers to code, debug, and implement the features of the task management system.

Description	Specifications
Model	Matebook D 15 (Boh-WAQ9R)
Processor	AMD RYZEN 5
Operating System	Windows 10
Graphic	NVIDIA GeForce RTX 3070, 8 GB GDDR6
Memory	8GB RAM
Storage	512GB NVME SSD

Table 2.1.1 Specifications of laptop

2.1.2 Firmware/Operating System

The project was developed primarily on Windows 10, which offered stable support for the Flutter SDK, Android Studio, and Firebase CLI. Visual Studio Code served as the main code editor, while Android Studio was used exclusively for running mobile emulators. For mobile testing, Android Studio's emulator runtime (Ladybug Feature Drop 2024.2.2) is used to simulate real Android devices, ensuring the system's features can be tested effectively before deployment. The development environment thus ensured compatibility with required frameworks, while also allowing real-time debugging through Android Debug Bridge (ADB) for both emulated and physical devices.

2.1.3 Database

The project uses Google Firebase Firestore as its primary database. Firestore is a NoSQL cloud database that provides real-time data synchronization across clients, supporting offline persistence and scalability. It is well-suited for mobile task management applications as it allows efficient storage and retrieval of hierarchical data, such as schedules, tasks, subtasks, collaborators, and notifications. Firestore's flexible schema enables the dynamic storage of various data types, which is important for features like prioritization, messaging, and schedule tracking.

Primary store: Firebase Firestore (NoSQL, document/collection model).
Object store: Firebase Storage (enabled under the paid tier with budget constraints).
Collections: users, notifications, schedules, todos (subcollections: comments, voices, attachments), categories, completed_tasks

In addition to Firestore, Firebase Storage was utilized to manage binary objects such as voice recordings and file attachments. Since this project operates under a constrained billing plan, several cost-control measures were implemented, including file size restrictions, format validation, and lifecycle policies to delete temporary or unused files. Together, Firestore and Storage provided a balance between structured data management and flexible multimedia storage. Firebase Authentication was used to manage user accounts securely, while Firebase Cloud Messaging (FCM) supported the delivery of task reminders, assistant responses, and collaborative updates.

2.1.4 Programming Language

The client application was developed in Dart using the Flutter framework. Flutter's widget-based architecture enabled the design of a consistent and responsive user interface across Android devices. Its integration with Firebase plugins provided seamless support for authentication, real-time updates, and notifications. On the backend, Node.js was used within Firebase Cloud Functions to implement serverless operations such as handling voice transcription requests and managing interactions with the Google Generative AI API. To ensure version control and collaborative development practices, all source code was pushed to GitHub. GitHub served not only as a repository for maintaining different versions of the system but also as a backup solution for safe storage and synchronization across development environments.

2.1.5 Algorithm

I. Dynamic Reordering Algorithm

This approach introduces a real-time schedule reordering mechanism that organizes all schedules entries from present to future based on the scheduled start date. The aim is to increase clarity and usability by ensuring that ongoing and recent events appear, followed by upcoming ones.

Schedules are categorized and sorted dynamically using the following formula:

$$If\ T_{start} \leq T_{now} \rightarrow Present$$

$$If\ T_{start} > T_{now} \rightarrow Future$$

Where:

- T_{now} = Current Time
- T_{start} = Schedule's start time

Behavioural Logic			
Condition	**Category**	**Placement**	**Sorting Criteria**
$T_{start} \leq T_{now}$	Present	Appears at the top	Ascending $startDate$
$T_{start} > T_{now}$	Future	Appears below present	Ascending $startDate$
$startDate == null$	Present	Appears at the very top	Treated as $currentDate$

Table I.I Dynamic Reordering Algorithm Behavioural Logic

This dynamic sorting approach improves chronological clarity, ensuring users see their most relevant schedules first. Besides, it supports real-time interaction, requires no manual refresh, and adapts to both user changes and time-based transitions naturally [20 -> 8].

II. Dynamic Weighted Task Prioritization Algorithm

This approach focuses on automatically evaluating and prioritizing tasks using a scoring model that adapts in real-time based on urgency, workload, task type, and current task status [21 -> 9]. Hence, higher score will be prioritized first.

Each task is assigned a dynamic priority score $\boldsymbol{P}$ calculated as:

$$P = (W_d \times U_d) + (W_l \times U_l) + (W_t \times U_t) + (W_s \times U_s)$$

Where:

- W_d: Weight of Deadline Urgency
- U_d: Urgency score based on proximity to the deadline (closer deadline = higher score)
- W_l: Weight of Estimated Workload
- U_l: Number of subtasks or estimated effort (scaled 0–1 based on max workload)
- W_r: Weight of Selected Reminder Urgency
- U_r: Score based on alert setting (More proactive reminders = higher score)
- W_p: Weight of Selected Priority Level
- U_p: Score based on High/Medium/Low selection

Time Remaining Until Deadline	$\boldsymbol{U_d}$
Within 1 hour	1.0
Within 1 day	0.8
Within 1 week	0.5
Within 1 month	0.3
Beyond 1 month	0.1

Table II.I Deadline Urgency Score

Number of Subtasks	$\boldsymbol{U_l}$
10 or more	1.0
7-9	0.8
4-6	0.6
2-3	0.4
1	1.2
0	0.0

Table II.II Subtasks Scores

Alert Setting	U_r
At time of event or None	0.1
5–15 minutes before	0.3
30 minutes – 1 hour before	0.5
1 day before	0.7
1 week before	1.0

Table II.III Alert Scores

Priority Level	U_p
High	1.0
Medium	0.5
Low	0.2

Table II.IV Priority Scores

III. Completion Detection

Schedule Completion Detection

Schedule completion is determined by comparing the current time with the schedule's deadline. To account for a grace period, the system will consider a schedule as "overdue" if the current time is greater than one day after the deadline. Hence, the schedule will be dynamically removed from the "Schedule" list and move to "Overdue Schedule" list.

Formula:

$$Overdue = Current\ Time > Schedule\ Deadline + 1\ day$$

This logic ensures that a task is only marked as overdue after a full day has passed beyond the original deadline, reducing false positives from minor delays.

Task Completion Detection

Task completion is determined in the following ways:

1. If the task is explicitly marked as completed by the user.
2. If all associated subtasks are individually marked as completed.

Once task is marked as completed, it will be dynamically removed from the "Task" list and move to "Completed Task" List.

IV. Speech-to-Text Algorithm

The Speech-to-Text (STT) algorithm, also known as Automatic Speech Recognition (ASR), is responsible for converting spoken audio into text. In this project, it is applied to allow users to record comments and issue voice commands to the AI assistant. The system integrates Google Cloud Speech-to-Text, which employs advanced deep learning models trained on multilingual datasets to achieve accurate recognition.

(a) Front-end feature extraction

The first stage of ASR is acoustic feature extraction, which transforms the raw waveform into compact representations that preserve phonetic information. Commonly, the speech signal $x(t)$ is segmented into short frames using a window function before applying the Short-Time Fourier Transform (STFT) [10]. The STFT is expressed as:

$$X(n, w) = \sum_{m} x[m] \cdot w[n - m] \cdot \ e^{-jwm}$$

Where $w[n]$ is a windowing function such as the Hamming window. From the magnitude spectrum, Mel filterbanks and Mel-Frequency Cepstral Coefficients (MFCCs) are derived, which map the spectrum to a perceptual frequency scale approximating human auditory perception [11], [12]. These features reduce dimensionality while retaining phonetic distinctions critical for recognition.

In the application, the Google Cloud Speech-to-Text API transcodes uploaded audio into 16 kHz mono WAV using FFmpeg before extracting log-Mel features, ensuring that multilingual recordings (English, Bahasa Malaysia, and Mandarin Chinese) are standardized for model input.

(b) Acoustic and Language Modeling

Google Speech-to-Text then processes the extracted features through deep neural networks that map acoustic patterns to linguistic units. While the internal implementation uses advanced architectures such as the Recurrent Neural Network Transducer (RNN-T) and Conformer encoders [13], [14], the principle remains the same:

- The acoustic model captures relationships between sound features and phonemes.

- The language model incorporates context to predict which word sequence is most likely.

For example, if the input audio corresponds to *"Submit assignment by Friday"*, the acoustic model recognizes the phonetic sequence, while the language model ensures that the output forms a grammatically correct sentence instead of unrelated words.

(c) Decoding Process

The decoding stage determines the final text output by selecting the word sequence with the highest probability. This is formulated as:

$$\hat{y} = \arg\max P\ (y|x)$$

where x represents the sequence of acoustic features and y represents a possible word sequence. In practice, Google's API employs beam search decoding to evaluate multiple candidate hypotheses in parallel before choosing the most likely transcription [15].

In this project, the speech-to-text process is integrated into two key features:

1. **Voice Comments**: When a user records a comment, the audio file is uploaded to the server and sent to Google Speech-to-Text. The recognized text is then stored in Firestore alongside the audio file reference, making the comment searchable by both voice and text.

2. **AI Assistant Commands**: When the assistant microphone is activated, the audio stream is transcribed in real time. The recognized text is then passed to the AI assistant (Gemini 2.5 Flash) as input, enabling users to issue natural voice commands such as "Plan my day" or "Reschedule meeting to 3 PM."

The recognition system supports multiple languages, and this project enables recognition in English, Bahasa Malaysia, and Mandarin Chinese. For Mandarin input, the transcription output is forced into Han characters instead of Pinyin, ensuring readability and accuracy for native users. Additionally, the system handles background noise by applying voice activity detection (VAD), ensuring that only speech segments are processed [16].

V. Generative AI

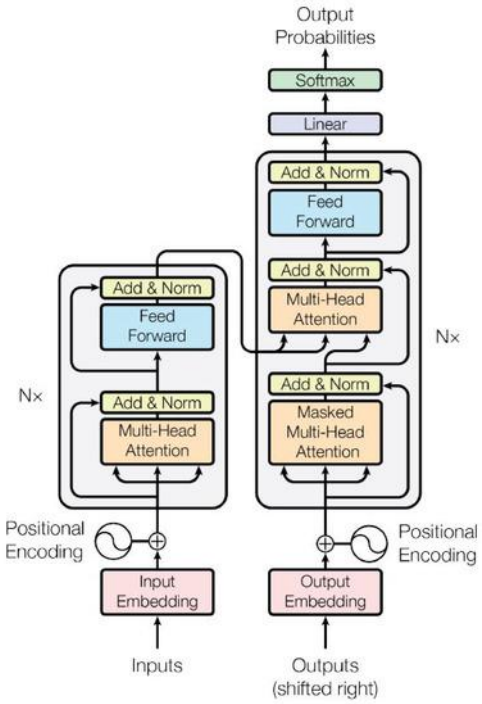

Figure V.I Transformer architecture

Google's Gemini 2.5 Flash is a large-scale generative artificial intelligence model built on the transformer architecture, specifically a decoder-only variant [17]. Transformers are neural network architectures that excel at processing sequential data, making them ideal for language and other multimodal tasks [18]. A key component of the Transformer is the self-attention mechanism, which allows the model to weigh the importance of different parts of the input sequence when processing each element [17]. The model functions as a multimodal reasoning engine capable of processing and generating text, audio, and images. In the context of this project, it is employed as the personal AI assistant that analyses Firestore data (tasks, schedules, and notifications) and returns structured responses in JSON format.

(a) Self-Attention Mechanism: The Core of Transformer

The self-attention mechanism is crucial for the Transformer architecture, allowing the model to weigh the importance of different words in the input sequence when processing each word 17]. It consists of applies self-attention to capture long-range dependencies between input tokens. Given a sequence of tokens $X = (x_1, x_2, \dots, x_n)$, each token is projected into a vector embedding [17]. These embeddings are then processed through multi-head self-attention layers.

The self-attention calculation is defined as:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Where:

- Q = query; K = key; V = value matrices derived from the input embeddings.
- d_k is the dimensionality of the key vectors.
- $softmax$ function normalizes the attention weights across the sequences.

This calculation allows the model to capture long-range dependencies and contextual relationships within the input sequence. For example, when processing the word "it" in a sentence, the self-attention mechanism can determine whether "it" refers to a "cat" or a "house" by looking at the surrounding words and their relationships. In this project, self-attention enables Gemini to reason over Firestore data holistically. For instance, when the user requests *"Plan my day"*, the model simultaneously attends to multiple attributes such as task deadlines, priority levels, and scheduled events. This ensures that urgent tasks (e.g., *"ITPE assignment due tomorrow"*) receive more attention than less critical items (e.g., *"Buy groceries"*). By dynamically focusing on relevant inputs, the assistant highlights the most urgent and contextually appropriate tasks for the user.

(b) Mixture-of-Experts (MoE)

Mixture-of-Experts (MoE) is a technique used to improve the efficiency and capacity of large models like Gemini 2.5 Flash. Instead of activating all model parameters for every input, MoE dynamically selects a subset of specialized "experts" (sub-networks) to process each input [18]. This significantly reduces the computational cost during inference while maintaining or even increasing the model's overall capacity.

Mathematically, for an input vector h:

$$y = \sum_{i=1}^{E} g_i\,(h) f_i(h)$$

Where:

- E: total number of experts
- $f_i(h)$: output of expert i

- $g_i(h)$: gating function that assigns a weight to each expert (often sparse, so only top-k experts are active).

The gating function often uses a $softmax$ or $top-k$ selection mechanism to ensure that only a few experts are active for each input [18]. This sparsity is what makes MoE models computationally efficient. By routing inputs to specialized experts, the model can learn a wider range of patterns and handle more diverse tasks without a proportional increase in computational resources. In practice, this allows Gemini to efficiently allocate resources depending on the user's request.

Example:

- If the user says, *"Summarize my overdue tasks"*, the task reasoning expert is routed.
- If the user says, *"Change meeting time to 3 PM"*, the schedule/time reasoning expert is activated.

This selective activation ensures fast, low-latency responses while maintaining a broad knowledge capacity. For the mobile-based assistant, this efficiency is critical in delivering real-time task suggestions without overloading computational resources.

(c) Attention and Probability Calculation

At each decoding step, Gemini predicts the next token by computing a probability distribution for the next token in JSON format [17]. The probability of generating token t_j given context tokens $t_1, \dots, t_{j-1}$ is:

$$P(t_j | t_1, \dots, t_{j-1}) = softmax(W_o h_j)$$

Where:

- h_j : hidden state output of the decoder for position j.
- W_o: output projection matrix
- $softmax$ function ensures the probabilities sum to 1 across all vocabulary tokens

This ensures that the most likely next token is selected during generation. In this project, this mechanism drives task suggestions and scheduling decisions. For example, given the context:

{
 "todosOpen": [
  { "title": "Study FYP report", "dueDate": "2025 − 09 − 01", "priority": "High" },
  { "title": "Buy groceries", "dueDate": "2025 − 09 − 20", "priority": "Low" }
 ]
}

The model computes higher probability for tokens related to *"Study FYP report"* due to its earlier deadline and higher priority. Consequently, the assistant suggests:

{
 "mode": "suggest",
 "suggestions": [
  { "taskTitle": "Study FYP report", "reason": "Due on 2025 − 09 − 01, high priority." }
 ]
}

This demonstrates how the attention mechanism and probability distribution directly result in prioritized, context-aware task recommendations.

2.1.6 Summary of the Technologies Review

In summary, the technologies selected for this project collectively form a robust and well-integrated platform for developing a mobile-based task management system that supports prioritization, collaboration, and intelligent daily planning. The hardware platform, comprising a mid-range laptop and physical Android device, provides adequate computational power and compatibility for development, emulation, and deployment testing. The use of Windows 10 ensures stable support for the Flutter SDK, Android Studio, and Firebase CLI, enabling smooth development workflows and reliable debugging across both simulated and real environments.

At the data layer, Firebase Firestore was chosen as the primary database due to its real-time synchronization, flexible document-based schema, and offline persistence, which are essential for collaborative task management. Complementing Firestore, Firebase Storage enables the secure storage of multimedia files such as voice recordings and attachments, while Firebase Authentication and Cloud Messaging ensure secure user management and effective delivery of task reminders. Together, these services provide a scalable and cost-efficient cloud backend that supports both structured and unstructured data under the constraints of a controlled billing plan.

The programming environment, built on the Dart language and Flutter framework, allows cross-platform development with a single codebase, reducing both development time and maintenance effort. Flutter's widget-based UI architecture ensures a consistent and responsive interface across devices, while backend processes are supported by Node.js Cloud Functions, which integrate seamlessly with Firebase and Google Cloud services. GitHub further enhances the workflow by serving as the central repository for version control, collaboration, and project tracking.

A set of specialized algorithms ensures that the system is not only functional but also intelligent in its behavior. The Dynamic Reordering Algorithm automatically organizes schedules to prioritize present and upcoming events, improving clarity and reducing cognitive load. The Dynamic Weighted Task Prioritization Algorithm calculates a composite score for each task based on urgency, workload, alerts, and user-defined priority, ensuring that the most critical items are surfaced first. The Completion Detection Algorithm provides automated tracking of overdue and completed tasks, maintaining accuracy in progress monitoring. Beyond these core mechanisms, two advanced service-based algorithms extend the system's functionality. Google

Speech-to-Text processes voice comments and commands into searchable text, enabling multimodal collaboration, while Google Generative AI (Gemini 2.5 Flash) analyses Firestore data to generate context-aware plans, suggestions, and insights, acting as a personal AI assistant to support daily scheduling and task management.

Provider	Model	Key Features	Integration	Suitability for Project
OpenAI	GPT-4o, GPT-3.5	High reasoning ability; supports structured JSON outputs; multimodal (text, image, audio)	Requires separate API, billing, and integration outside Firebase	Strong model, but adds complexity and cost management outside the existing Google Cloud stack
Anthropic	Claude 3 (Opus, Sonnet, Haiku)	Emphasis on safety, alignment, and long-context reasoning; supports structured outputs	External API, no direct Firebase integration	Reliable conversational model, but less suited for Firebase-based task management system
Google	Gemini 2.5 Flash	Built on Transformer architecture with Mixture-of-Experts and sparse attention; optimized for low-latency responses; multimodal capability	Native integration with Cloud Generative AI API and Firebase Functions; schema-constrained JSON output supported	Best fit: provides structured outputs for tasks and schedules, integrates directly with Firestore, and benefits from Google's free usage tier

Table 2.1.6(a): Comparison of Cloud Generative AI APIs and Models [19][20][21][22]

Option	Strengths	Limitation	Suitability for Project
Google Cloud Speech-to-Text (v2)	High accuracy, multilingual (EN, BM, CN), integrates with Firebase	Requires internet; usage is billed after free tier; latency depends on network	Best fit: seamless Firebase integration, reliable multilingual support
Flutter On-Device STT	Works offline, no cloud charges, fast response	Accuracy varies by device/OS, limited language coverage	Useful as fallback but not consistent for collaboration

Open-source models (Whisper / local server)	Free to use; supports many languages; Whisper is robust in noisy conditions	Large model size, heavy CPU/GPU, hard to integrate in Flutter	Impractical for mobile-first Firebase app
Other Cloud APIs (Microsoft Azure, AWS Transcribe)	Competitive accuracy; scalable infrastructure	Separate integration, extra billing setup	Less suited since project already uses Google Cloud

Table 2.1.6(b): Comparison of Speech-to-Text Options [23][24][25][26][27][28]

Overall, the integration of these technologies and algorithms allows the system to extend beyond conventional task management applications. By combining a stable cloud backend with intelligent algorithms and modern development practices, the platform achieves a comprehensive, scalable, and user-centered solution that enhances productivity through dynamic prioritization, seamless collaboration, and proactive planning support. After evaluating multiple cloud generative AI APIs, Google Gemini 2.5 Flash was selected due to its direct Firebase integration, schema-constrained JSON outputs for structured scheduling, and low-latency performance within a cost-controlled free tier. Similarly, after comparing speech-to-text approaches, Google Cloud Speech-to-Text v2 was chosen because it provides accurate multilingual transcription (English, Bahasa Malaysia, Mandarin Chinese). It integrates seamlessly with Firebase Cloud Functions and Firestore and ensures consistency across devices unlike the on-device STT or open-source models which could not guarantee.

2.2 Review on Existing System

2.2.1 Apple Reminder

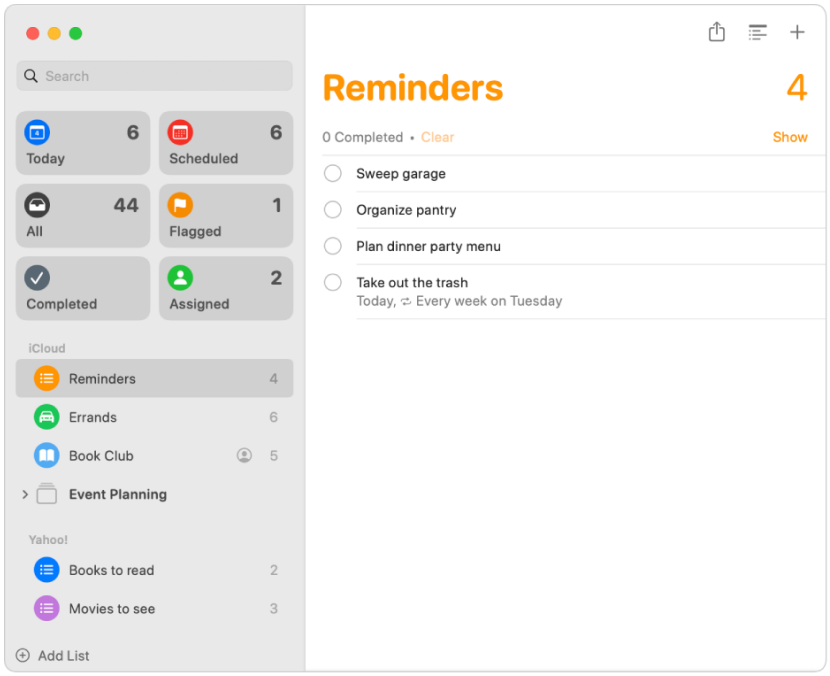

Figure 2.2.1 Apple Reminders

Apple Reminders [2] is a task management tool built specifically for Apple's ecosystem, making it accessible to users of iPhones, iPads, Macs, and other Apple devices. Its core functionalities revolve around task creation, organization, and management, with a key feature being Siri integration. Siri enables voice-based task creation, allowing users to quickly add tasks without needing to type. Another prominent feature is iCloud synchronization, which ensures that all tasks are synced across Apple devices, providing a seamless experience for users who switch between different devices. Subtasks, due dates, and location-based reminders are other helpful features that allow users to break down larger tasks, assign deadlines, and receive reminders when they reach specific locations. The tool also offers collaboration features, allowing users to share lists with others, which is particularly useful for small group projects or household task management. However, its exclusive integration with Apple's ecosystem limits its use for individuals or teams that rely on non-Apple devices.

Strengths

1. Uses iCloud for seamless real-time synchronization across Apple devices with minimal latency.

2. Simple and intuitive interface, making it user friendly.
3. Works with Siri, Shortcuts, Calendar, and Apple Mail natively.
4. Supports shared lists for basic collaboration among Apple users. There are no extra collaboration tools like commenting.

Weaknesses

1. Only available on Apple devices [29].
2. Tasks do not automatically adjust based on deadlines or priorities. [29].
3. Tasks updates are not reflected instantly across collaborators which may cause potential miscommunication [30].
4. No granular task progress tracking [31].
5. Does not automatically reorder tasks based on urgency or workload changes [32].
6. Lacks advanced reporting tools or project visualization tools for project progress tracking [33].

2.2.2 Microsoft To-Do

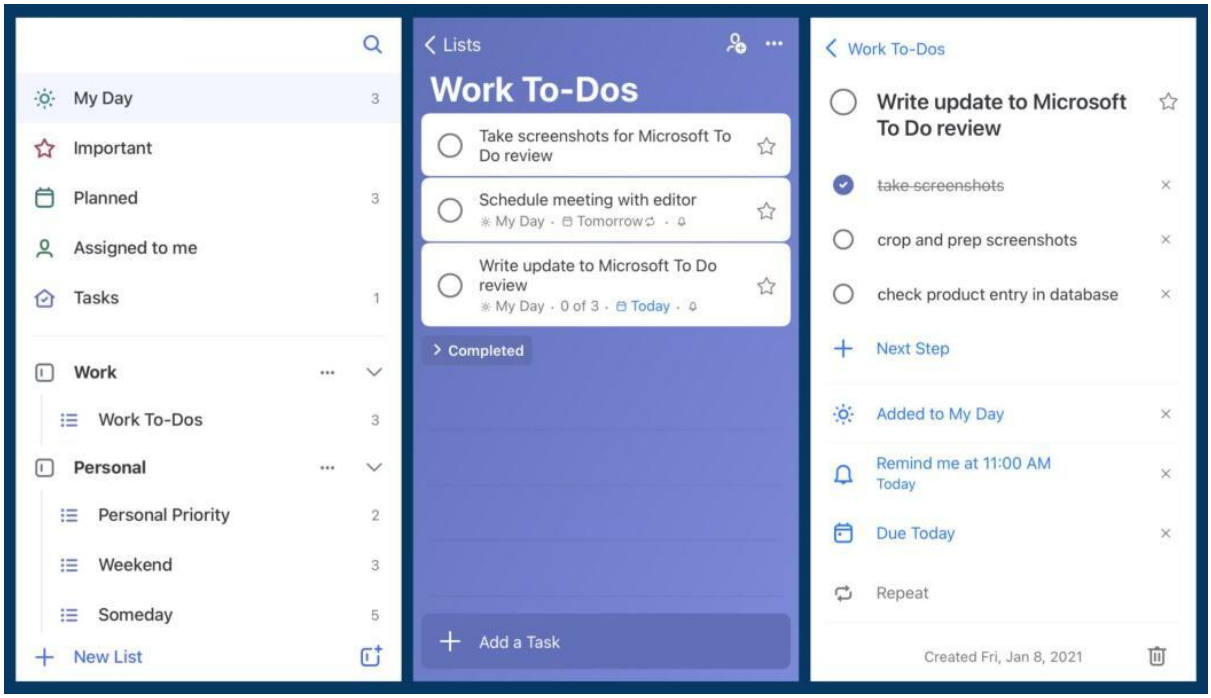

Figure 2.2.2 Microsoft To-Do

Microsoft To-Do [3] is part of the Microsoft 365 suite. It is a versatile task management app that integrates deeply with Outlook. This tool allows users to create and organize tasks into lists, set due dates and reminders, and break down larger tasks into subtasks. File attachments to tasks are supported, making it ideal for users who need to manage both personal and professional projects. With its ability to sync across platforms like Windows, iOS, Android, and the web. It provides flexibility for users working across different devices. One of its strong points is the collaboration feature, allowing users to share lists and assign tasks to team members, thus making teamwork more efficient. Additionally, Microsoft To-Do offers end-to-end encryption, ensuring that all user data is securely stored, which is a significant advantage for users handling sensitive information.

Strengths

1. Integration of Microsoft 365 enables users to sync tasks with Outlook and other Microsoft applications.
2. Simple and clean interface with basic task management features.
3. “My Day” feature helps users focus on daily tasks.

Weaknesses

1. Tasks do not automatically prioritize based on deadlines or urgency [29].
2. Changes made to shared tasks are not updated in real-time, which may cause potential delays [30].
3. Lacks of advanced tracking features for project progress tracking [31].
4. Does not support dynamic task priorities based on the changes of deadlines or dependencies [32].
5. Does not have detailed dashboards or visualization tools for tasks created [33].

2.2.3 Todoist

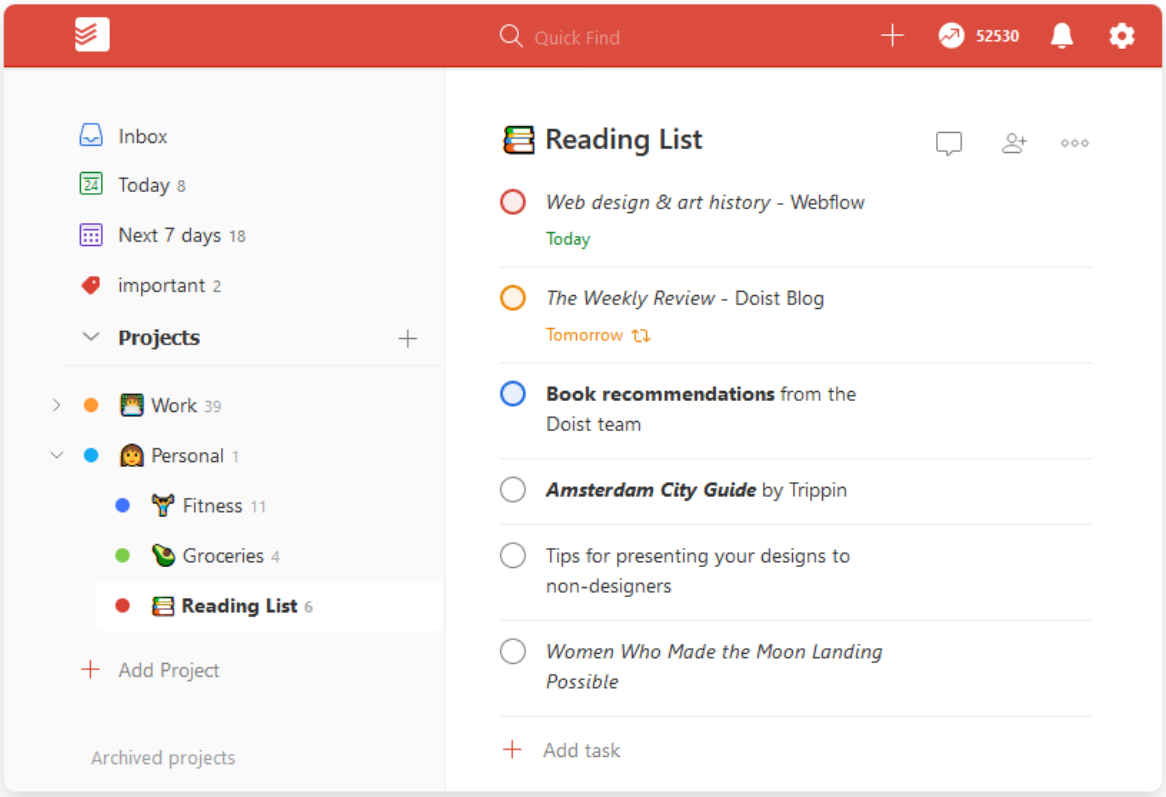

Figure 2.2.3 Todoist

Todoist [4] is widely recognized for its customizability and ability to manage everything from simple tasks to complex projects. One of its standout features is the Kanban-style boards, which offer a visual method for managing tasks, helping users see their workflow at a glance. Todoist supports real-time syncing across devices, ensuring that updates are reflected instantly across all platforms, making it convenient for users on the go. The platform offers subtasks, labels, filters, and priority levels, enabling users to organize their tasks in a highly personalized way. Additionally, Todoist shines in its integration with third-party apps, such as Google Calendar, Slack, and Trello, enhancing its capabilities for automation and workflow management. The platform's collaboration features allow for task delegation, which is helpful for team projects where responsibilities need to be assigned.

Strengths

1. Excellent visual task management through boards, lists and cards.
2. Supports smart task creation with natural language processing.
3. Supports integration with external tools like Slack (a productivity tools).

Weaknesses

1. Collaborated task updates are not reflected instantly which may cause confusion in team-based work [30].
2. Does not offer comprehensive dashboards or analytics for tracking project progress [33].
3. Does not dynamically reorder or prioritize tasks based on evolving schedules and dependencies [32].
4. It can feel overwhelming for beginners due to complex functionalities.

2.2.4 TickTick

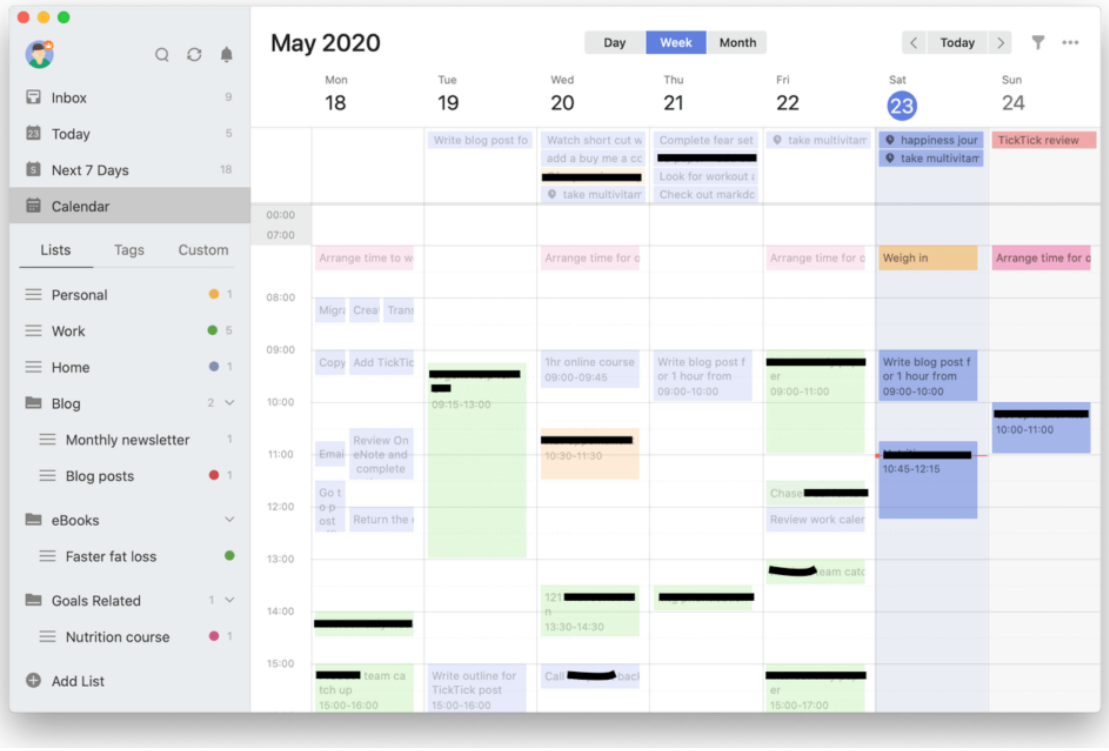

Figure 2.2.4 TickTick

TickTick [8] is a versatile task management app designed to enhance productivity through a wide array of features. It allows users to create and organize tasks into lists and projects, with options to set due dates, priorities, and subtasks. The app integrates seamlessly with Google Calendar, offering synchronized views of tasks and deadlines, and supports Kanban-style boards for visual task management. Additional features include a Pomodoro timer for focused work intervals, habit tracking to monitor personal goals, and cross-platform support across iOS, Android, Windows, and macOS. While TickTick provides robust functionalities, limitations include restricted advanced features in the free version and occasional performance issues, which may be challenging for new users to navigate.

Strengths

1. Supports task categorization and priority levels for structured task management.
2. Offers natural language input for quick task creation.
3. Available across multiple platforms including web, mobile, and desktop.

Weaknesses

1. No detailed status updates beyond completed or incomplete tasks [31].
2. Lacks real-time prioritization based on changing workloads or deadlines [32].
3. Lacks sub-task dependencies.

2.2.5 Trello

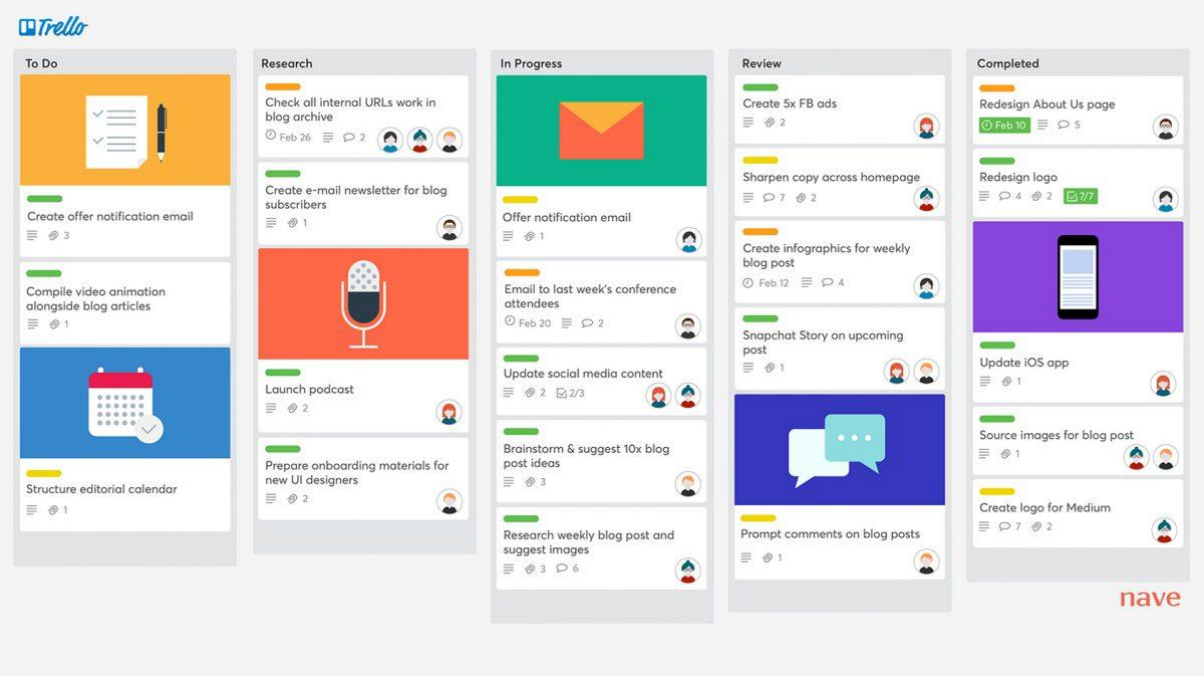

Figure2.2.5 Trello

Trello [9] is a simpler but highly flexible task management tool that relies on Kanban boards to organize tasks visually. In Trello, each project is represented as a board, and tasks are organized into cards, which can be moved between columns such as "To Do," "In Progress," and "Done" as they progress. Trello's drag-and-drop interface makes it intuitive and easy to use, especially for individuals and small teams working on less complex projects. It also supports file attachments, comments, and task sharing, enabling basic collaboration features. Third-party integrations with apps like Google Drive, Slack, and Dropbox enhance Trello's functionality, providing additional tools for users to manage documents and communication within their projects. While Trello's simplicity makes it accessible, it can also limit its effectiveness for managing more intricate projects that require task dependencies or advanced reporting features.

Strengths

1. Offers a built-in Pomodoro timer to enhance focus and productivity.
2. Provides habit tracking features
3. Cross-platform availability with calendar integration.
4. Supports task categorization and smart lists for better organization.

Weaknesses

1. Does not dynamically prioritize task based on urgency or deadlines [32].
2. Does not allow user to create sub-task dependencies.

2.3 Limitations of Previous Studies

2.3.1 Platform Limitation

Apple Reminders is tightly integrated with Apple devices, making it incompatible with non-Apple platforms like Android or Windows. This exclusivity can be a major drawback for users who rely on devices outside of the Apple ecosystem. For instance, if a team consists of both Apple and non-Apple users, collaboration is limited since the app cannot be used across all devices. This reduces flexibility for teams or individuals who operate in mixed device environments [31].

2.3.2 Absence of Dynamic Task Reordering

Tasks in **Apple Reminders**, and **Microsoft To-Do** are static in terms of order. They are not automatically adjusted based on factors such as upcoming deadlines, task urgency, or priority changes. This limitation can make it harder to manage overlapping tasks effectively, as users need to manually reorder or update tasks instead of relying on automatic adjustments. Discussions in the Microsoft Community highlight user experiences with the inability to manually sort tasks in the "My Day" list, reflecting this limitation [32][33].

2.3.3 Lack of Real-Time Sync for Collaboration

Apple Reminders, **Microsoft To-Do**, and **Trello** does not offer real-time synchronization for task collaboration. This means when multiple team members are working together, task updates and changes do not appear instantly for everyone. As a result, team members may not always have the most current version of a task list or project status, leading to confusion or miscommunication. Teams need to use external communication tools (like email or messaging apps) to stay aligned, adding extra steps and potential friction to the collaborative process [30] [31].

2.3.4 Limited Integrations

Although **Microsoft To-Do** integrates well with the Microsoft 365 suite, it lacks support for other popular platforms like Google Calendar or Slack. This limits flexibility for users who rely on non-Microsoft tools for project management, making it more difficult to create a seamless workflow across various software ecosystems. Users with diverse toolsets may find

Microsoft To-Do limiting in terms of app integrations, reducing its overall utility for managing both personal and professional tasks across multiple platforms [34].

2.3.5 No Granular Task Progress Tracking

Apple Reminders, **Microsoft To-Do**, and **Todoist** lacks advanced features like task progress tracking, such as completion percentages or detailed task statuses (e.g., "in progress," "review pending"). This makes it difficult for users to gauge how far along they are on a specific task, forcing them to rely on vague markers like task completion (done or not done). This lack of granular tracking is a disadvantage when managing larger projects where task progress visibility is crucial for ensuring work is on schedule [35].

2.3.6 Limited Intelligent Task Scheduling

Apple Reminders lacks dynamic task prioritization, relying on static priority levels that require manual adjustments. **Microsoft To-Do** offers basic task organization but does not automatically reorder tasks based on urgency or workload changes. **Tick Tick** and **Todoist** provide categorization and sorting options, but they lack real-time prioritization that adapts to shifting deadlines and dependencies. **Trello** excels in visual task management but does not dynamically adjust task priorities based on evolving schedules. Overall, these apps offer useful features but do not fully address the need for dynamic task prioritization, which ensures tasks are continuously ranked based on urgency, dependencies, and workload [29][34][36].

2.3.7 Limited Reporting and Visualization

Apple Reminders, **Microsoft To-Do**, and **Trello** does not provide advanced reporting tools or project visualization options like project dashboards or detailed progress reports. For users who need insight into task timelines, bottlenecks, or overall project status, Trello's limited visualization capabilities can make it harder to monitor the flow of work and make data-driven decisions [30].

2.4 Proposed Solutions

To tackle the challenges of task management and boost project efficiency, we're proposing a robust new system designed with several key features.

Our first major improvement is **dynamic task prioritization**. We'll implement a smart algorithm that automatically adjusts task priorities based on deadlines, dependencies, and workloads. This means that as deadlines approach or tasks become more interconnected, the system will dynamically update task priorities and reorder them to ensure students focus on the most urgent and important tasks. Additionally, we'll introduce **detailed task statuses**, such as "In Progress," "Awaiting Review," and "Completed," complemented by visual elements like progress bars and charts. This will provide students with a clearer view of each task's status and overall project progress, making it easier to manage overlapping deadlines and identify where attention is needed.

We understand that internet access isn't always reliable, so our system will also include **offline capabilities**. This feature ensures that students can still access and update their tasks without needing a constant internet connection, which is especially useful in situations where connectivity is intermittent.

When it comes to collaboration, our system will enhance communication through **real-time commenting** directly within tasks. This will allow students and team members to exchange feedback, ask questions, and discuss tasks in the context where they're needed, reducing reliance on external communication tools and keeping all discussions linked to specific tasks. We'll also **integrate document management** by allowing users to link shared documents directly within the workspace. This integration will streamline access to project materials and ensure that everyone has the latest versions of documents, avoiding confusion and fragmentation.

To make the system more versatile, we'll ensure **cross-platform compatibility**, integrating with popular tools like Google Calendar and user device Calendar. This will accommodate users who rely on different platforms and enhance flexibility.

For tracking and visualizing projects, we'll provide **granular task progress tracking**, showing detailed completion levels (e.g., 50% or 80% complete). This will offer students a precise view of their progress. A **unified dashboard** will aggregate information from all ongoing projects,

including deadlines, task statuses, and overall progress, helping students manage their workload and identify tasks that need immediate attention.

By incorporating these features, we aim to create a user-friendly platform that improves task management, streamlines collaboration, and enhances project tracking, effectively addressing the limitations of existing systems and meeting the needs of students and teams.

Features	Apple Reminders	Microsoft To-Do	Todoist	TickTick	Trello	Proposed System
Dynamic Task Prioritization	✗	✗	✗	✗	✗	✓
Collaboration	✓	✓	✓	✓	✓	✓
Real-Time Commenting	✗	✗	✓	✗	✓	✓
Document Management	✗	✓	✓	✓	✓	✓
Granular Task Status	✗	✗	✓	✗	✗	✓
Progress Visualization	✗	✗	✓	✓	✓	✓
Dynamically adjusting to new tasks and deadlines	✗	✗	✗	✗	✗	✓
Calander Integration	✗	✗	✓	✓	✗	✓
Task Completion Detection	✗	✗	✗	✗	✗	✓
Personalized Assistant	✗	✗	✗	✗	✗	✓

Table 2.5 Comparison between existing and proposed applications

CHAPTER 3

Proposed Method/Approach

3.1 System Design Diagram

3.1.1 Block Diagram

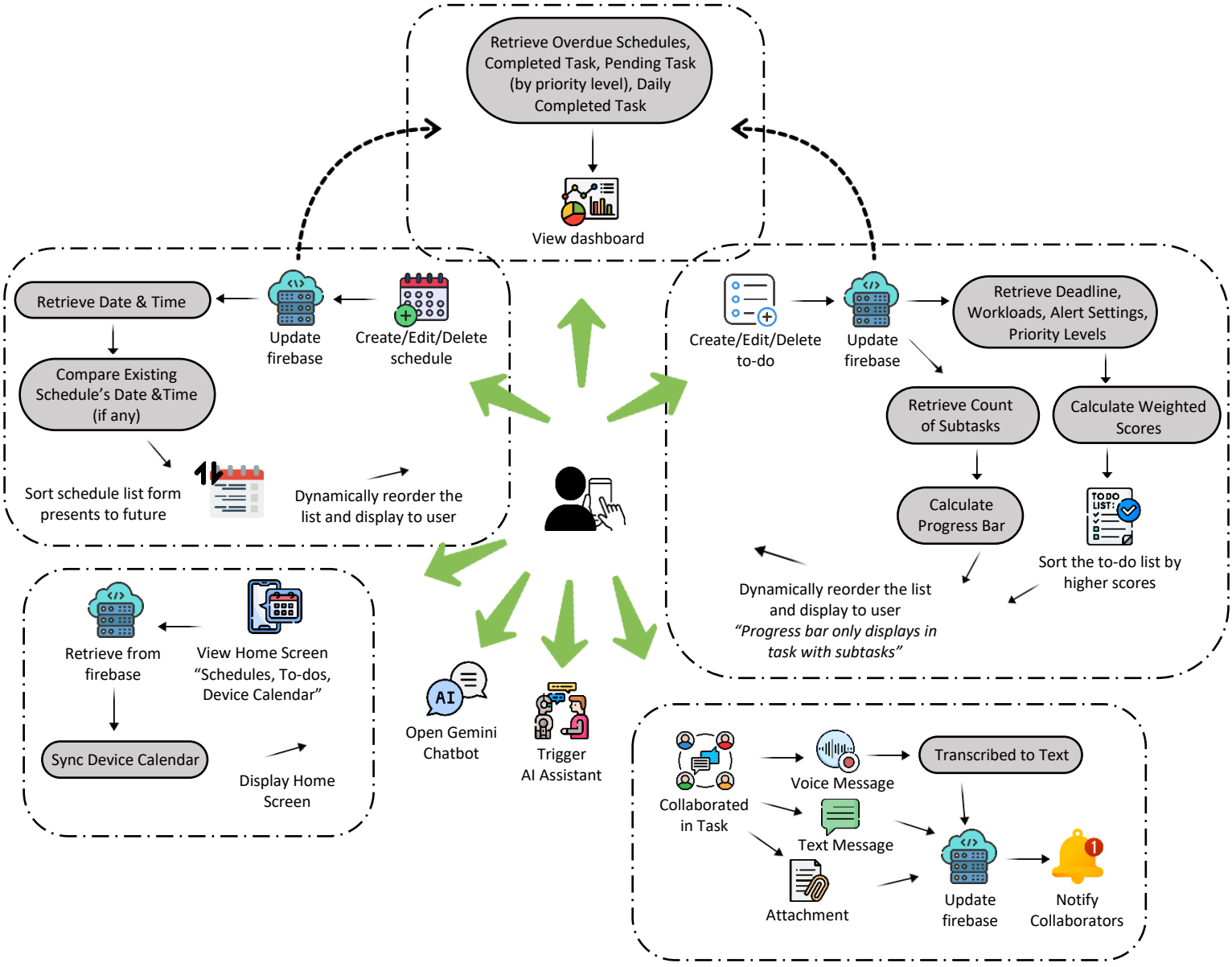

Figure 3.1.1 Block Diagram

3.1.2 Entity Relationship Diagram

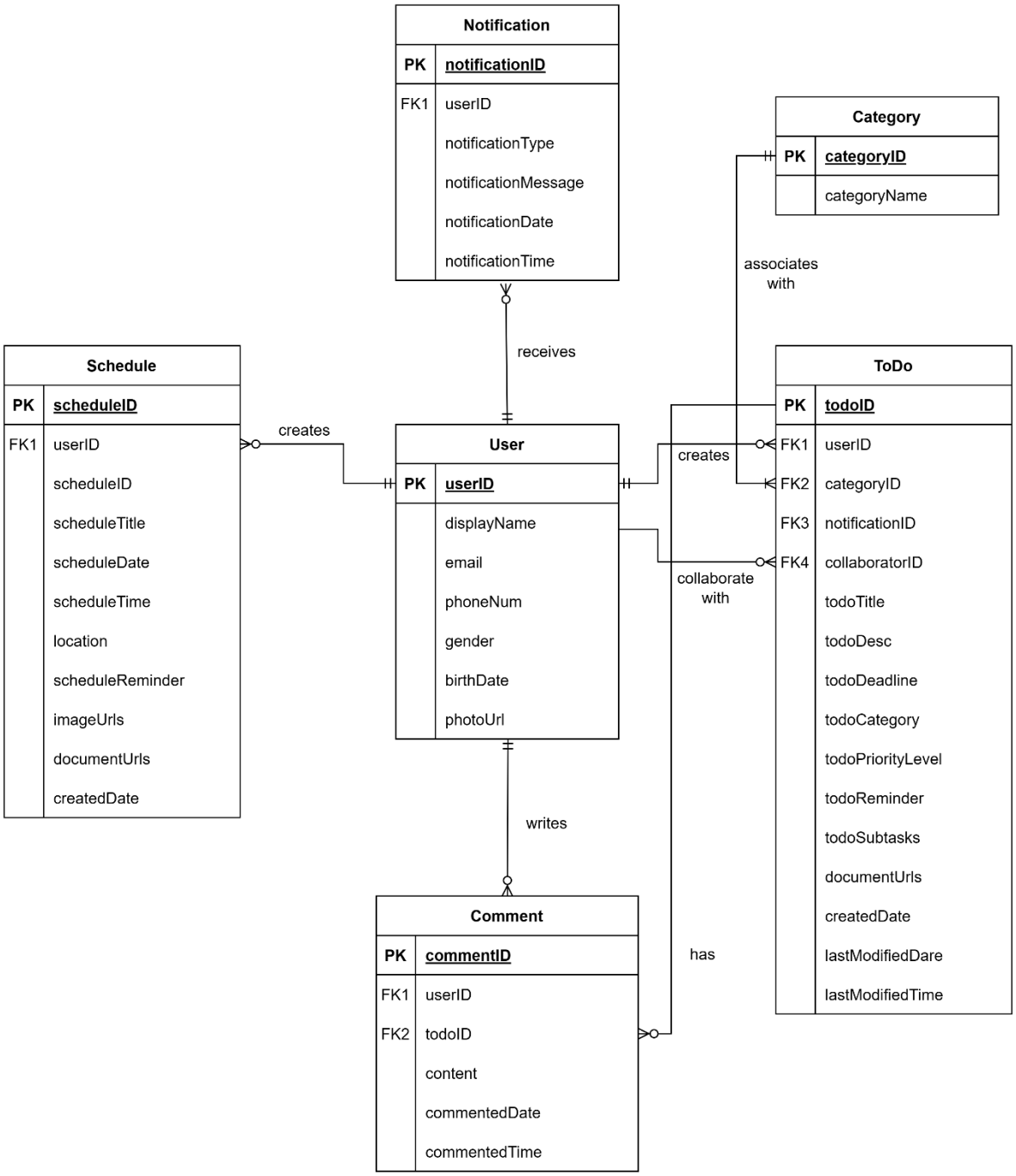

Figure 3.1.2 Entity Relationship Diagram

3.1.3 Class Diagram

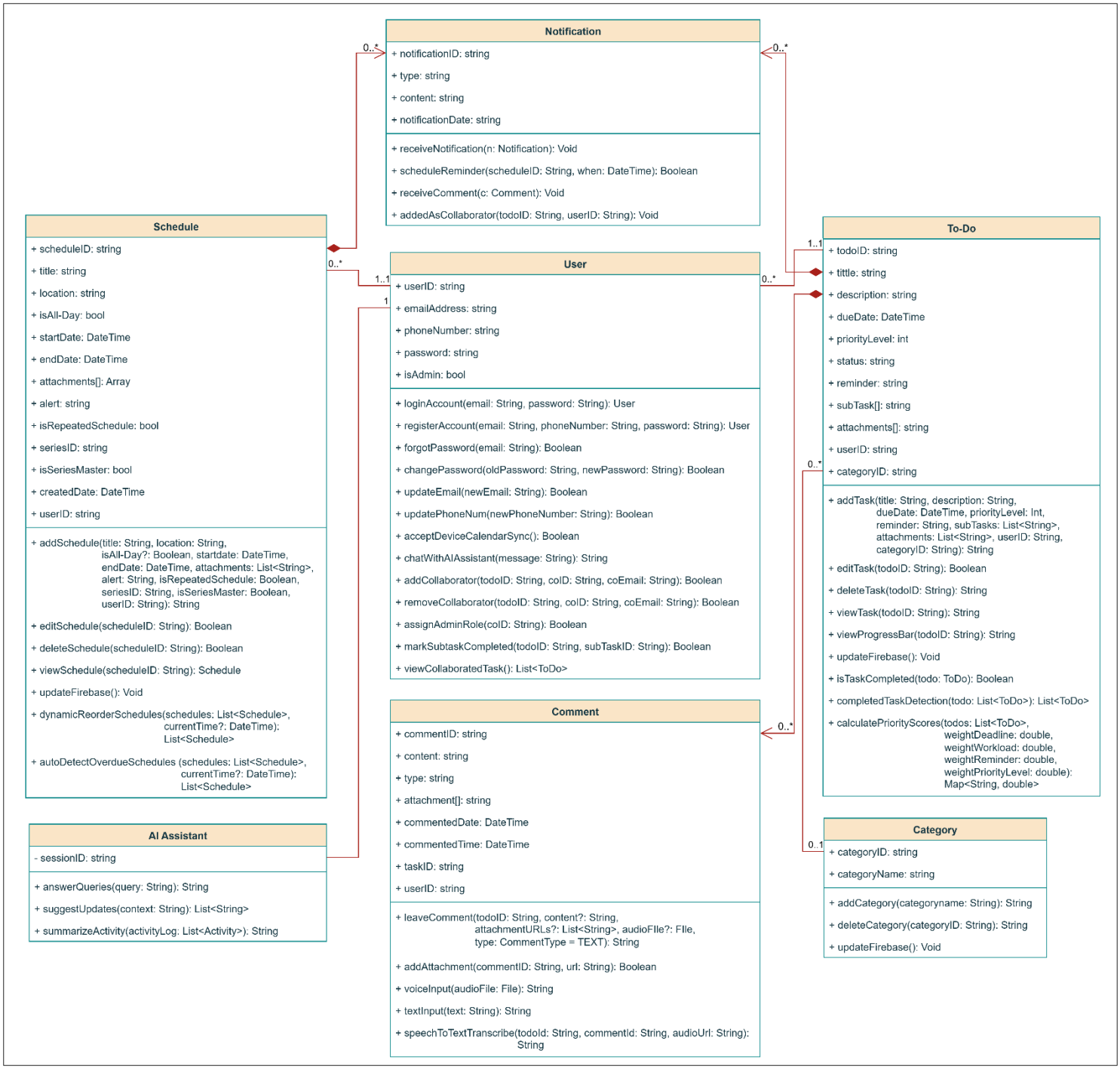

Figure 3.1.3 Class Diagram

3.1.4 Use Case Diagram

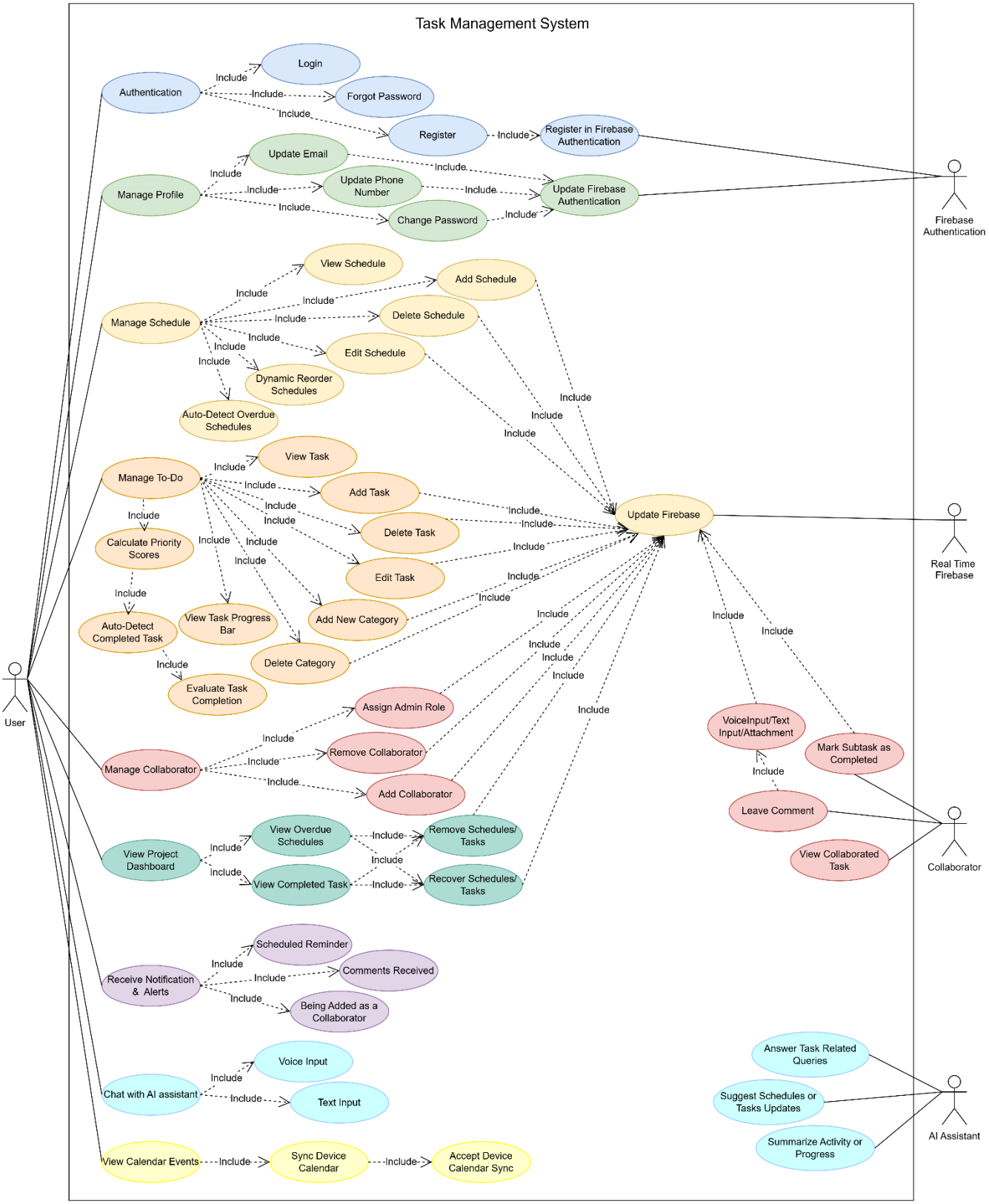

Figure 3.1.4 Use Case Diagram

3.1.5 Dashboard

Activity Diagram

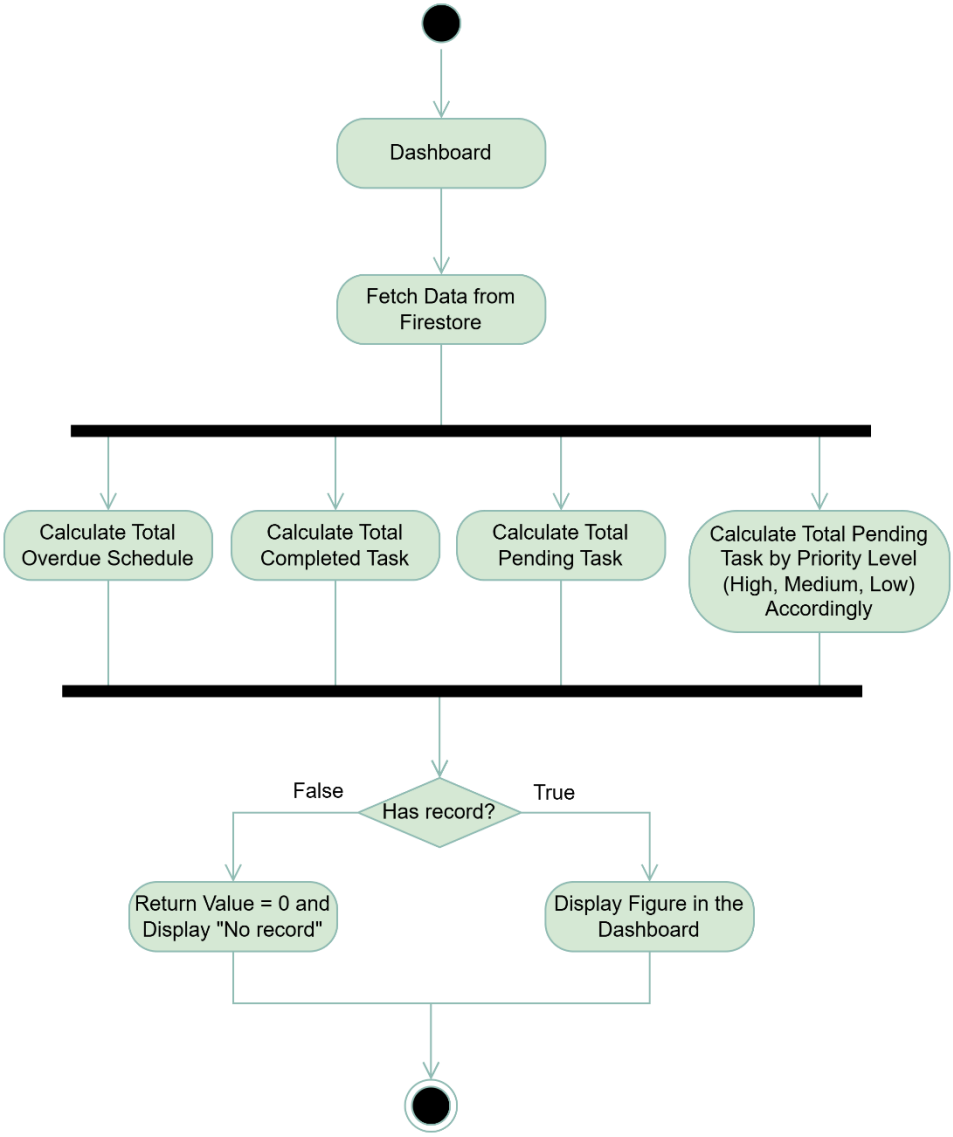

Figure 3.1.5 Activity Diagram – Dashboard

Use Case Description

Use Case	Display Dashboard Metrics
Actor	User, Firestore (data source)
Pre-conditions	1. User is authenticated and on the Dashboard screen. 2. Firestore is available and contains schedule/task data.
Main Flow	1. The user navigates to the Dashboard. 2. The system fetches data from Firestore. 3. The system performs the following calculations in parallel: - Calculates the total overdue schedules. - Calculates the total completed tasks. - Calculates the total pending tasks. - Calculates the total pending tasks by priority level (High, Medium, Low). 4. The system checks if there are records available from the calculations. 5. If records are available, the system displays the calculated figures (total overdue schedules, total completed tasks, total pending tasks, and pending tasks by priority) on the dashboard.
Alternative Flow	In step 4, if no records are available, the system returns a value of 0 for all calculations and displays a "No record" message on the dashboard.
Exception Flow	If the system fails to fetch data from Firestore in step 2, it logs an error and displays an error message on the dashboard, such as "Failed to load data."
Post-conditions	1. The dashboard displays the calculated figures (total overdue schedules, total completed tasks, total pending tasks, and pending tasks by priority) if records are available. 2. If no records are found, the dashboard shows a "No record" message with all values set to 0.

Table 3.1.5 Use Case Description – Dashboard

3.1.6 Progress Bar

Activity Diagram

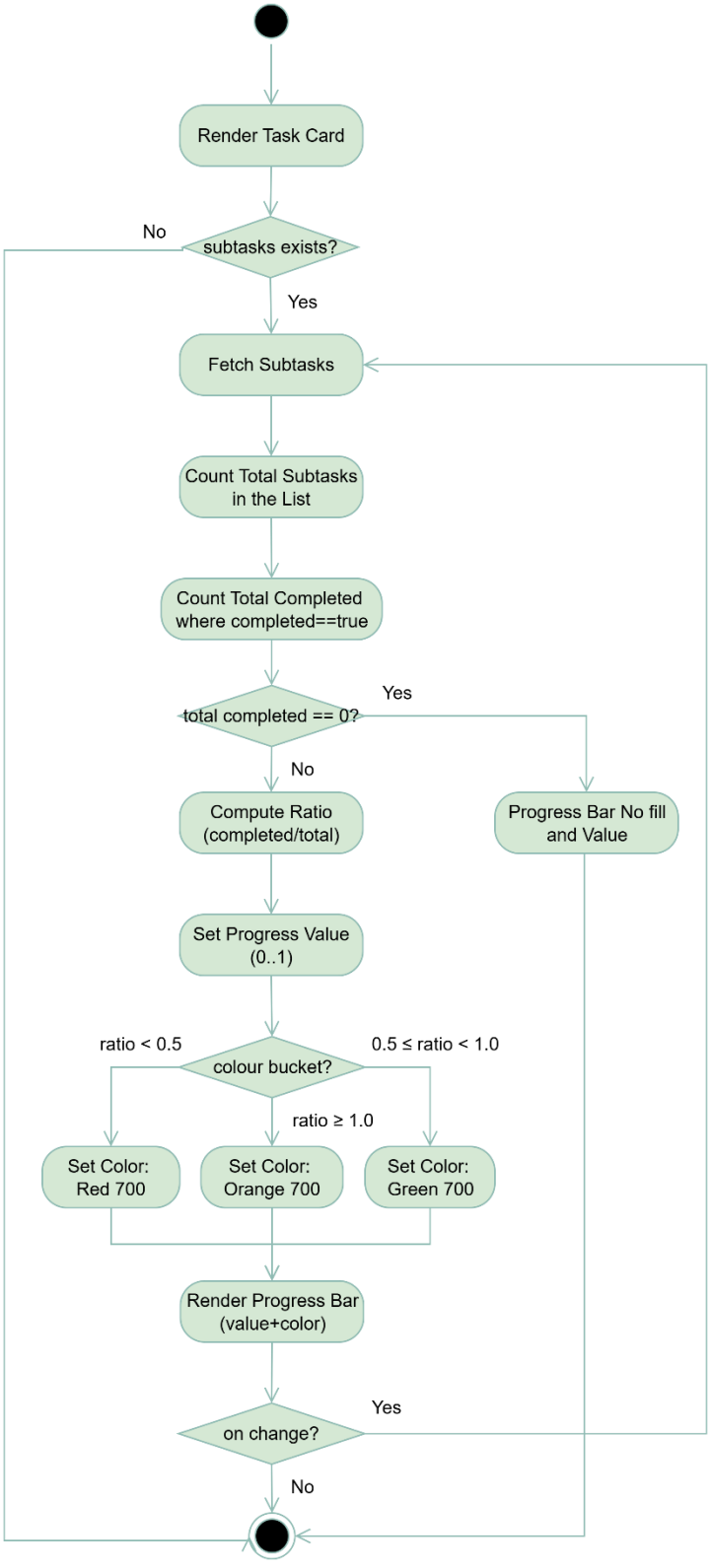

Figure 3.1.6 Activity Diagram – Progress Bar

Use Case Description

Use Case	Show Task Progress Bar (when subtasks exist)
Actor	User, System, Firestore
Pre-conditions	1. The user is logged into the system. 2. User is authenticated and can view the task. 3. Task document is readable; if subtasks exist, they are retrievable as a list.
Main Flow	1. Render Task Card. 2. Check "Subtasks exist?" - If **no** → do not render a progress bar (end). - If **yes** → continue. 3. **Fetch Subtasks** (from task doc or subcollection). 4. **Count Total Subtasks** = subtasks.length. 5. **Count Completed** = number of items where: completed == true. 6. Compute Ratio: - If total == 0 → treat as 0.0 (and show empty bar). - Else ratio = completed / total (clamped to 0..1). 7. **Set Progress Value** = ratio (0..1). 8. Choose Colour Bucket: - ratio >= 1.0 → **Green 700** - 0.5 <= ratio < 1.0 → **Orange 700** - ratio < 0.5 → **Red 700** 9. Render Progress Bar (value + color). 10. On Change: - Subscribe to subtask changes (snapshot/stream). - On any add/update/delete or completed toggle → **recompute** from step 3 and re-render.
Alternative Flow	**AF-1: Zero Completed (but subtasks exist)** • Show a progress bar with value 0% and Red 700. **AF-2: All Completed** • Show a full bar (100%) with Green 700. **AF-3: Intermediate Completion** • Show partial bar with Orange 700 for ratios in [0.5, 1.0). **AF-4: Smooth Visual Update (optional)**

	• Animate value/colour transitions (~200 ms) when recomputing to avoid junk.
Exception Flow	**EF-1: Data Read Failure** • If Firestore read fails, hide the bar and retry indicator (do not block the card). **EF-2: Malformed Subtask Items** • If any list item is not a map or lacks completed, treat it as **not completed** (no crash). **EF-3: Division Safety** • Guard against total == 0 to avoid divide-by-zero (render empty or hide per rule above).
Post-conditions	1. If no subtasks exist, the task card shows **no** progress bar. 2. If subtasks exist, the task card shows a progress bar whose **value** and **colour** reflect completed/total subtasks and update in real time when subtasks change.

Table 3.1.6 Activity Diagram – Progress Bar

3.1.7 Personalized AI Assistant

Activity Diagram

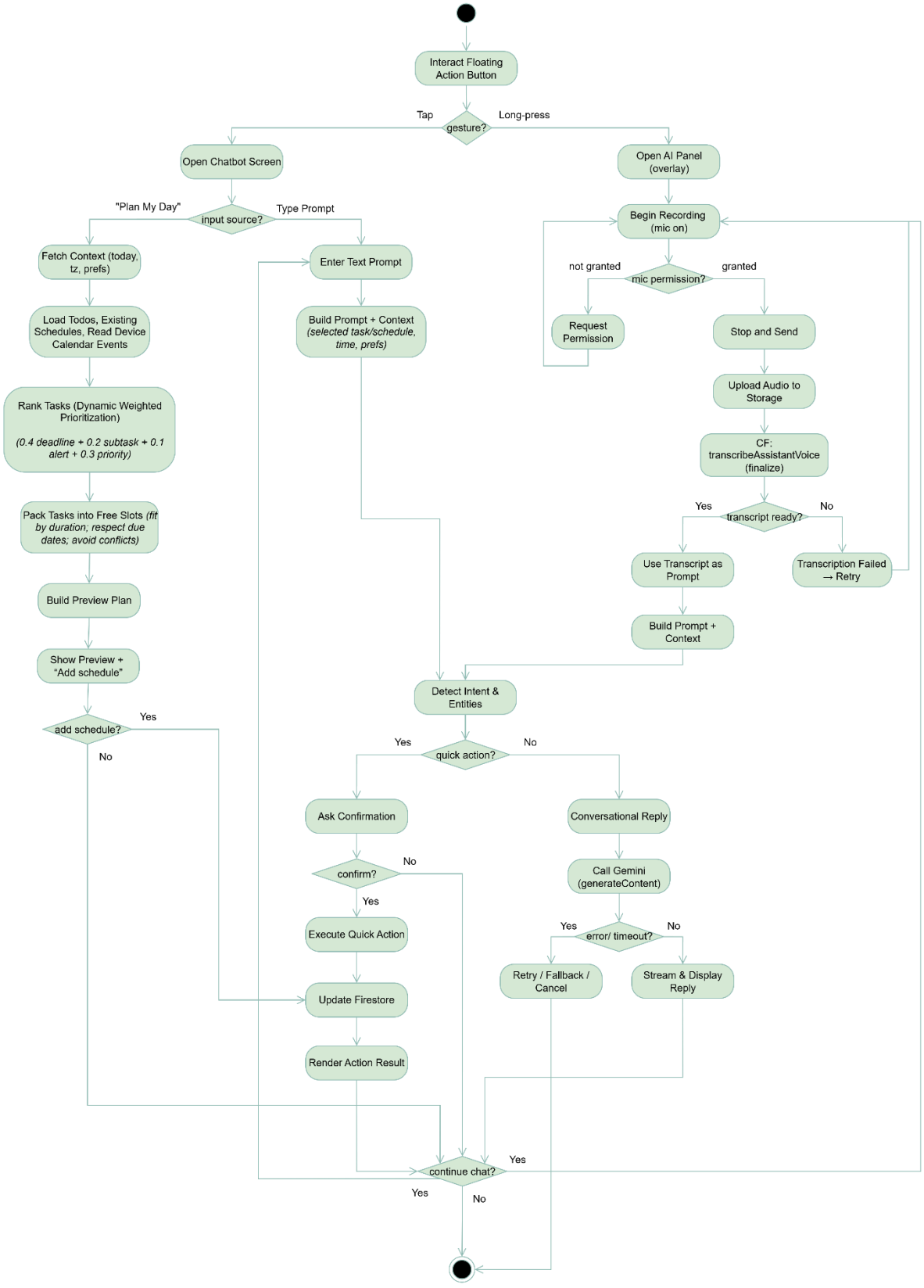

Figure 3.1.7 Activity Diagram – Personalized AI Assistant

Use Case Description

Use Case	AI Assistant for Personalized User Support (Chatbot & Voice Panel)
Actor	User, AI Assistant System, Cloud Functions, Firestore/Storage, Google Speech-to-Text, Gemini LLM, Device Calendar
Pre-conditions	1. User is authenticated and online. 2. AI services are reachable (Cloud Functions, Gemini). 3. Microphone permission granted for voice panel 4. Calendar permission granted for "Plan my Day". 5. The system can read user data (todos, schedules) from Firestore; device calendar events available if permitted.
Main Flow	**A) Entry via Floating Action Button (FAB)** 1. User taps or long-presses the FAB. 2. System branches by gesture: 3. Tap → Chatbot screen 4. Long-press → AI voice panel (overlay) **B) Chatbot (tap)** 5. System asks for **input source**: **Plan my Day** (preset) or **Type Prompt**. **B1) Plan my Day (preset)** 6. System fetches context (today/zone/prefs), then in parallel loads **todos**, **existing schedules**, and **device calendar events**. 7. System **ranks tasks** using Dynamic Weighted Prioritization (0.4 deadline + 0.2 subtask + 0.1 alert + 0.3 priority). 8. System **packs tasks into free slots** (respect due dates, avoid conflicts) and builds a **preview plan**. 9. System shows **Preview + "Add schedule" / Cancel**. 10. If user confirms **Add schedule**: 11. System executes the action (Cloud Function), **writes schedules** to Firestore, and (if applicable) **notifies** collaborators. 12. System **renders** the updated schedule and posts a summary reply in chat. 13. Flow returns to chat (user may continue). 14. If user cancels, return to chat without changes. **B2) Type Prompt (free text)** 15. User enters text.

	16. System builds prompt + context (selected task/schedule, time, preferences) and detects intent & entities. 17. **[Decision] Quick action?** 18. **Yes:** System asks for confirmation ("Proceed?"). If confirmed, execute via Cloud Function, update Firestore, render the action result, and return to chat. 19. **No:** System calls Gemini for a conversational reply; on success streams & displays the reply. **C) Voice Panel (long-press)** 20. System opens overlay and begins recording. 21. **[Decision] Mic permission?** If needed, request; if denied, exit panel. 22. On **Stop & Send**, system uploads audio to Storage; a Cloud Function is triggered to transcribe (STT). 23. **[Decision] Transcript ready?** 24. **Yes:** Use transcript as the prompt; continue as in steps 11–12 (build context, detect intent, quick action vs conversational). 25. **No / Failed:** Show Retry transcription or let the user type instead. 26. System displays the reply in the panel (optionally with TTS). User can continue (record again) or dismiss the panel.
Alternative Flow	**AF-1: Continue Chat** • After any reply, the user can send another prompt or record again, loop to the appropriate input step. **AF-2: No free slots in Plan my Day** • System proposes alternatives (different time window, shorter tasks) and lets the user adjust or cancel. **AF-3: Quick action variants** • Create/update tasks, schedules, notes, or other supported operations; all require **user confirmation** before write.
Exception Flow	**EF-1: Permission denied** • Microphone or calendar permission not granted → show guidance to enable in settings. **EF-2: Network or service error** • Gemini/Cloud Function timeout or 503 → offer **Retry / Fallback / Cancel**. **EF-3: Transcription failure**

	• STT error → show inline error with **Retry**; voice message can still be typed manually. **EF-4: Insufficient context** • System asks clarifying questions or suggests creating the needed data (e.g., add task durations) before proceeding.
Post-conditions	1. The user receives a response (conversational or action result). 2. If a quick action or "Add schedule" is confirmed, Firestore is updated, and the UI reflects the change. 3. Conversation can continue in a multi-turn loop until the user exits.

Table 3.1.7 Use Case Description – Personalized AI Assistant

3.1.8 Task Collaboration

Activity Diagram

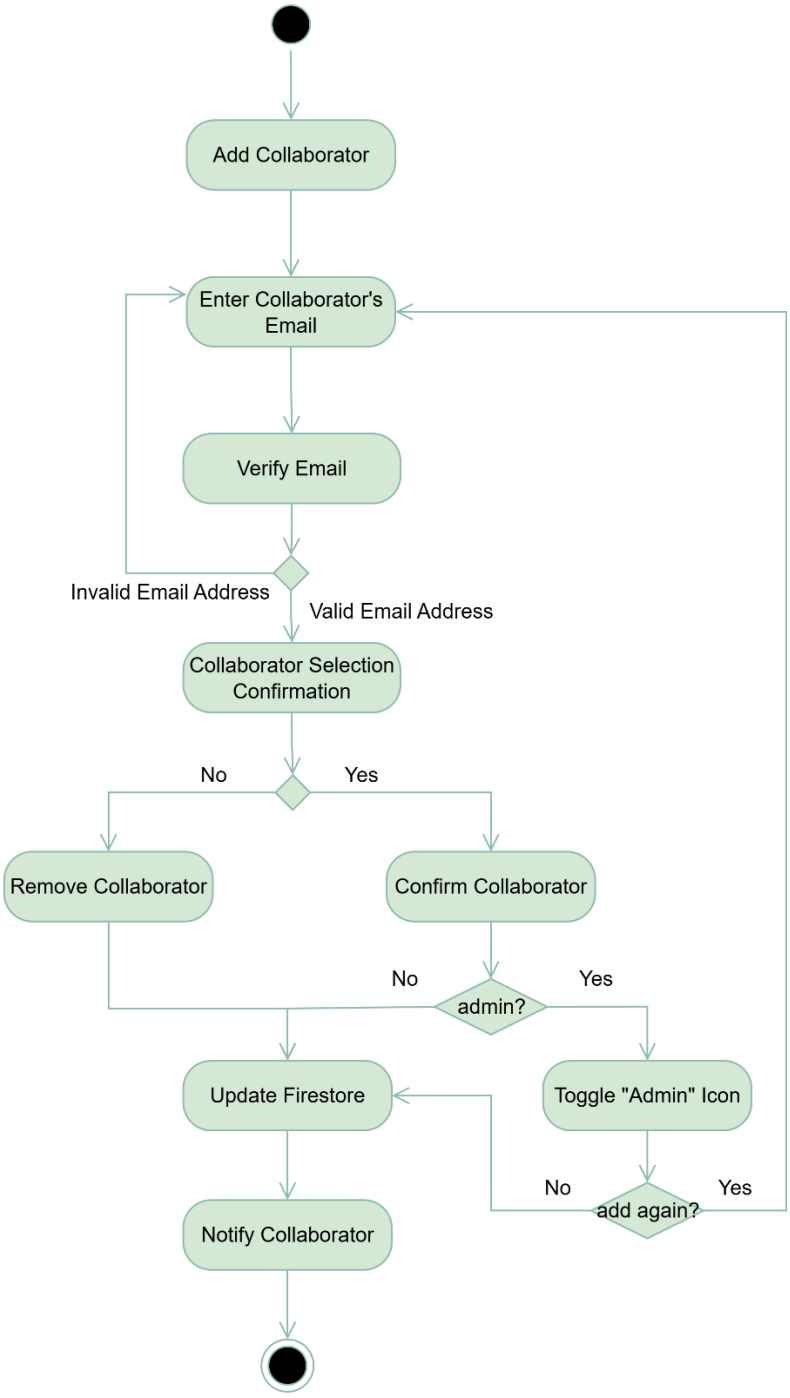

Figure 3.1.8 Activity Diagram – Task Collaboration

Use Case Description

Use Case	Add Collaborator
Actor	User, Collaborator, System
Pre-conditions	1. The user is logged into the system. 2. User has permission to manage collaborators on the task (owner/admin). 3. Task exists and is selectable.
Main Flow	1. User selects Add Collaborator. 2. System prompts the user to enter the collaborator's email. 3. System **verifies email**: format check, user existence, and not already a collaborator. 4. If invalid/not found/already added, system shows an inline error and remains on the email entry; on success continue. 5. System shows **Collaborator Selection Confirmation** (summary card with email and an **Admin** toggle/icon). 6. User chooses one of the following on the confirmation card: - **Confirm Collaborator** (default role = member). - **Toggle "Admin"** to grant/revoke admin before confirming. - **Remove Collaborator**. 7. System **updates Firestore** accordingly (single commit/transaction): - **On confirm:** create/merge tasks/{taskId}/collaborators/{uid} with fields like {role: "member"|"admin", addedBy, addedAt}. - **If admin toggled:** set role: "admin" (or "member" if toggled off). - **On remove:** ensure no write occurs (or delete any pending local draft state). 8. System **notifies collaborator** (and, if needed, existing admins) and refreshes the UI list. 9. Flow can **repeat** from step 2 to add more collaborators.
Alternative Flow	**AF-1: Assign Admin During Add** 1. From the confirmation card, user enables the admin icon/toggle and confirms. 2. System writes the collaborator with role: "admin", then proceeds to notify and refresh. **AF-2: Remove Before Confirm**

	1. User toggles remove icon on the confirmation card. 2. System discards the pending addition and returns to the collaborator list (no Firestore change). **AF-3: Quick Role Flip After Confirm (optional UI)** 1. From the list row, user taps the admin icon to grant/revoke admin on an already-added collaborator. 2. System updates role in Firestore and notifies the collaborator of the role change.
Exception Flow	**EF-1: Email Errors** • Invalid format, user not found, or already a collaborator → show specific error and keep focus on the email field. **EF-2: Firestore Update Failure** • Network/permission/write failure → show "Couldn't update collaborators. Retry?" with Retry/Cancel. On Retry, re-attempt the write; on Cancel, show current list. **EF-3: Notification Failure** • If push/send fails, logs; addition/role change remains saved.
Post-conditions	1. The collaborator is either successfully added and notified, assigned an admin role, or removed. 2. The affected collaborator receives a notification about being added and/or role assignment; existing admins may also be notified on removals/role changes. 3. UI shows the updated collaborator list (with admin action state).

Table 3.1.8 Use Case Description – Task Collaboration

3.1.9 Manage Comment

Activity Diagram

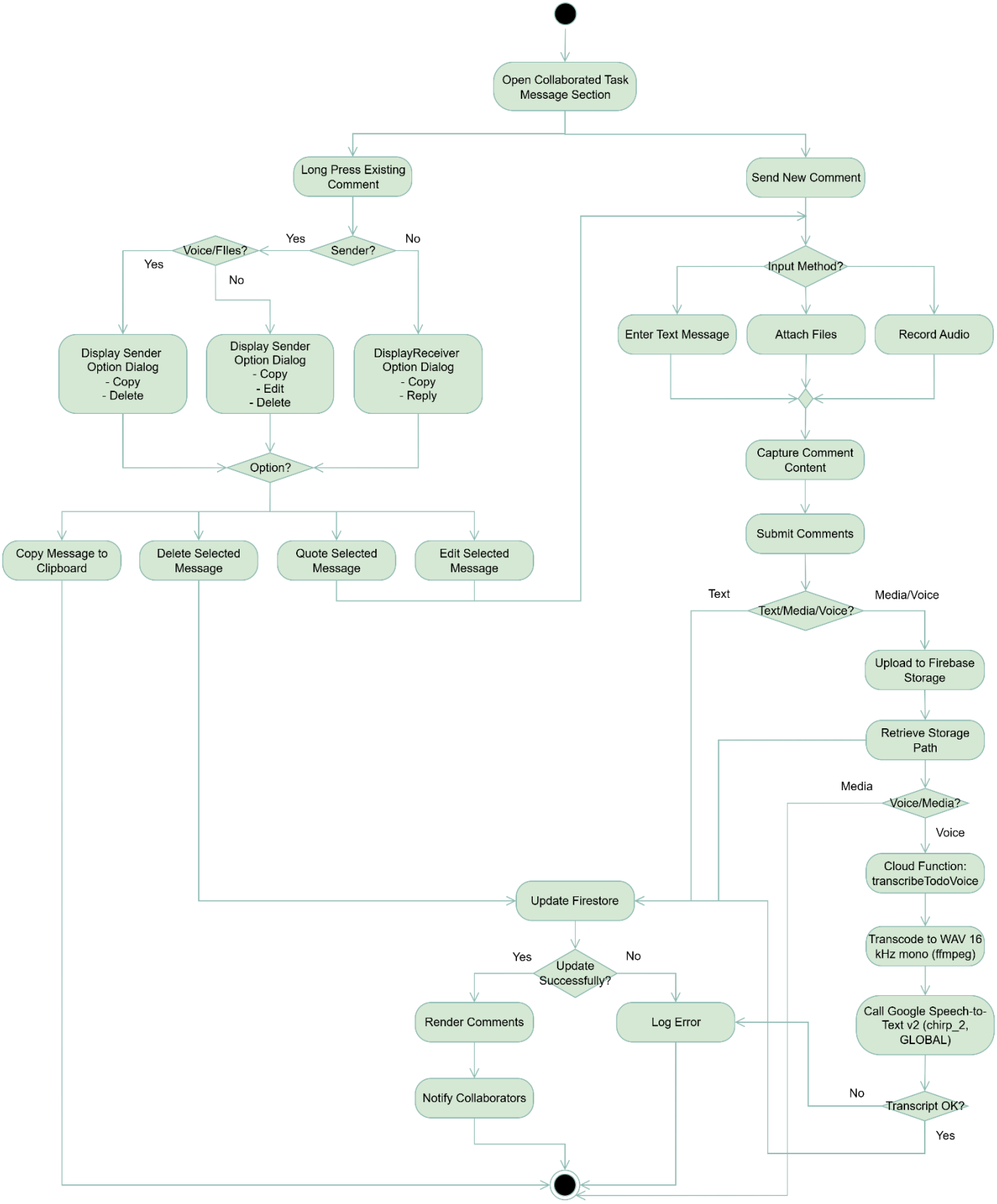

Figure 3.1.9 Activity Diagram – Manage Comments

Use Case Description

Use Case	Post and Manage Comments on a Collaborated Task
Actor	User, System, Firestore
Pre-conditions	1. User is authenticated and authorized for the task. 2. Microphone permission granted (for voice). 3. Device has basic connectivity (for posting; offline read is optional). 4. Cloud Function transcribeTodoVoice deployed and able to call Google STT
Main Flow	1. User opens collaborated task message section. 2. System shows input methods: **Enter Text**, **Attach Files**, **Record Audio**. 3. User provides content: - **Text** → type in composer. - **Voice** → press/hold to record, then stop. - **Attach File** → system opens file picker (images/videos/docs); preview is shown. 4. User taps **Send**. 5. System validates (non-empty, proper file size/type) 6. System writes a comment to Firestore (type, author, timestamps, file/audio URLs, etc.) and renders it optimistically. 7. **If voice/ media:** System uploads path to Firebase Storage and creates/updates Firestore. **Async STT Pipeline (Cloud) — Always On** 8. Storage **finalize** triggers transcribeTodoVoice. 9. Transcode to WAV 16 kHz mono (ffmpeg). 10. Function calls **Google Speech-to-Text v2** (GLOBAL/chirp_2; fallback model as configured). 11. **On success**: - Write transcribed text, language code, confidence to Firestore. - Update Firestore: voices/{voiceId}.status = "done". 12. System updates the UI showing the new comment in the thread. 13. System identifies all other collaborators on the task. 14. System sends each a notification (push or in-app). 15. **On failure**: - Set voices/{voiceId}.status = "error" and log details.

	- Clients pick up snapshot changes and update the UI: the transcript button becomes **"View transcript"**.
Alternative Flow	**AF-1: Edit Existing Comment Long-press Actions** 1. User long-presses or taps an existing comment. 2. System shows an option dialog box: - **Sender + Text** → Copy, Edit, Delete - **Sender + Voice/Files** → Copy, Delete - **Receiver (any)** → Copy, Reply (Quote) **AF-2: View / Hide Transcript (per voice bubble)** 1. User taps **"View transcript"** under a voice message. 2. System expands an inline panel showing transcribed text. The button changes to **"Hide transcript"**. 3. User can collapse it again; button toggles back to **"View transcript"**. **AF-3: Reply with Quote** • User taps **Reply**, composer shows quoted preview (of text or voice label). • On Send, a new comment is created with quoted message reference. **AF-4: Edit Existing Comment (Sender + Text)** • Inline edit, save sets edited=true, editedAt, rerender, notify. **AF-5: Delete Existing Comment (Sender)** • Confirm → soft delete; update Firestore, rerender.
Exception Flow	**EF-1: Validation Failure** • Empty input or invalid/oversized file → highlight and stay on composer. **EF-2: Save Failure** • Firestore write fails. **EF-3: STT Failure** • Comment remains playable, no transcript. **EF-4: Notification Failure** • Log and retry with backoff; comment state is not rolled back.
Post-conditions	1. New/edited/deleted comments persisted and rendered to all participants. 2. Voice comments are always transcribed (even if no transcode is needed).

	3. Transcript text is saved to Firestore and indexed for later search (together with text comments and extracted URLs). 4. Each voice bubble shows a toggle button to reveal/collapse the transcript. User preference (expanded/collapsed) affects only local UI unless you choose to persist it. 5. Collaborators (excluding author) are notified of new/edited/deleted comments.

Table 3.1.9 Use Case Description – Manage Comments

3.1.10 Voice/Text/URLs Comment Searching

Activity Diagram

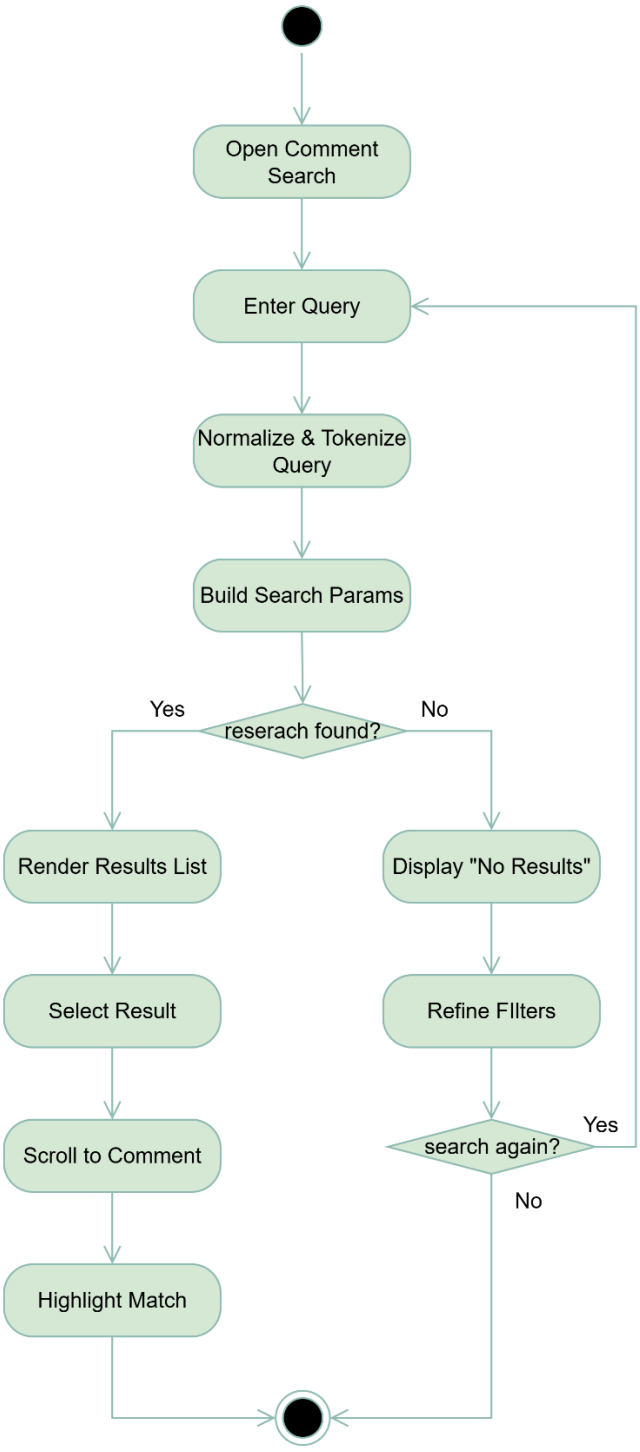

Figure 3.1.10 Activity Diagram – Voice/Text/URLs Comment Searching

Use Case Description

Use Case	Search Comments (Text / Voice Transcripts / URLs)
Actor	User, System, Firestore
Pre-conditions	1. User is authenticated and authorized for the task. 2. Comment content has been indexed for search: - text comments, - voice transcripts (from STT), - extracted URLs (and optional filenames). 3. Network available (or local cache available for offline search).
Main Flow	1. **Open Comment Search** – User launches the search UI. 2. **Enter Query** – User types keywords. 3. **Normalize & Tokenize Query** – System lowercases/normalizes text (diacritics, script forms), tokenizes (e.g., words/ngrams). 4. **Build Search Params** – System composes filters/scope (task, type = All/Text/Voice/URLs, author/date if available), paging, and ranking strategy. 5. **Query Search Index** – System queries Firestore index fields (e.g., searchText, searchTokens, URL tokens). 6. **If results found:** - Render Results List (type badges, snippets with bolded hits). - Select Result – User taps a result. - Scroll to Comment – System navigates to the exact message in the thread. - Highlight Match – System highlights the matched text. 7. **If results not found:** - Display "No Results" – System shows empty state with tips. - Refine Filters – User adjusts query/filters. - Search again? – If yes, loop back to Enter Query; if no, exit.
Alternative Flow	**AF-1: URL Search** • If the query is a URL (or domain/path fragment), results show URL badges/snippets. **AF-2: Pagination / Load More (optional)** • If results exceed page size, **Load More** appends additional matches.
Exception Flow	**EF-1: Offline / Network Error**

	• If offline, search falls back to cached index if present; otherwise show "No connection—try again" with a **Retry** action. • If the index query fails, show "Couldn't search right now. Retry?" (**Retry/Cancel**).
Post-conditions	1. The system shows matches (or a clear "No Results") without modifying data. 2. When a result is selected, the thread **scrolls to the comment** and the **matched fragment is highlighted**.

Table 3.1.10 Activity Diagram – Voice/Text/URLs Comment Searching

3.1.11 Notifications

Activity Diagram

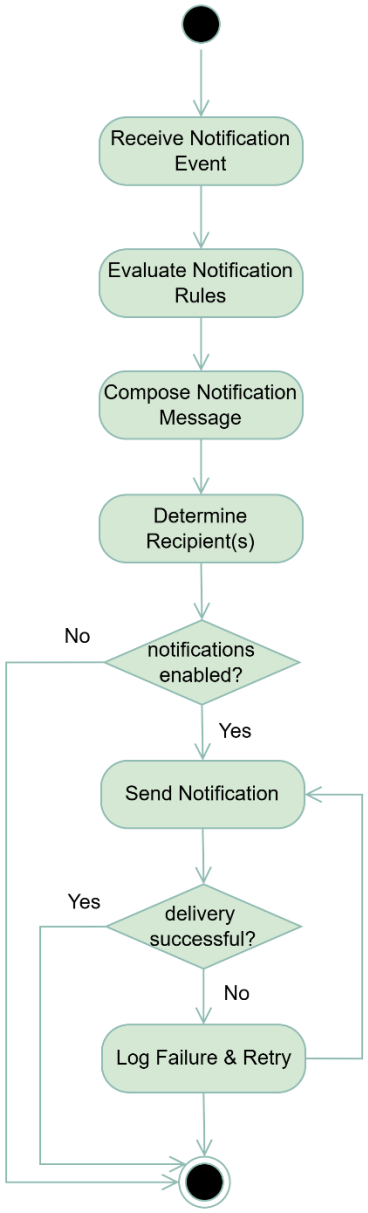

Figure 3.1.11 Activity Diagram - Notifications

Use Case Description

Use Case	Notification System
Actor	System, Push/Email Service, User
Pre-conditions	1. A triggering event has occurred (e.g., task deadline, completed task/event, comments, etc.). 2. The user has enabled notifications in their profile. 3. Notification rules (reminders, thresholds, channels) are configured.
Main Flow	1. An internal trigger fires. 2. System checks which rule(s) apply for this event (e.g. "5-minute reminder," "on-comment alert"). 3. System builds the payload: title, body text, metadata (links, IDs). 4. Based on ownership, assignments, collaborators, or global broadcast rules. 5. If a recipient has disabled notifications, skip them. 6. System hands off each message to the Push/Email Service (or SMS gateway). 7. For each channel, verify success. - On success → done. - On failure → log the error, attempt up to N retries (back to step 6).
Alternative Flow	For high-volume events, system groups multiple alerts into one batched message to reduce noise.
Exception Flow	1. If the notification fails to deliver (e.g., invalid email, system error), the system logs the failure and attempts redelivery. 2. If a user has disabled notifications, the system does not send the message.
Post-conditions	1. All enabled recipients receive a notification via their preferred channel. 2. Delivery attempts (and any failures) are logged for audit. 3. The user's UI (or lock-screen) shows the new alert.

Table 3.1.11 Use Case Description – Notifications

3.1.12 Device Calendar Sync

Activity Diagram

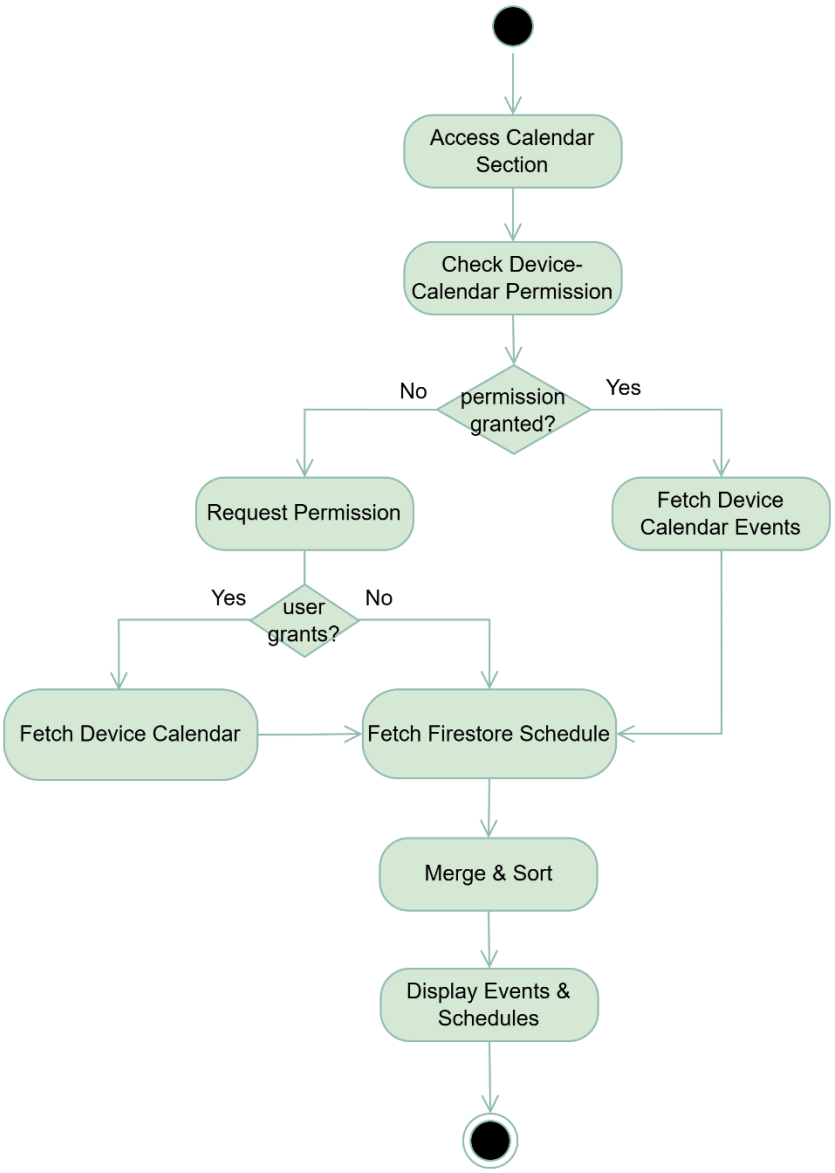

Figure 3.1.12 Activity Diagram –Device Calendar Sync

Use Case Description

Use Case	Synchronize Events with Device Calendar
Actor	User, Device Calendar Provider (OS)
Pre-conditions	1. The user is logged into the system. 2. The app has been granted READ_CALENDAR (and optionally WRITE_CALENDAR) permission at the OS level.
Main Flow	1. User navigates to the in-app Calendar tab (Home Screen). 2. System checks if it already has permission to read the device calendar. 3. **Request Permission** *(if needed)* - If not granted, system prompts the user: "Allow this app to access your device calendar?" - User taps **Allow** → system proceeds; **Deny** → go to Step 6. 4. System reads upcoming events from the device's built-in calendar provider. 5. In parallel, system queries Firestore for the user's app-created schedules. 6. System merges device events and Firestore schedules, orders them chronologically/ dynamic-reordering. 7. The UI presents a unified list or calendar view of both sets of entries.
Alternative Flow	1. If the user denies calendar permission, system skips Step 4, fetches only Firestore schedules (Step 5), merges (just the one source) and displays them. 2. The UI shows a banner "Enable calendar permission to view device events" with a button linking to Settings.
Exception Flow	**EF-1: Permission Permanently Denied** • If the OS reports "Don't ask again," system shows a persistent notice: Calendar access blocked – enable in Settings." • System continues to show only Firestore schedules. **EF-2: Read Error** • If fetching from the device calendar fails, log the error, show a toast "Unable to load device events," then proceed with Firestore schedules only. **EF-3: Firestore Unavailable**

	• If Firestore read fails, display "Cannot load your schedules; retry?" with **Retry**/**Cancel**.
Post-conditions	1. The user sees a combined view of device-native calendar events and their app schedules. 2. If permission is missing or an error occurs, at minimum the Firestore schedules are still displayed. 3. Any errors or permission issues are surfaced with clear UI guidance for next steps.

Table 3.1.12 Use Case Description – Device Calendar Sync

3.1.13 Register and User Login

Activity Diagram

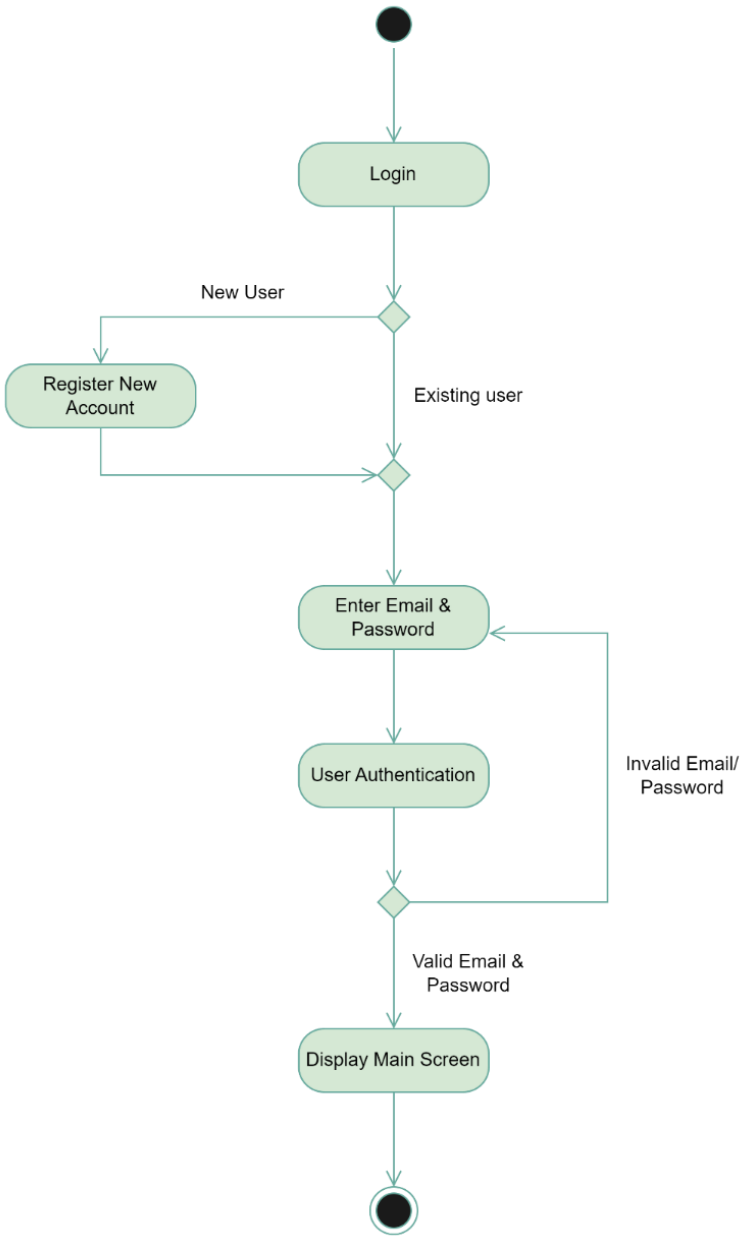

Figure 3.1.13 Activity Diagram – Register and User Login

Use Case Description

Use Case	User Login and Registration
Actor	User
Pre-conditions	1. The user must have an internet condition 2. If logging in, the user must already have an account.
Main Flow	1. The user opens the application and selects “Login”. 2. The system prompts the user to enter their email and password. 3. The user enters their credentials. 4. The system performs user authentication. - If the credentials are valid, proceed to Step 5. - If the credentials are invalid, prompt the user to re-enter the email and password. 5. The system displays the Main Screen.
Alternative Flow	1. Instead of logging in, the user selects “Register New Account”. 2. The system provides a registration form. 3. The user fills in the required details and submits the form. 4. The system creates the account and redirects the user to the main screen.
Exception Flow	1. If authentication fails multiple times, the system may lock the account or provide a password recovery option. 2. If registration fails (e.g., email already in use), the system notifies the user and asks for corrections.
Post-conditions	1. If successful, the user is redirected to the main screen. 2. If unsuccessful, the user remains at the login page with appropriate error messages.

Table 3.1.13 Use Case Description – Register and User Login

3.1.14 Home Page

Activity Diagram

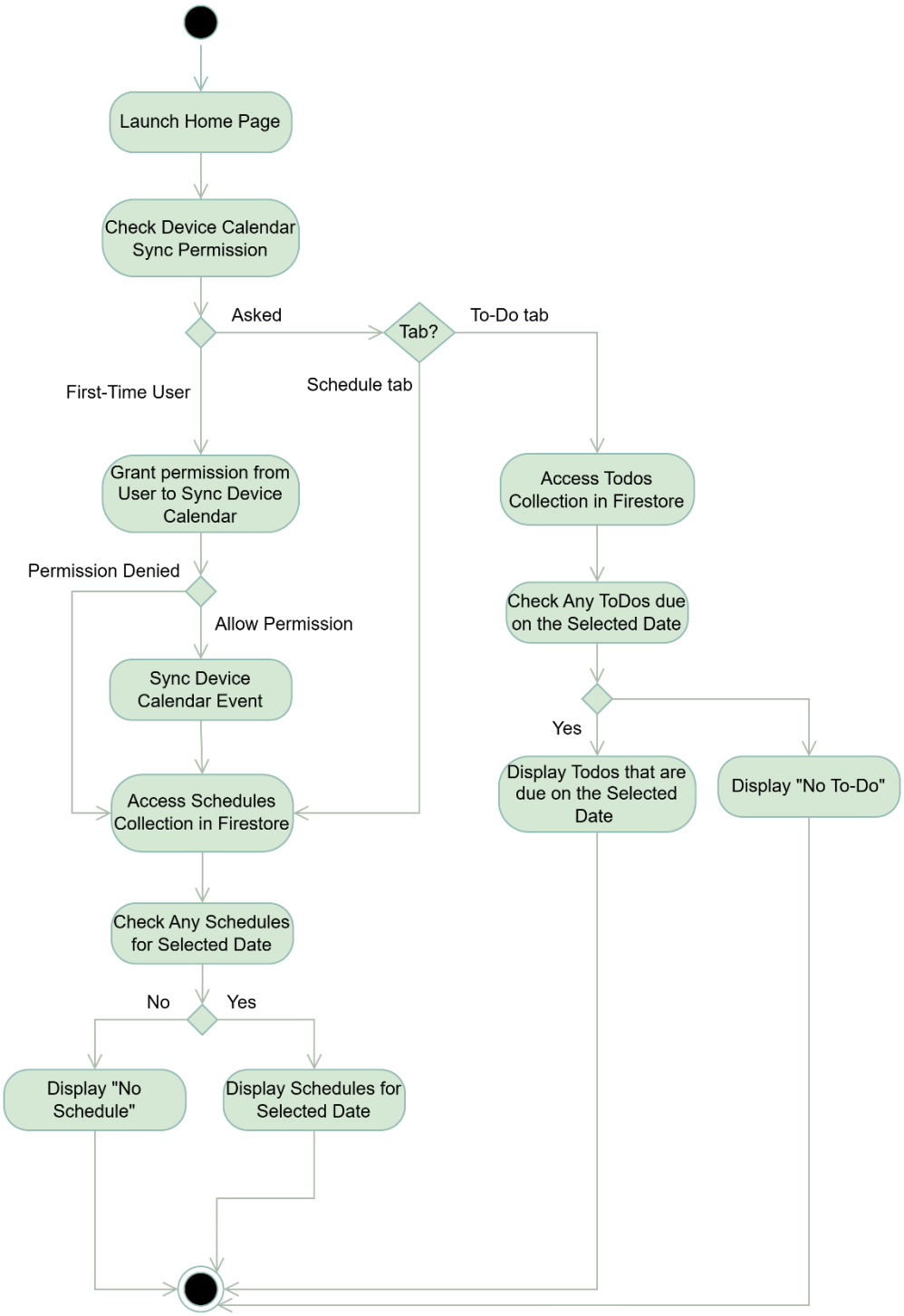

Figure 3.1.14 Activity Diagram – Home Page

Use Case Description

Use Case	View Daily Schedules & To-Dos
Actor	User
Pre-conditions	1. The user is authenticated, and the Home page is loaded. 2. The device has network connectivity. 3. The app may or may not already have calendar-sync permission.
Main Flow	1. The user launches the home page. 2. Check calendar-sync permission. 3. Request permission (if not already granted). - System prompts: "Allow app to sync with your device calendar?" - User grants permission 4. Sync device calendar events. 5. Display calendar table and two tabs: Schedules | To-Dos. 6. User selects a tab for a specific date. - If Schedules chosen, system fetches Firestore schedules and newly synced device-calendar events for the selected date. - If To-Dos chosen, system fetches Firestore to-dos and filters for items due on the selected date. 7. If To-Dos chosen: - System fetches Firestore to-dos. - System filters for items due on the selected date.
Alternative Flow	**AF1: Permission Already Granted** • Steps 3.1–3.2 skipped; proceed directly to Step 4. **AF2: Permission Denied** • User declines calendar-sync request in Step 3. • System displays a toast: "Calendar sync disabled; showing app schedules only." • On the Schedules tab, system fetches only Firestore schedules (skip Step 4).
Exception Flow	**EF1: Calendar Sync Failure** • If syncing the device calendar errors out, log the failure, show "Unable to sync calendar; showing app schedules," then continue to fetch Firestore schedules. **EF2: Firestore Unavailable**

	• If reading schedules or to-dos from Firestore fails, display an error dialog: "Cannot load data. Retry?" and allow the user to retry or cancel.
Post-conditions	1. The Home page shows the user's schedules (including device events if permitted) or to-dos for the selected date. 2. If no items exist, the corresponding "No Schedule" or "No To-Do" message is displayed.

Table 3.1.14 Use Case Description – Home Page

3.1.15 Schedule

Activity Diagram

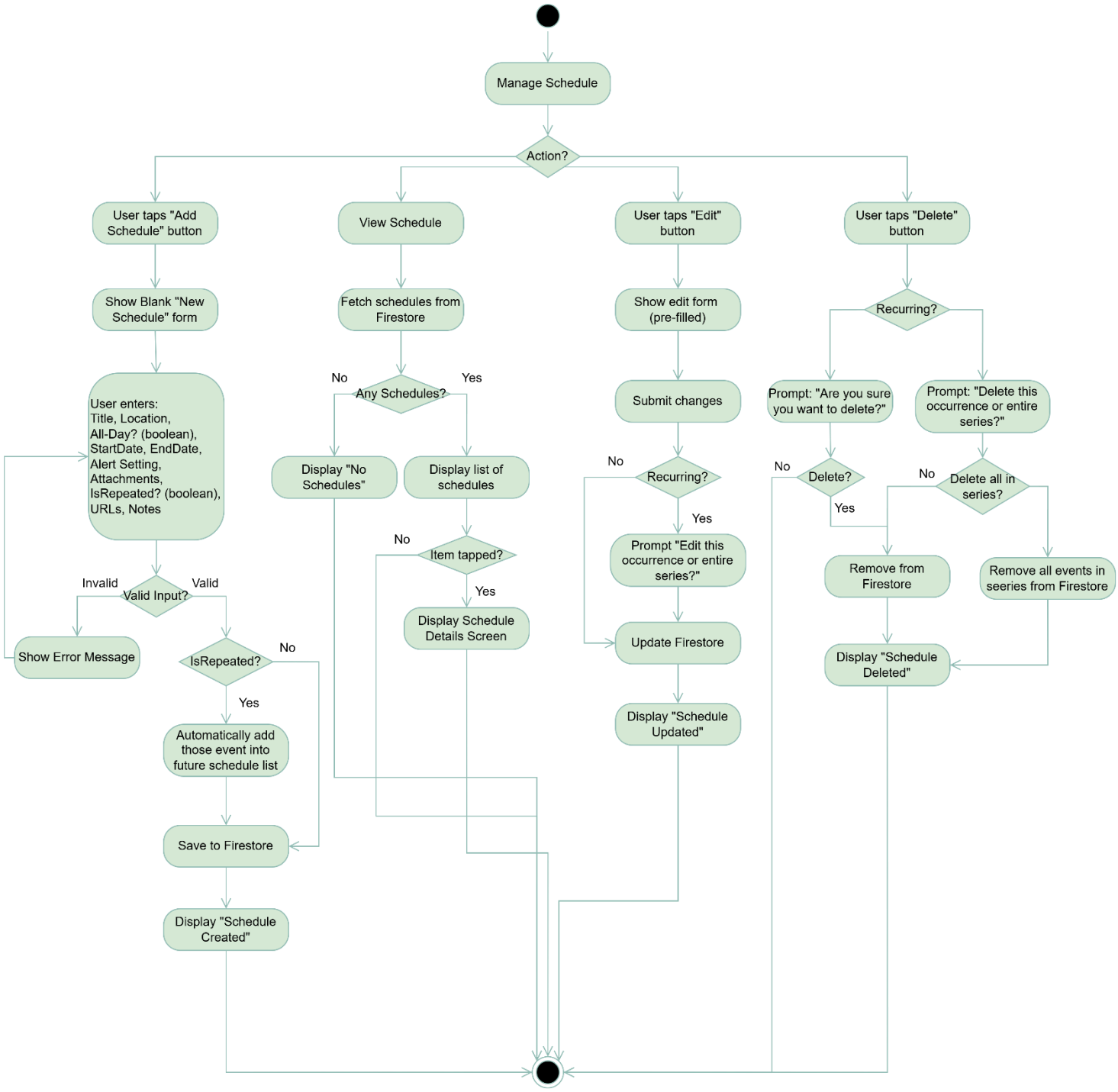

Figure 3.1.15 Activity Diagram – Manage Schedule

Use Case Description

Use Case	Manage Schedule
Actor	**User:** wants to create, view, modify or remove personal schedule items. **System:** must reliably store and retrieve events, enforce data integrity, and handle errors gracefully.
Pre-conditions	1. The user is authenticated and has navigated to the Schedule List screen. 2. The app has network connectivity and access to Firestore.
Main Flow	**1. Display Schedule List** - System fetches all schedules from Firestore and displays them in a scrollable list. - A prominent "+ Add Schedule" button is visible. **2. Create New Schedule** 2.1 User taps "+" button. 2.2 System shows a blank "New Schedule" form. 2.3 User enters required fields (title, location, start/end date-times, all-day flag, alert, attachments, and whether it repeats). 2.4 If "Is Repeated" is checked, system prompts for recurrence pattern and series end date. 2.5 System validates entries. If invalid, show error and remain on form (see AF-1). 2.6 On success, system writes the new schedule (and series instances, if any) to Firestore. 2.7 System returns to the Schedule List, refreshing to include the newly created item. **3. View Schedule Details** 3.1 From the Schedule List, user taps one schedule entry. 3.2 System opens the Schedule Details screen, showing all fields (title, location, start/end, alerts, recurrence info, attachments). 3.3 On this screen, two action buttons are available: Edit and Delete. **4. Edit Schedule** 4.1 User taps Edit button in the Schedule Details screen. 4.2 System displays a pre-filled edit form. 4.3 User modifies fields and submits.

	4.4 If the schedule is part of a series, system prompts: "Edit this occurrence only" or "Edit the entire series?" 4.5 System validates inputs (see AF-1) and then updates Firestore (single instance or whole series). 4.6 System shows "Schedule updated" and returns to the Schedule Details screen, refreshed. **5. Delete Schedule** 5.1 User taps Delete button in the Schedule Details screen. 5.2 System prompts: "Are you sure you want to delete this schedule?" 5.3 If it's recurring, system then prompts: "Delete this occurrence only" or "Delete the entire series?" 5.4 System performs the deletion in Firestore. 5.5 System shows "Schedule deleted" and returns to the Schedule List, refreshed.
Alternative Flow	**AF-1: Validation Error (Add/Edit)** • If the user submits a form with missing or invalid fields, system highlights errors (e.g. "End date must be after start date") and remains on the form until corrected. **AF-2: Empty Schedule List** • If the initial fetch in Step 1 returns no schedules, system displays "No schedules. Tap '+ Add Schedule' to create one." **AF-3: User Cancels Recurrence Prompt** • If the user cancels when asked about series vs. single instance, the flow returns to the form without making Firestore changes. **AF-4: User Navigates Back** • At any point on New Schedule, Schedule Details, or Edit Schedule screens, tapping a back button returns the user to the previous screen without saving changes.
Exception Flow	**EF-1: Firestore Read/Write Failure** • On any fetch, save, update, or delete error, system displays an alert ("Unable to communicate with server.") **EF-2: Network Unavailable**

	• If the device loses connectivity mid-operation, system shows "Connection lost. Please check your network," then returns to the **Schedule List** or current form.
Post-conditions	**Add:** A new schedule (and any generated series) exists in Firestore and appears in the Schedule List. **View:** No data is modified; details are displayed. **Edit:** The schedule (or series) in Firestore reflects the user's changes. **Delete:** The selected schedule instance(s) are removed from Firestore; the list is refreshed.

Table 3.1.15 Use Case Description – Manage Schedule

3.1.16 Schedule Dynamic Reordering

Activity Diagram

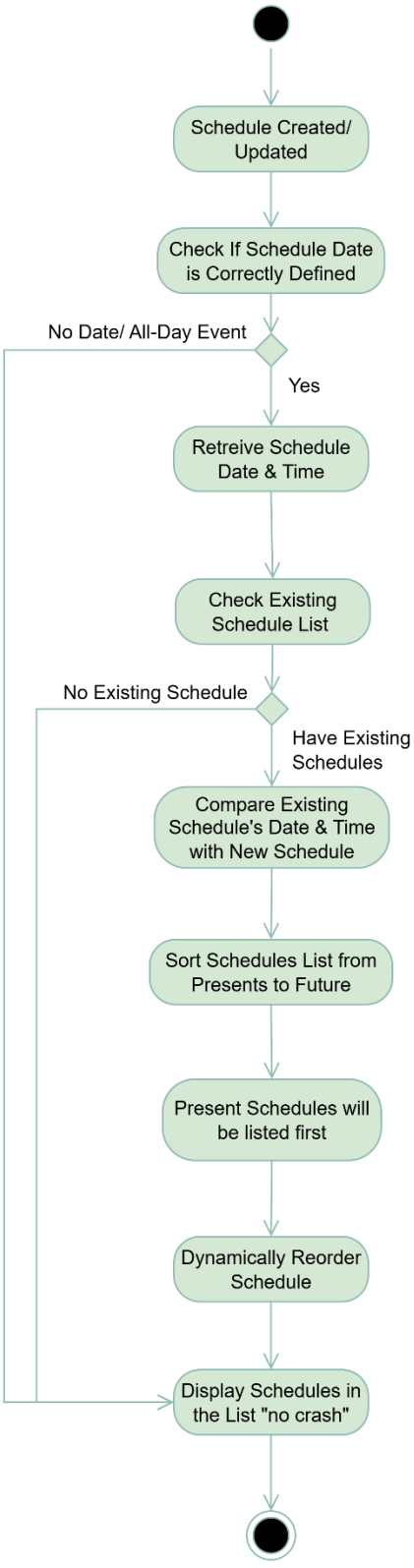

Figure 3.1.16 Schedule Dynamic Reordering

Use Case Description

Use Case	Schedule Dynamic Reordering
Actor	User, System (invoked when schedules are created/updated or when the user views their schedule list)
Pre-conditions	1. The user is authenticated and on (or is about to see) the Schedule List screen. 2. One or more schedule entries exist in Firestore (or are about to be created/updated). 3. The system clock is available.
Main Flow	1. The system receives a created or updated schedule. 2. The system checks if the schedule's date is correctly defined. 3. The system retrieves the date and time of the new schedule. 4. The system checks the existing schedule list. 5. The system compares the date and time of the existing schedules with the new schedule. 6. The system sorts the schedule list, ordering entries from present to future (present schedules are listed first). 7. The system dynamically reorders the schedule list as needed. 8. The system displays the updated schedule list to the user.
Alternative Flow	1. If Firestore returns zero schedules, the system skips sorting and displays an empty-state message ("No schedules to show"). 2. Treat any schedule with a null start-date as Present, placing it at the top of the list.
Exception Flow	**EF-1: Firestore Read Error** • If fetching schedules fails, show an alert: "Unable to load schedules. Retry?" • On Retry → go back to Step 4; on Cancel → abort and leave the previous list in place. **EF-2: Invalid Date Validation** • If any schedule's start/end dates are malformed, log the error, exclude that entry from the list, and continue processing the others.
Post-conditions	The user sees their schedules ordered so that all "present" (ongoing or past-started) events appear first, followed by upcoming events, each subgroup sorted chronologically.

Table 3.1.16 Use Case Description – Schedule Dynamic Reordering

3.2.17 To-Do

Activity Diagram

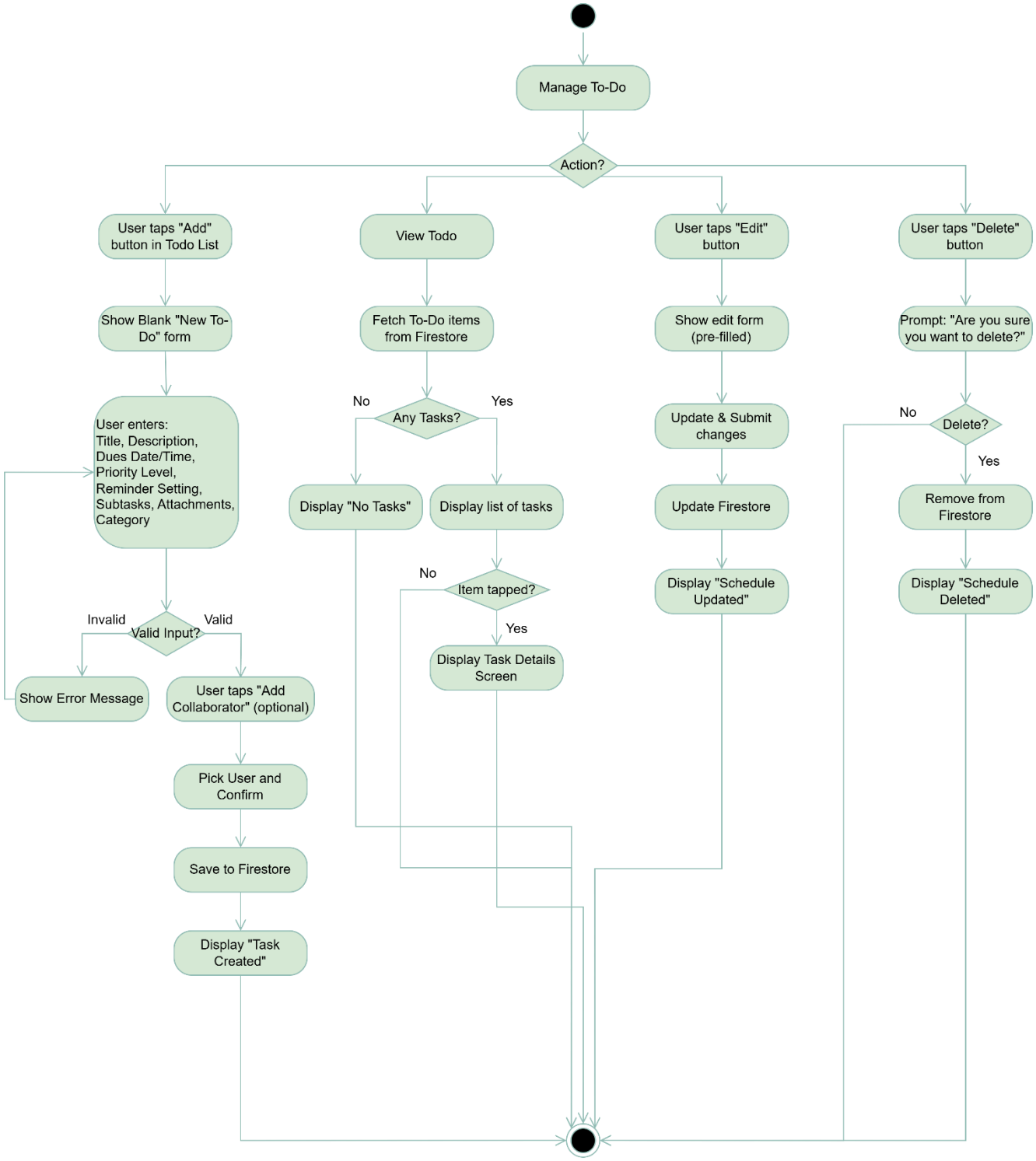

Figure 3.1.17 Activity Diagram – Manage To-Do

Use Case Description

Use Case	Manage To-Do
Actor	**User:** wants to create, view, modify or remove personal to-do items. **System:** must reliably store and retrieve tasks, enforce data integrity, and handle errors gracefully.
Pre-conditions	enticated and has navigated to the To-Do List screen. ve connection to Firestore.
Main Flow	**1. Display Task List** - System fetches all active to-do items from Firestore. - System displays a scrollable list of tasks (title, due date, priority) and an **Add** Task button. **2. Create New Task** 2.1 User taps **"+"** task button 2.2 System shows a blank "**New Task**" form. 2.3 User enters required fields. 2.4 User may add optional subtasks, attachments/images, category, and collaborators. 2.5 System validates entries. If invalid, highlight errors and remain on form (see AF-1). 2.6 On success, system writes the new task to Firestore. 2.7 System returns to the To-Do List, refreshing to include the newly created item. **3. View Task Details** 3.1 From the To-Do List, user taps one task entry. 3.2 System opens the **Task Details** screen, showing all fields. 3.3 On this screen, two action buttons are available: **Edit** and **Delete**. **4. Edit Task** 4.1 User taps **Edit** in the Task Details screen. 4.2 System displays a pre-filled edit form. 4.3 User modifies any fields, subtasks, attachments, or collaborators and taps **Save**. 4.4 System validates inputs (see AF-1). 4.5 On success, system updates the task in Firestore.

	4.6 System shows **"Task updated"** and returns to the refreshed Task Details screen. **5. Delete Tasks** 5.1 User taps **Delete** in the Task Details screen. 5.2 System prompts: "Are you sure you want to delete this task?" 5.3 If confirmed, system deletes the task from Firestore. 5.4 System shows **"Task deleted"** and returns to the To-Do List, refreshed.
Alternative Flow	**AF-1: Validation Error (Add/Edit)** • If required fields are missing or invalid (e.g., due date in the past), system highlights the errors and remains on the form until corrected. **AF-2: Empty To-Do List** • If no tasks exist on initial load, system displays **"No tasks. Tap '+' task to create one."** and bypasses steps 3–5. **AF-3: Cancel Delete** • At the delete confirmation prompt, if the user taps **Cancel**, system aborts deletion and stays on the Task Details screen.
Exception Flow	**EF-1: Firestore Read/Write Failure** • On any fetch, save, update, or delete error, system displays an alert: "Unable to communicate with server" **EF-2: Network Lost Mid-Operation** • If connectivity drops during a CRUD operation, system rolls back partial changes, shows **"Network error. Please try again."** and returns to the last stable screen.
Post-conditions	**Add:** The new task exists in Firestore and appears in the To-Do List. **View:** Task details are displayed; no data was modified. **Edit:** The selected task's record in Firestore reflects the user's updates. **Delete:** The selected task is removed from Firestore and no longer appears in the list.

Table 3.1.17 Use Case Description – Manage To-Do

3.1.18 Dynamic Weighted Task Prioritization

Activity Diagram

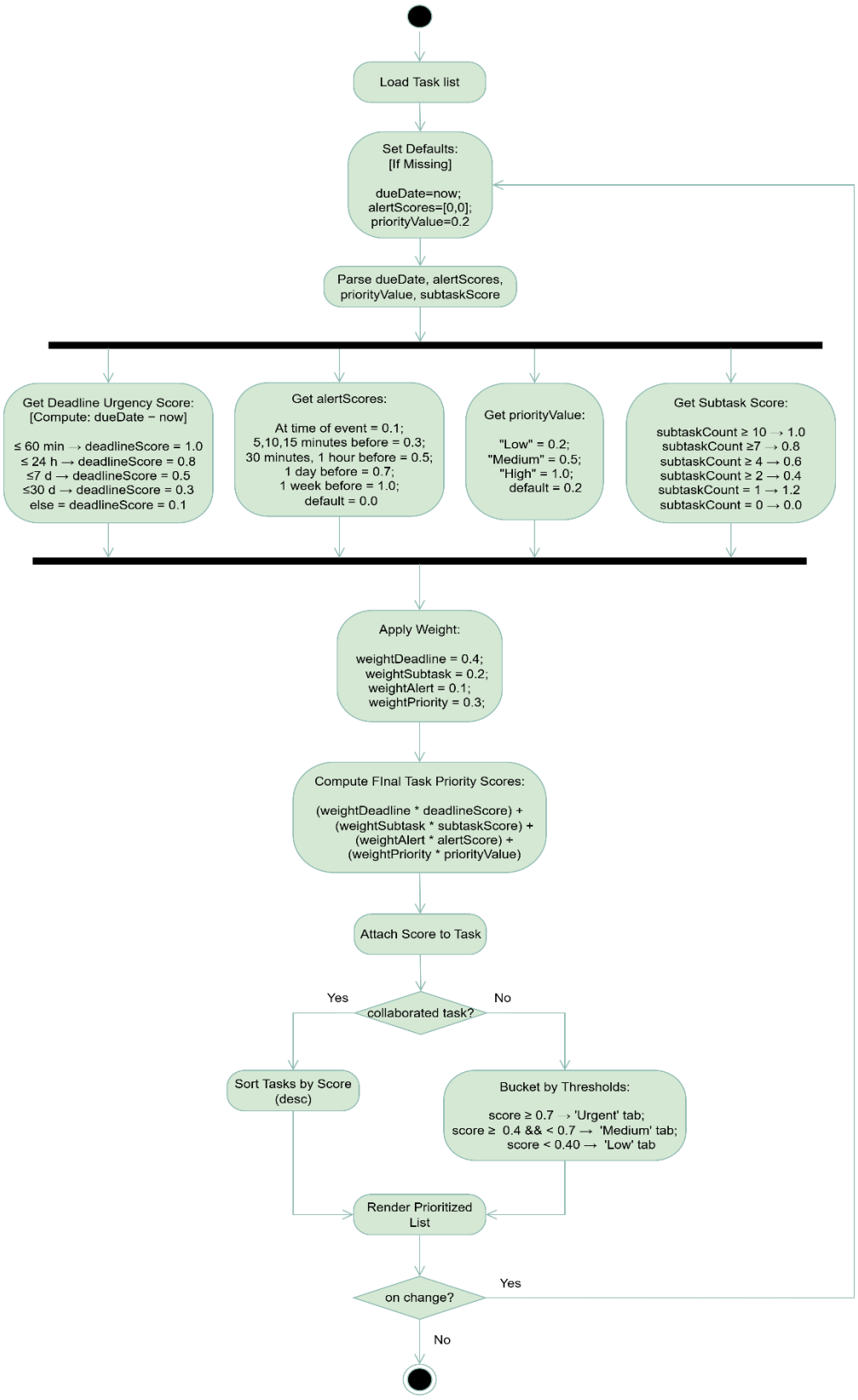

Figure 3.1.18 Activity Diagram – Dynamic Weighted Task Prioritization

Use Case Description

Use Case	Dynamic Weighted Task Prioritization
Actor	User, Firestore, System (Automated Scheduler)
Pre-conditions	1. User is authenticated and can read the tasks. 2. Task documents expose: dueDate, subtasks[], alertScores[], priorityValue (missing values allowed as defaults applied). 3. Device clock is available.
Main Flow	1. Load task list. 2. **Set defaults** for missing fields (dueDate=now, alertScores=[0,0], priorityValue=0.2). 3. **Fork → parallel conceptual computations**: - Compute deadline urgency score. - Compute subtask score. - Compute alert score. - Compute priority value. 4. **Apply weights:** Deadline=0.4, Subtask=0.2, Alert=0.1, Priority=0.3. 5. **Compute final score:** score = 0.4*deadline + 0.2*subtask + 0.1*alert + 0.3*priority. 6. Attach score to task. 7. If collaborated task: sort by score (descending). 8. If not collaborated task: **Bucket by thresholds;** ≥0.70 → Urgent, 0.40–<0.70 → Medium, <0.40 → Low (sort within buckets by score descending order). 9. Render prioritized list. 10. **Subscribe for changes** (task/subtask/alerts/priority); on change, recompute from step 2 and re-render.
Alternative Flow	**AF-1: Empty list** • show empty state, no scoring. **AF-2: Manual refresh** • re-fetch, recompute, re-render. **AF-3: View toggle (All vs Collaborated)** • switch between sort-only vs bucketed layout.
Exception Flow	**EF-1: Invalid/missing dueDate** • default to now, log, continue. **EF-2: Read failure**

	• show cached list (if any) and a retry banner; recompute when data arrives. **EF-3: Numeric safety** • clamp score to [0..1]; guard against null value.
Post-conditions	1. Each task has a computed **priority score** in [0..1]. 2. The list is **sorted** by score (descending) or **bucketed** into Urgent/Medium/Low (per your diagram). 3. The view updates automatically when inputs change.

Table 3.1.18 Use Case Description – Dynamic Weighted Task Prioritization

3.1.19 Past Activity – Completion Detection

Activity Diagram

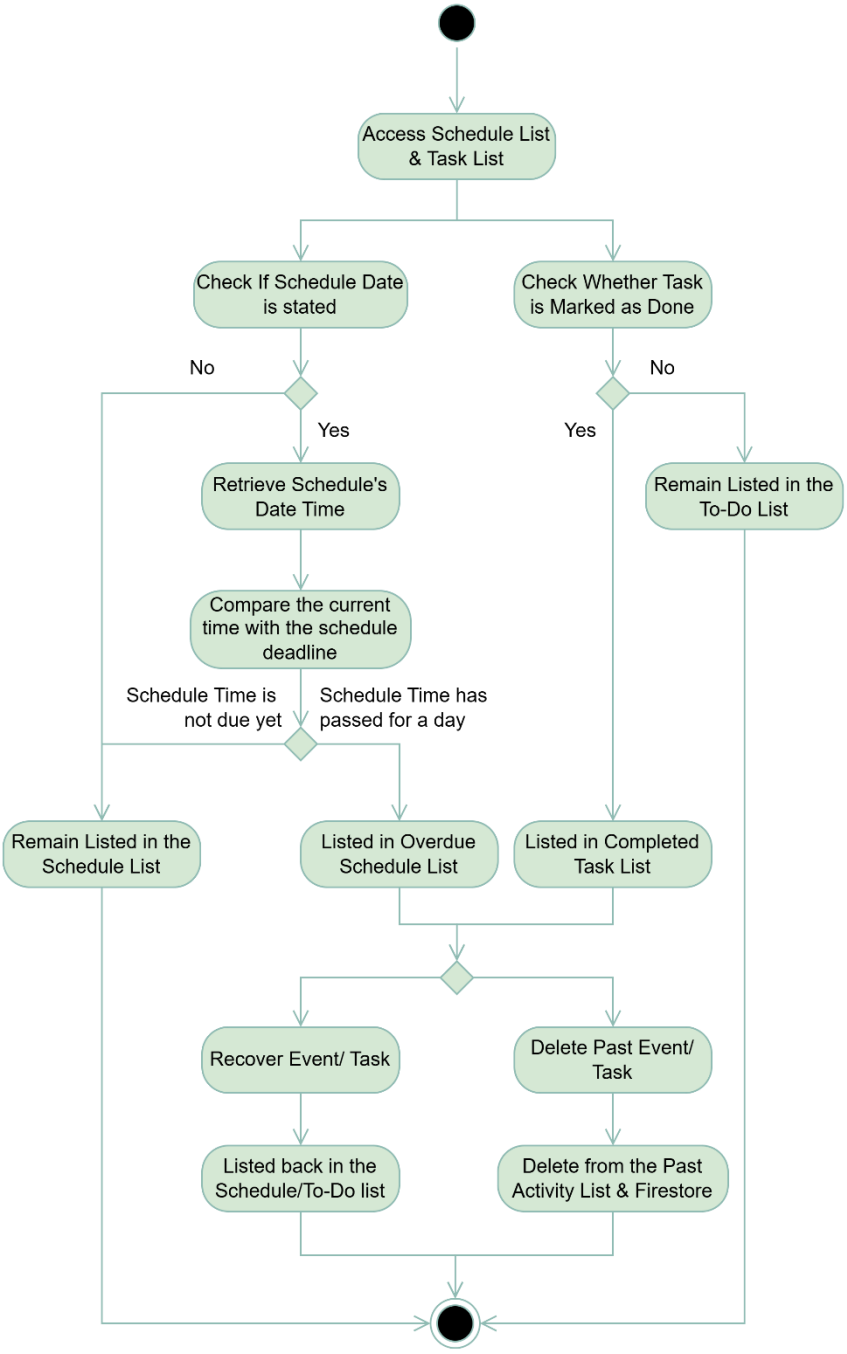

Figure 3.1.19 Activity Diagram – Past Activity

Use Case Description

Use Case	Past Activities Management
Actor	User
Pre-conditions	1. The user is logged into the system. 2. The system has previously recorded tasks or events that are completed or overdue.
Main Flow	**1. System evaluates schedules:** 1.1 For each schedule with a defined date, system compares currentTime vs. schedule.deadline + 1 day 1.2 If overdue, system moves that schedule from the **Active Schedule List** into the **Overdue Schedules** screen's list. Otherwise it remains active. **2. System evaluates to-dos:** 2.1 For each task, system checks • task.isCompleted == true **OR** • all subtasks are completed 2.2 If completed, system moves that task from the **Active To-Do List** into the **Completed Tasks** screen's list. Otherwise it remains active. 3. The Schedule List and To-Do List screens now omit any moved items. **4. User views overdue schedules:** 4.1 User navigates to the **Overdue Schedules** screen. 4.2 System displays all archived schedules there. 4.3 User selects one and chooses either: **Recover** → system restores it to the Schedule List and removes it from Overdue Schedules. **Delete** → system prompts "Permanently delete?"; if confirmed, it deletes from Firestore and the Overdue list.4.4. Screen refreshes to reflect the change. 4.4 Screen refreshes to reflect the change. **5. User views completed tasks:** 5.1 User navigates to the Completed Tasks screen. 5.2 System displays all archived tasks there. 5.3 User selects one and chooses either:

	Recover → system restores it to the To-Do List and removes it from Completed Tasks. **Delete** → system prompts "Permanently delete?"; if confirmed, it deletes from Firestore and the Completed list.
Alternative Flow	If a task or event remains in the past activity list for an extended period of 30 days, the system may automatically delete it.
Exception Flow	If a user attempts to recover a past task/event but the related data is missing or corrupted, the system displays an error message and prevents recovery.
Post-conditions	1. Overdue schedules appear only in the **Overdue Schedules** screen. 2. Completed tasks appear only in the **Completed Tasks** screen. 3. Recovering an item returns it to its original active list and removes it from the archive. 4. Deleting an item removes it permanently from Firestore and the archive screen.

Table 3.1.18 Use Case Description – Past Activity

3.2 Methodology

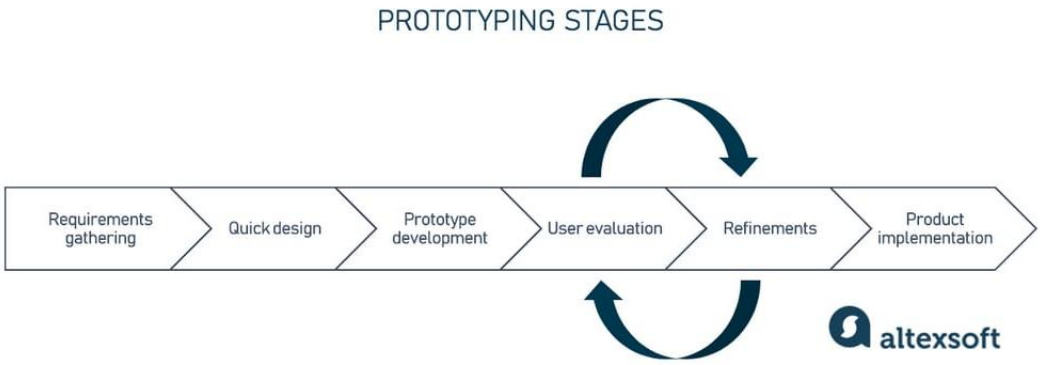

Figure 3.2 Prototyping Methodology

The proposed project follows the prototyping methodology, which is widely used in software development for its iterative and user-centered approach [37][38]. This methodology consists of six essential phases, allowing for continuous feedback and improvements throughout the development cycle. The first phase is requirement analysis, where user needs and expectations are gathered through discussions and research. The next phase is quick design, where an initial draft of the application is created. This design is not final but serves as a foundation for discussions with my supervisors, allowing early modifications based on expert feedback [39]. Following design approval, the prototype development phase begins, where the system's core functionalities are implemented. Once a working prototype is ready, the evaluation phase is conducted by my supervisor. The feedback helps identify usability issues and areas for improvement. Based on the evaluation, the refinement phase is initiated, making necessary adjustments through multiple iterations [40]. This cycle continues until the prototype meets expectations in terms of usability, functionality, and efficiency. The final phase is implementation, where the completed application is tested under various scenarios before deployment. Regular maintenance ensures long-term usability and efficiency. The prototyping methodology is ideal for this project as it allows quick modifications, early issue detection, and ensures that the final system is practical and user-friendly.

3.3 Implementation Challenges and Issues

Throughout the development of this project, which spanned both FYP I and FYP II, several critical challenges arose that shaped the implementation process. One of the earliest difficulties involved gaining proficiency in the Flutter framework and Dart programming language. Substantial time was devoted to studying architectural patterns, state management approaches, and user interface structuring techniques. This required iterative practice and continuous refinement of the codebase before stable and functional modules could be produced.

Another significant challenge concerned the integration of Artificial Intelligence (AI) and speech-to-text algorithms. Considerable effort was invested in configuring Google's Generative AI (Gemini) for assistant features and Cloud Speech-to-Text for transcription, both of which demanded multiple trials and adjustments to align with system requirements. Additional complications included the possibility of Gemini being overloaded during periods of high traffic, which occasionally disrupted responses, and the slower transcription speeds of Google Cloud Speech-to-Text when handling multilingual inputs without explicitly narrowing the language region for Malaysian speakers.

Cloud service management also presented difficulties. To enable features such as attachments, storage, and collaboration, subscription to the Firebase Blaze plan was required. Although the plan provided flexible scaling, usage beyond the no-cost tier. For example, storage above 1 GB or bandwidth exceeding 10 GB introduced budgetary constraints. Continuous monitoring of usage across Cloud Functions, Firestore, and Storage was necessary to prevent exceeding limits. Consequently, attachment storage was optimised, and non-essential email services were discontinued. Notifications were retained solely within the application rather than delivered through email, as integrating external mailing services such as SendGrid or Brevo required billing upgrades that were not feasible within the project's scope.

Additional challenges arose in implementing reminders and notification scheduling, where inconsistencies were observed in triggering alerts at the intended times. These issues were partly attributed to permission handling and background execution restrictions. While refinements in scheduling logic improved performance, further optimisation remains necessary. Collaboration features also posed complexity in ensuring real-time synchronisation and providing an intuitive interface. Basic functionality such as collaborator assignment and file attachment was implemented in FYP I, while more advanced functions, including detailed comments, integrated voice transcription, and smart search, were developed during FYP II.

3.4 Project Timeline

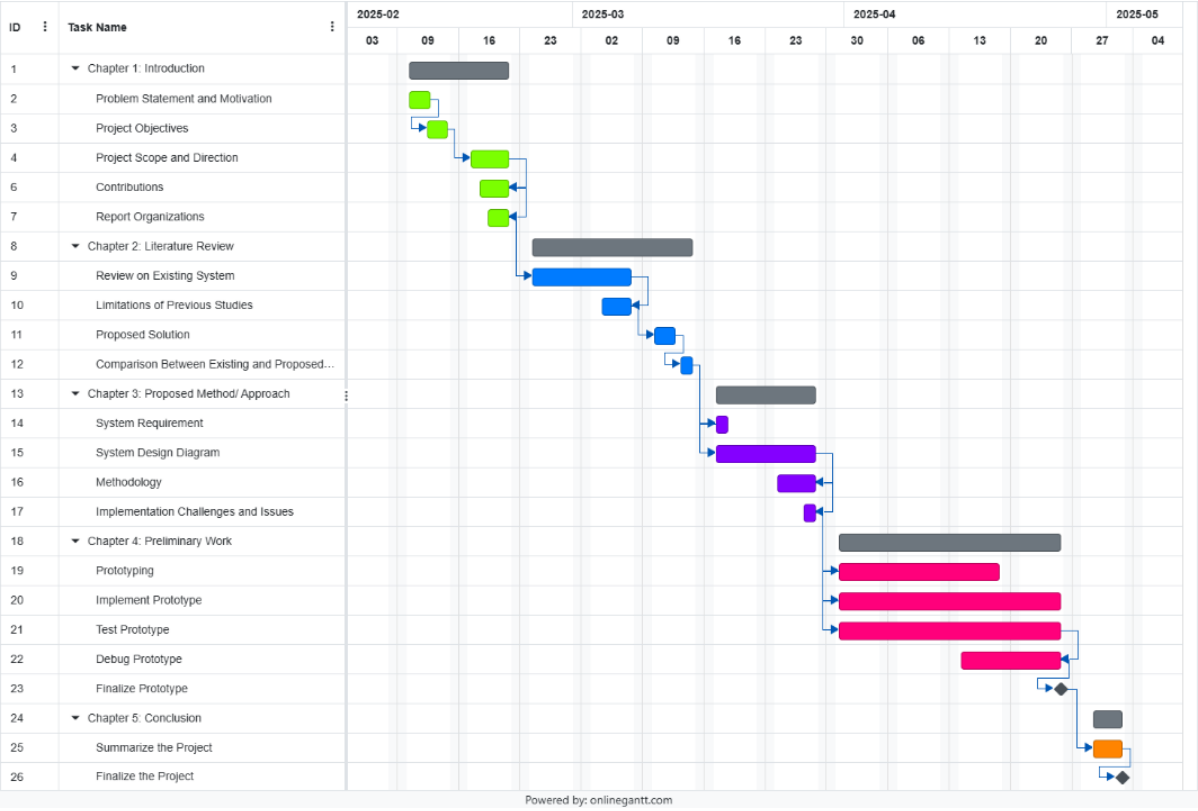

Figure 3.4.1 FYP 1 Timeline – Gantt Chart

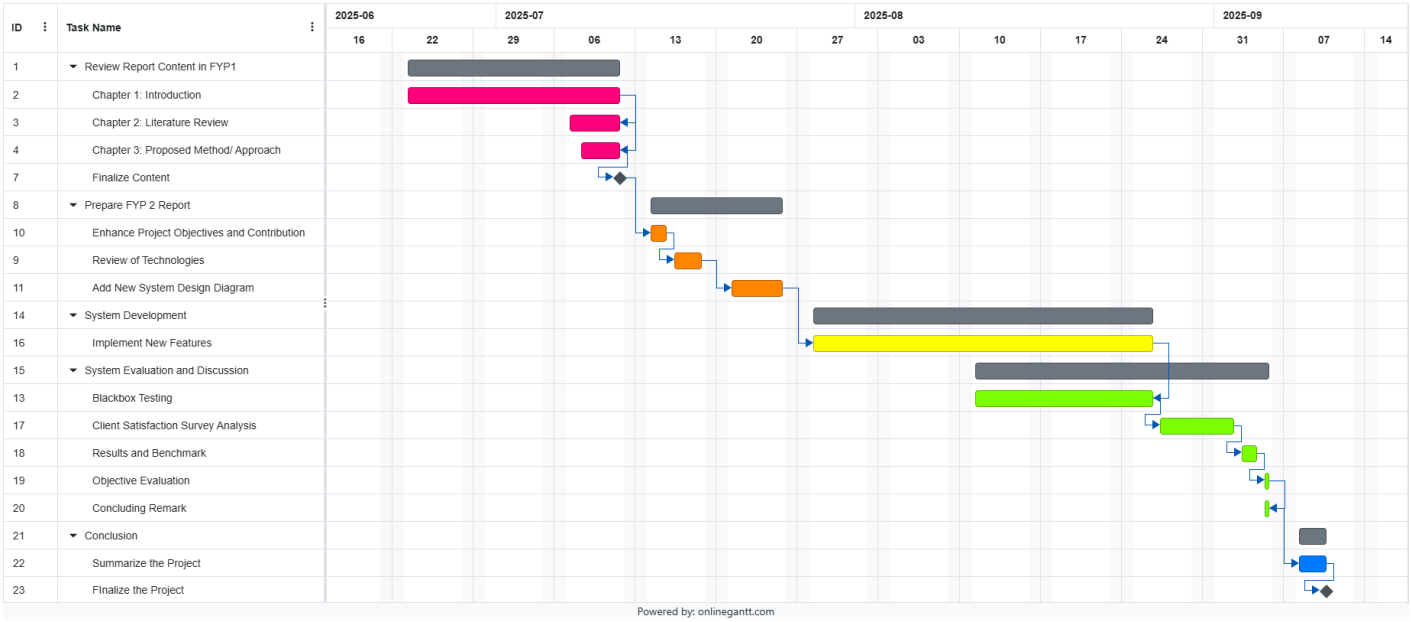

Figure 3.4.2 FYP 2 Timeline – Gantt Chart

CHAPTER 4

System Evaluation and Discussion

4.1 Blackbox

4.1.1 Authentication

Module/ Test Name	Input/ Pre-conditions	Steps	Expected Output	Actual Output	Pass/ Fail
Register – valid credentials	New email, strong password (>=8, letters + digit)	Submit Register	Account created; user doc written, navigates to Home	Account successfully created, Firestore user's doc stored, navigated to Home.	Pass
Register – duplicate email	Email already existed	Submit Register	Error "Email already in use"; stay on Register	Error shown: "Email already in use".	Pass
Register – invalid email format	abc@	Submit Register	Inline validation error; no account created	Inline error, button disabled until fixed.	Pass
Register – weak password	123456	Submit Register	Inline "weak password"; no account created	Inline "Password should be at least 6 characters" (Firebase Auth default).	Pass
Login - valid	Correct email & password	Submit Login	Navigates to Home; FCM token stored/updated	Redirected to Home; FCM token saved under user profile.	Pass
Login – wrong password	Valid email, wrong password	Submit Login	Error "Invalid credentials"; remain on Login	Error "The password is invalid".	Pass
Forgot password – registered email	Known email	Request reset	Password reset email sent; success toast	Password reset email sent; confirmation toast displayed.	Pass
Forgot password – unregistered email	Unknown email	Request reset	Non-disclosing message "If this email exists, a link was sent"	Generic message sent (non-disclosing).	Pass
Session persistence – relaunch app	User logged in	Kill & relaunch app	User remains logged in (unless signed out)	User stayed logged in after relaunch (Firebase Auth persistence working).	Pass
Sign out – normal flow	Logged-in user	Tap Sign Out	Session cleared; back to Login	Session cleared, returned to Login screen.	Pass

Table 4.1.1 Test Case - Authentication

4.1.2 Schedule Management (CRUD, Reordering, Overdue)

Module/ Test Name	Input/ Pre-conditions	Steps	Expected Output	Actual Output	Pass/ Fail
Add schedule – normal	Valid title; future start/end	Create	Appears in list with correct times	Saved to schedules collection, appeared in list immediately (synced real-time).	Pass
Edit schedule – normal	Existing schedule	Edit fields	Updated values persist; list reflects changes	Updated fields persisted, UI refreshed instantly.	Pass
Delete schedule – normal	Existing schedule	Delete	Removed from list; no ghost entry	Entry removed from Firestore and UI.	Pass
Reordering – present before future	Start = now−1m, end = now+29m	Create	Present schedules listed first	Ongoing schedule shown above future ones.	Pass
Overlap – two events	[10:00–11:00] & [10:30–11:30]	Create both	Both shown; sorted by start, then earlier end	Both events displayed; sorted by start time.	Pass
Both shown; sorted by start, then earlier end	End < now−24h	Wait or create past	Auto-moved to Overdue; not in active list	Auto-moved into “Overdue” section.	Pass
Invalid time – end < start	End before starts	Create	Validation error; cannot save	Validation error: cannot save.	Pass
Time zone switch handling	Change device TZ	Reopen app	Times reflect local TZ; no data corruption	All times recalculated to device TZ correctly.	Pass

Table 4.1.2 Test Case – Schedule Management

4.1.3 To-do Management (CRUD, Subtasks, Archive)

Module/ Test Name	Input/ Pre-conditions	Steps	Expected Output	Actual Output	Pass/ Fail
Create to-do with subtasks	Title; due date; 3 subtasks	Save	To-do saved; subtasks listed	Stored in todos collection with subtasks subcollection; UI shows all subtasks.	Pass
Complete via subtasks – auto	3 subtasks (all unchecked)	Mark all complete	To-do auto becomes Completed	Auto-marked as Completed when last subtask done.	Pass
Complete via manual toggle	No subtasks	Toggle “Done”	To-do archived to Completed	Task moved to Completed list.	Pass
Boundary – exactly 10 subtasks	10 subtasks	Save	Workload bucket “≥10” applied; UI remains responsive	Accepted; workload score capped; UI still smooth.	Pass

Boundary – due = now + 1h	Due in 60 minutes	Save	Urgency bucket "≤1 hour" applied	Urgency score mapped to "≤1 hour".	Pass
Invalid due date – past	Due yesterday	Save	Validation/warning; save blocked or corrected	Save blocked; error message shown.	Pass
Edit to-do – title/priority	Existing to-do	Edit & Save	Changes persist; lastModified updated	Title and priority updated; lastModified timestamp changed.	Pass
Delete to-do – cascade check	To-do with comments/ subtasks	Delete	Removed; no orphaned subcollections	Removed from Firestore; subcollections deleted by cascade.	Pass
Collaborated Tasks – views	Existing collaborated task list	View & Action	Only creator and admin can toggle "Done"	Only creator/admin could toggle Done; normal users restricted.	Pass

Table 4.1.3 Test Case – Todo Management

4.1.4 Collaboration & Comments (Text, Voice, Files, URLs)

Module/ Test Name	Input/ Pre-conditions	Steps	Expected Output	Actual Output	Pass/ Fail
Add collaborator – valid	User email exists	Add collaborator	Collaborator appears; in-app notification sent	Collaborator added; in-app notification received.	Pass
Post text comment – normal	Connected	Send comment	Appears instantly to all (real-time)	Comment appeared in real-time across devices.	Pass
Post URL – autolink	https://…	Send	Clickable link; opens external browser	Auto hyperlinked; tap opened browser.	Pass
Attach file – within limit	Valid PDF ≤ size cap	Upload	Upload succeeds; thumbnail/filename shown	Upload succeeded to Firebase Storage; link stored in Firestore.	Pass
Record voice (EN/BM/CN)	Short note (10s)/ Long not (>60s)	Upload	Storage object created (Transcript stored in EN/BM/CN); voice bubble visible	File uploaded, transcript auto generated; Mandarin returned in pinyin.	Partial
Unsupported audio – handling	e.g., unusual code	Upload	Transcoded or rejected with descriptive error	File transcoded to WAV; if failed, error recorded in comment doc.	Pass
Delete own comment	Own text/voice	Delete	Entry removed for all; permissions enforced	Removed from Firestore for all users.	Pass

Reply message	Existing messages send by others	Reply	Message quoted correctly and allow to jumpy to message successfully	Quoted message displayed, tapping jumps to original message.	Pass

Table 4.1.4 Test Case – Collaboration and Comment

4.1.5 Dynamic Weighted Task Prioritization (Scoring & Reordering)

Module/ Test Name	Input/ Pre-conditions	Steps	Expected Output	Actual Output	Pass/ Fail
Ranking – urgent vs distant	Task A: due today, High Task B: due +14d, Low	Create both	A ranks above B	A ranks above B	Pass
Boundary – deadline buckets	Tasks due at 1h, 24h, 7d, 30d	Create tasks	Each mapped to correct urgency bucket; order reflects buckets	Each mapped to correct urgency bucket; order reflects buckets	Pass
Boundary – workload buckets	Subtasks: 0, 1, 3, 6, 9, 10	Create tasks	Scores: 0; 1.2 bump for single; 0.4; 0.6; 0.8; 1.0	Scores assigned 0, 1.2, 0.4, 0.6, 0.8, 1.0 as expected.	Pass
Alert aggressiveness mapping	Alerts: none/5m/30m/1h/ 1d/1w	Set & save	Alert values mapped; final score P(t) consistent	Mapped correctly (None=0 → 1 week=highest weight).	Pass
Tie-break policy	Two tasks same P(t)	Compare	Order by earlier due date → higher user priority → more subtasks	Earlier due date listed first; if same, higher user priority wins.	Pass
Recompute on time drift	Task near bucket boundary	Wait threshold passes	List reorders when bucket changes; no crash	When due time crossed boundary, list auto-reordered.	Pass

Table 4.1.5 Test Case – Dynamic Weighted Task Prioritization

4.1.6 Completion Detection (schedules & to-dos)

Module/ Test Name	Input/ Pre-conditions	Steps	Expected Output	Actual Output	Pass/ Fail
Schedule overdue within grace	End passed < 24h	Observe	Still in active; not overdue yet	Remained in active list.	Pass
Schedule overdue after grace	End passed = 24h	Observe	Moves to Overdue automatically	Moved to Overdue.	Pass
To-do incomplete with pending subtasks	3 subtasks	Complete 2/3	To-do remains incomplete	Remained incomplete.	Pass

To-do auto complete by subtasks	3 subtasks	Complete 3/3	To-do marked Completed	Auto-marked Completed.	Pass
Manual completion override	Any subtasks state	Toggle "Done"	To-do marked Completed regardless	Marked Completed regardless of subtasks.	Pass

Table 4.1.6 Test Case – Completion Detection

4.1.7 Universal Search (Text, Transcripts, Files, URLs)

Module/ Test Name	Input/ Pre-conditions	Steps	Expected Output	Actual Output	Pass/ Fail
Search text comment keyword	Known keyword exists	Search	Matching comments returned; highlights; jump-to works	Correct comments highlighted.	Pass
Search voice transcript phrase	Phrase only in transcript	Search	Voice thread returned with transcript snippet	Transcript results returned from Firestore.	Pass
Search by attachment name	"report.pdf" uploaded	Search "report"	Attachment message returned	Attachment message returned in results.	Pass
Search by URL domain	URL contains "https"	Search URL	URL message returned	Message with URL returned.	Pass
Case-insensitive matching	Mixed case content	Search lower/upper	Identical results (case-insensitive)	Results identical for lower/upper case.	Pass
No-match behaviour	Random string	Search	"No results" state; input preserved	UI showed "No results found".	Pass
Large result set – pagination	>100 matches	Scroll results	Incremental load; smooth scrolling; no freezes	Loaded incrementally; smooth scrolling.	Pass

Table 4.1.7 Test Case – Universal Search

4.1.8 AI Assistant (Consent-Gated Actions)

Module/ Test Name	Input/ Pre-conditions	Steps	Expected Output	Actual Output	Pass/ Fail
Plan my day – generate plan only	Tasks & schedules exist	Tap "Plan my day"	Returns plan with time blocks; no writes without consent.	Generated schedule blocks based on tasks/schedules; did not auto-write without user confirmation.	Pass
Add to schedule – on confirm	Plan generated	Tap "Add to schedule"	Schedules created; success toast; Home	Created new events in schedules; push notification sent.	Pass

			shows new entries.		
Reschedule event/task	Target event exists	"Reschedule … to 15:00"	Event/Task updated to 15:00; notify collaborators if applicable.	STT sometimes couldn't detect the speech correctly and causing the action incomplete.	Fail
Delete event/task	Target event exists	Prompt deletion with the title	Event/Tasks deleted from Firestore.	Event/Tasks deleted from Firestore successfully	Pass
No free window today	Calendar fully busy	Tap "Plan my day"	Assistant explains conflict; proposes next slots/day	Assistant explained conflict; suggested next available slot.	Pass
List overdue tasks	Overdue exist	Command	Correct overdue list returned	Returned overdue tasks list correctly.	Pass

Table 4.1.8 Test Case – AI Assistant

4.1.9 Notifications (FCM)

Module/ Test Name	Input/ Pre-conditions	Steps	Expected Output	Actual Output	Pass/ Fail
Event alert – 1 hour before	Alert set 1h prior	Wait	Notification delivered exactly 60 min before	Push Notification doesn't receive at correct time.	Fail
Collaborator comment	Collaborator adds comment	Trigger	All collaborators receive in-app notification.	All collaborators received in-app notification.	Pass
Task completed – push	Mark task completed	Trigger	Collaborators notified of completion in app	In-app notification sent to collaborators.	Pass
Background restrictions handling	OS limits enabled	Wait for alert	Notification still delivered; or limitation documented	Push Notification doesn't send	Fail
Deep link on tap	Any received notification	Tap	App opens to correct screen/thread	App opened at correct task/schedule screen.	Pass

Table 4.1.9 Test Case – Notifications

4.1.10 Device Calendar Integration

Module/ Test Name	Input/ Pre-conditions	Steps	Expected Output	Actual Output	Pass/ Fail
Permission granted – merge events	User grants calendar access	Enable	Device events merged with Firestore schedules	Device calendar events merged into Home view.	Pass

Permission denied – graceful	User denies access	Deny prompt	Only Firestore schedules shown; no crash	Only Firestore schedules shown; no crash.	Pass
New device event – reflection	Create in device calendar	Refresh Home	Event appears in merged view	Reflected in merged view after refresh.	Pass
Runtime revoke – handling	Granted earlier, now revoked	Revoke in settings	App hides device events; no fatal error	Device events hidden; app continued normally.	Pass

Table 4.1.10 Test Case – Device Calendar Integration

4.1.11 Settings, Permissions & Profile

Module/ Test Name	Input/ Pre-conditions	Steps	Expected Output	Actual Output	Pass/ Fail
Update profile – email/phone	Valid values	Save	Firestore updated; UI reflects changes	Firestore user's doc updated; changes reflected in UI.	Pass
Notification channel behaviour	Channel silent → audible	Toggle in settings	App notifications adopt new channel behaviour	Audible applied correctly.	Pass
Microphone permission denied	Mic permission OFF	Attempt record	Clear prompt; recording blocked; no crash	Prompt shown; recording disabled.	Pass
Storage permission denied	Storage permission OFF	Attach file	Clear prompt; attach disabled; no crash	Prompt shown; file attachment disabled.	Pass

Table 4.1.11 Test Case – Settings, permission & Profile

4.1.12 Offline & Latency

Module/ Test Name	Input/ Pre-conditions	Steps	Expected Output	Actual Output	Pass/ Fail
Create to-do while offline	Disable network	Add to-do	Cached locally; UI indicates offline; syncs later	Cached locally; synced when online.	Pass
Post comment offline – queue	No network	Post text	Queued; posts when online; no duplicate	Queued locally; posted when online; no duplicate.	Pass

Table 4.1.12 Test Case – Offline & Latency

4.1.13 UI Micro-interactions & Utilities

Module/ Test Name	Input/ Pre-conditions	Steps	Expected Output	Actual Output	Pass/ Fail
Search result – jump & highlight	Have a match in long thread	Tap search hit	Scrolls to original message; row highlighted	Scrolled to message; highlighted in thread.	Pass

Long-press shortcuts – assistant	Home +, schedule +, to-do +	Long-press	Assistant voice/text dialog appears	Assistant dialog appeared (voice/text options).	Pass
Keyboard focus reliability	Any input bar	Tap input	Keyboard opens; no focus trap or autohide	Opened correctly; no focus trap.	Pass
Infinite list – pagination	Long conversation	Scroll	Next page loads; scroll position maintained	New page loaded smoothly; scroll position maintained.	Pass
External link navigation	Comment with URL	Tap link or files	Opens browser; back returns to app state	Opened in browser; back returned to app state.	Pass

Table 4.1.13 Test Case – UI Micro-interactions & Utilities

4.1.14 Failed Test Cases Analysis

Test Case: Event alert – 1 hour before

The first issue identified relates to scheduled event notifications, particularly alerts set to trigger one hour before a task or event. Despite correct configuration, some notifications were either delayed or not received at the expected time. This failure can be attributed to system-level restrictions on exact alarm scheduling in modern mobile operating systems, particularly under Doze mode or battery optimization settings. In addition, missing runtime permissions such as SCHEDULE_EXACT_ALARM (Android 12+) or incorrect time-zone handling may have contributed to inconsistencies. To mitigate this, exact alarm APIs such as setExactAndAllowWhileIdle() should be adopted, and all times should be stored in UTC with conversion to the device's time zone at scheduling. Furthermore, fallback mechanisms through Firebase Cloud Messaging (FCM) can be implemented to ensure timely delivery even when local alarms are suppressed.

Test Case: Background restrictions handling

The second challenge concerns push notification delivery in restricted background conditions. In scenarios where the device is in battery saver mode or has imposed background execution limits, push notifications were observed to fail. This suggests that the notification priority may not have been set to "high," or that app-level optimizations were blocked by the operating system's power-saving features. To address this, notifications must be configured as high priority within FCM, while notification channels should be created with high importance and lock-screen visibility enabled. Additionally, the system should be designed to re-register alarms and push tokens after device reboot or app updates to maintain resilience. Clear user

guidance for whitelisting the application from manufacturer-specific background restrictions could also reduce the likelihood of failed delivery.

Test Case: Reschedule event/task

A further issue was observed in the speech-to-text (STT) component, where the system occasionally failed to accurately detect speech, leading to incomplete transcription and preventing users from completing intended actions. This problem is heightened in multilingual contexts such as Malaysia, where inputs may shift dynamically between English, Bahasa Malaysia, and Chinese. Although the system currently uses bundled recognizers and fallback models, misinterpretation remains possible when low-confidence recognition outputs are accepted or when the input includes rapid code-switching. To improve robustness, explicit regional language models (e.g., en-MY, ms-MY, cmn-Hans-CN) should be prioritized, and confidence thresholds should be introduced to trigger retries or prompt users to clarify input.

Test Case: Record voice (EN/BM/CN)

An additional complication within STT transcription was the recurrent occurrence of Chinese speech being returned as Pinyin rather than Hanzi characters. This discrepancy reduces readability for native Chinese users and undermines the accuracy of stored transcripts. The underlying cause lies in the recognition model defaulting to ASCII outputs when Han character bundles are not strongly enforced, or when the detected confidence level for Chinese script is low. While the system already includes a “pinyin guard” retry with Han bundles, results remain inconsistent under noisy conditions or mixed-language inputs. To resolve this, the transcription pipeline should strictly enforce Han output for Chinese language codes, introduce post-processing filters to reject Pinyin-like results, and provide user-facing settings to lock transcription into a specific script when required. Such improvements would ensure more reliable transcription quality and enhance usability for multilingual teams.

4.2 Client Satisfaction Survey Analysis (10 Respondent)

Section A: About You & Your Work/ Study Habits

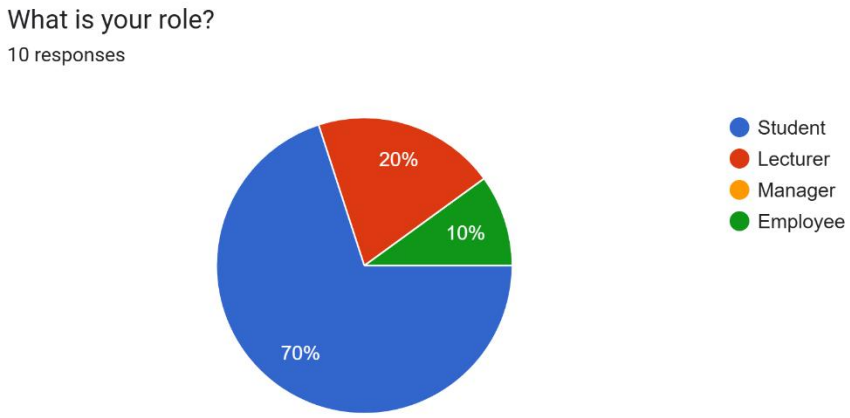

Figure 4.2.1 Section A Question 1

Figure 4.2.1 illustrates the distribution of respondents' roles. Out of ten participants, seven are students, two are lecturers, and one is an employee. This indicates that the majority of the respondents are students, although there is some representation from professional and academic staff, which allows for perspectives beyond a purely student population.

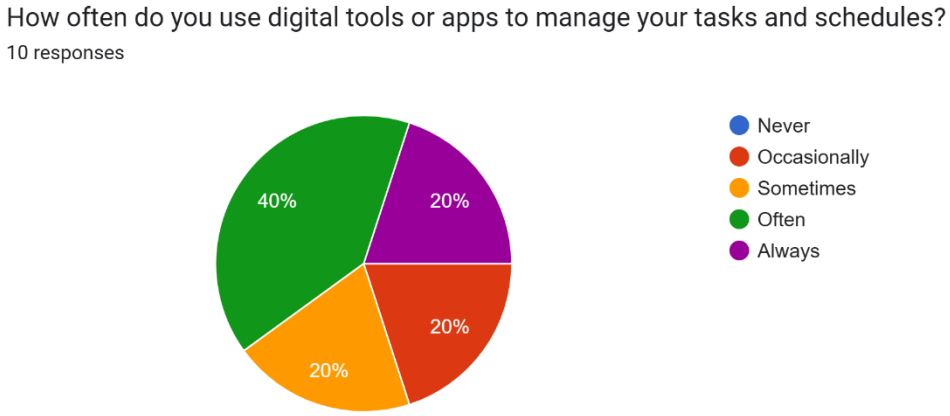

Figure 4.2.2 Section A Question 2

Figure 4.2.2 shows the frequency of using digital tools or applications to manage tasks and schedules. Four respondents reported often use such tools, two reported always using them, while the remaining four indicated occasional or sometimes usage. This suggests that while most participants are accustomed to digital task management, a portion still adopt these tools less consistently.

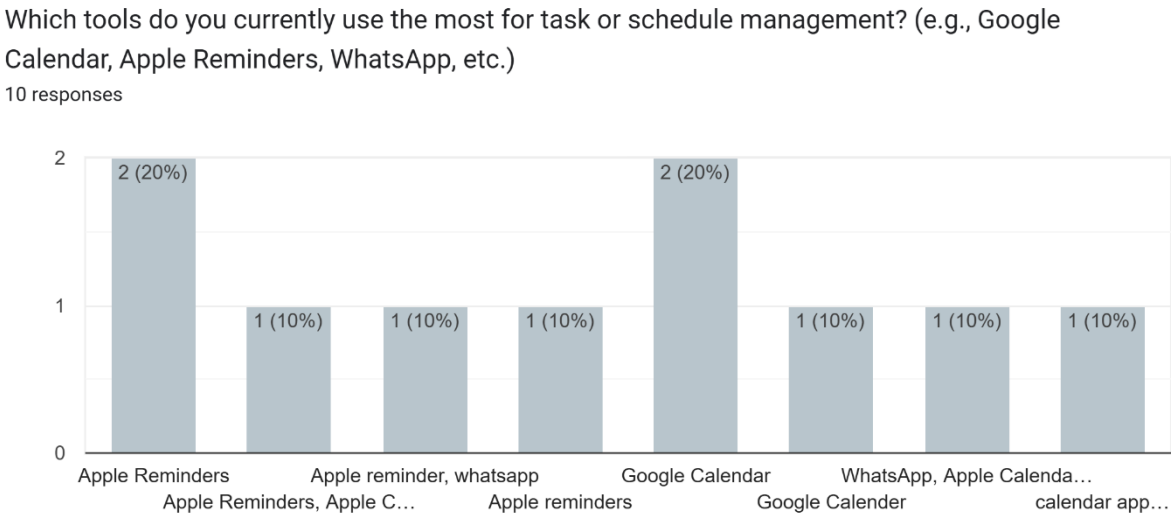

Figure 4.2.3 Section A Question 3

Figure 4.2.3 presents the tools most frequently used by respondents. Apple Reminders and Google Calendar are the most common, each selected by two respondents. The rest reported varied combinations such as WhatsApp, Apple Calendar, device calendar and other reminder applications. This demonstrates that respondents often rely on multiple applications to support their scheduling and task management needs.

Figure 4.2.4 Section A Question 4

Figure 4.2.4 depicts the average number of tasks or activities managed weekly. Half of the respondents reported managing between 5–10 tasks per week, four managed fewer than five, and one managed 11–20 tasks. This reflects a generally moderate workload, suggesting that organization remains important, even when the volume of tasks is not excessively high.

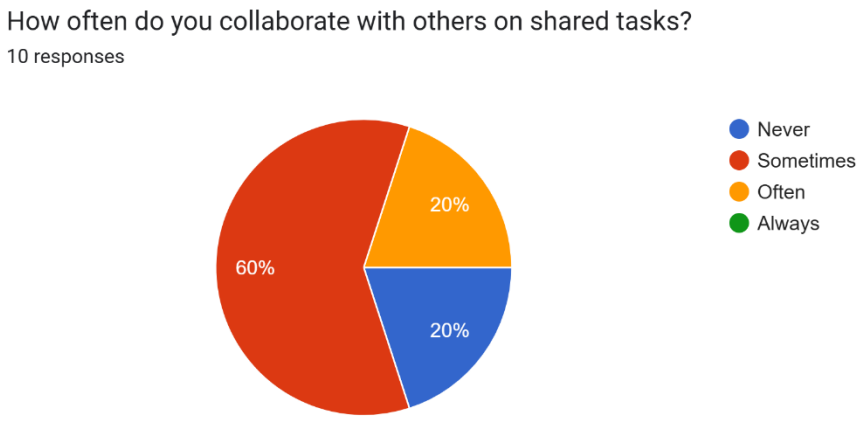

Figure 4.2.5 Section A Question 5

Figure 4.2.5 illustrates how frequently respondents collaborate with others on shared tasks. Six respondents indicated that they collaborate often, two reported always collaborating, and two reported sometimes collaborating. This highlights that collaboration is a significant part of most participants' task management experience.

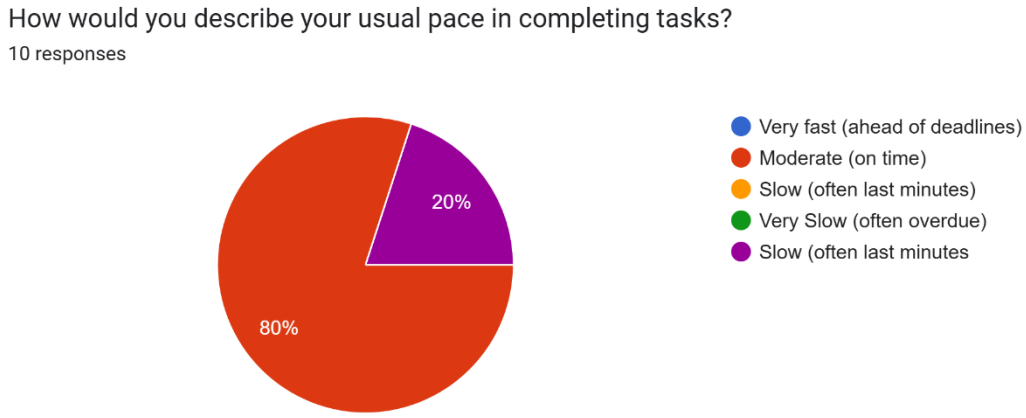

Figure 4.2.6 Section A Question 6

Figure 4.2.6 demonstrates the pace at which respondents usually complete their tasks. Eight respondents described their pace as moderate, meaning they complete tasks on time, while two respondents reported completing tasks at the last minute. This shows that punctual task completion is common, though procrastination still exists among some participants.

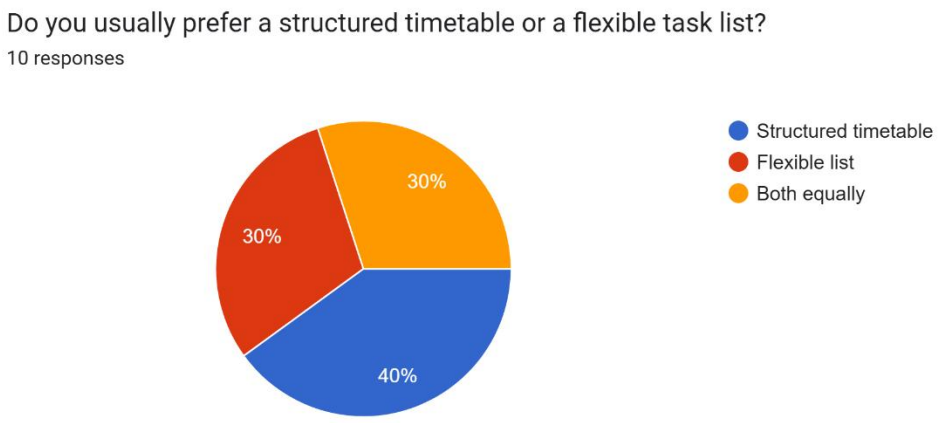

Figure 4.2.7 Section A Question 7

Figure 4.2.7 four respondents prefer a structured timetable, while three prefer a flexible task list and three value both equally. This distribution indicates a slight preference for structured scheduling, yet a substantial portion require flexibility, suggesting that both timetable-style planning and adaptable task lists should be supported in the application's design.

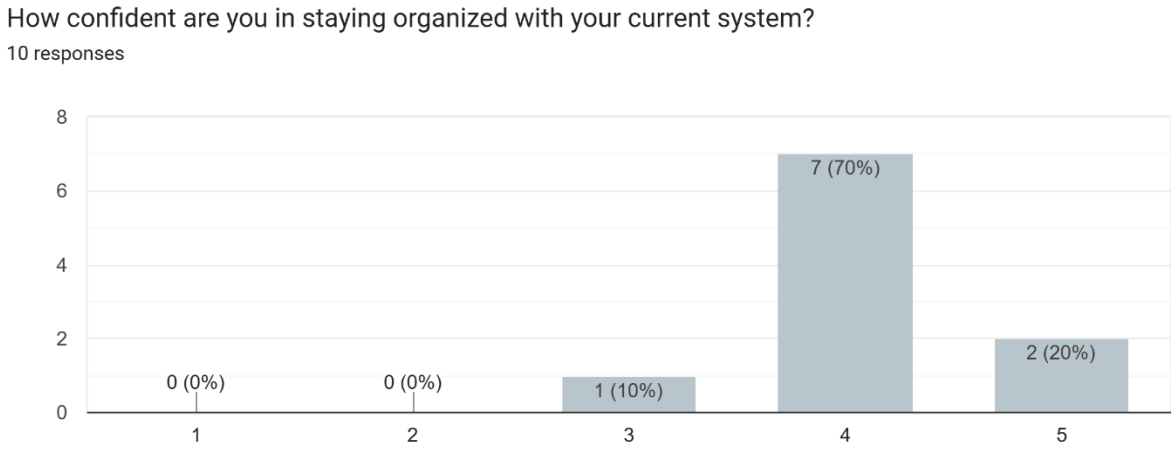

Figure 4.2.8 Section A Question 8

Figure 4.2.8 shows respondents' confidence levels in staying organized with their current system. Seven respondents rated themselves at level 4 out of 5, two gave the highest rating of 5, and one rated at level 3. This indicates that most respondents feel confident in their current organizational systems.

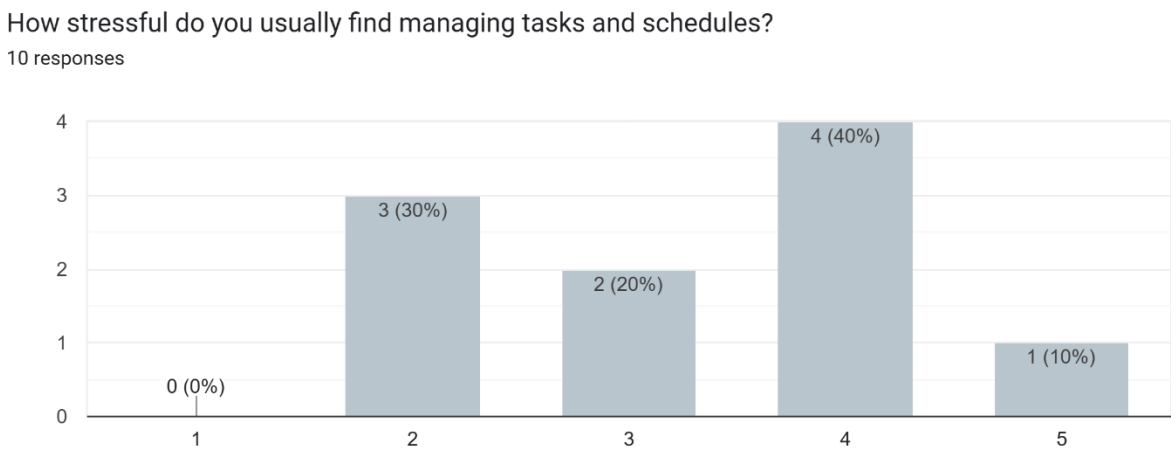

Figure 4.2.9 Section A Question 9

Figure 4.2.9 illustrates how stressful respondents find managing tasks and schedules. Four respondents rated their stress level at 4 (quite stressful), three rated it at 2 (low stress), two rated it at 3 (moderate), and one rated it at 5 (very stressful). These findings suggest that stress levels vary among respondents, with some coping effectively while others experience higher levels of difficulty.

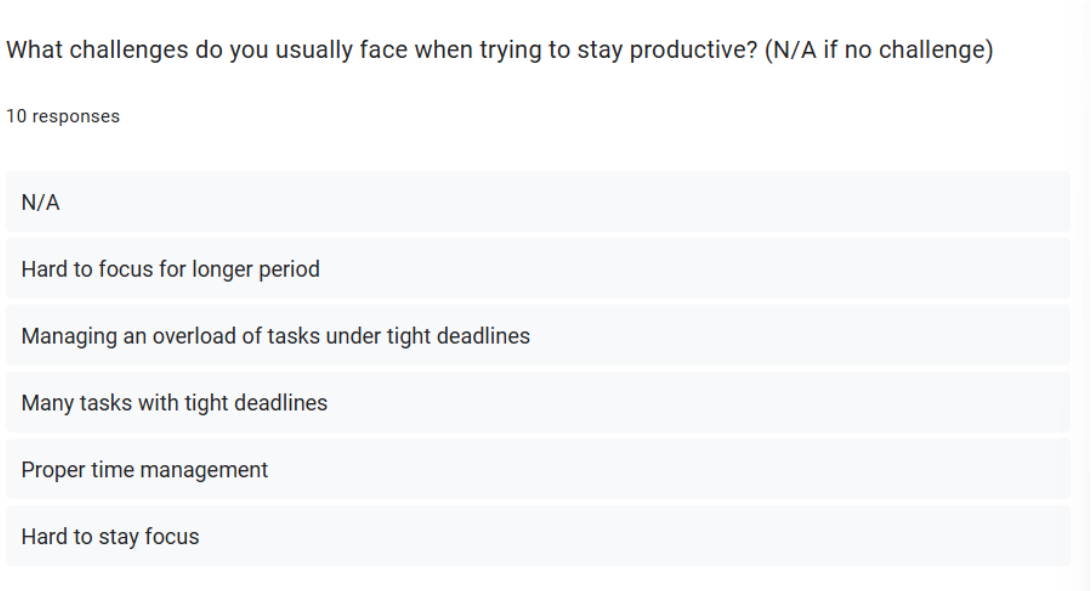

Figure 4.2.10 Section A Question 10

Figure 4.2.10 presents the reported challenges faced when staying productive. The challenges include difficulty focusing for long periods, managing an overload of tasks under tight deadlines, proper time management, and maintaining concentration. These responses highlight that productivity issues are primarily linked to workload pressure and personal discipline.

Section B: Comparing Experience with StudyMate vs Your Current Application

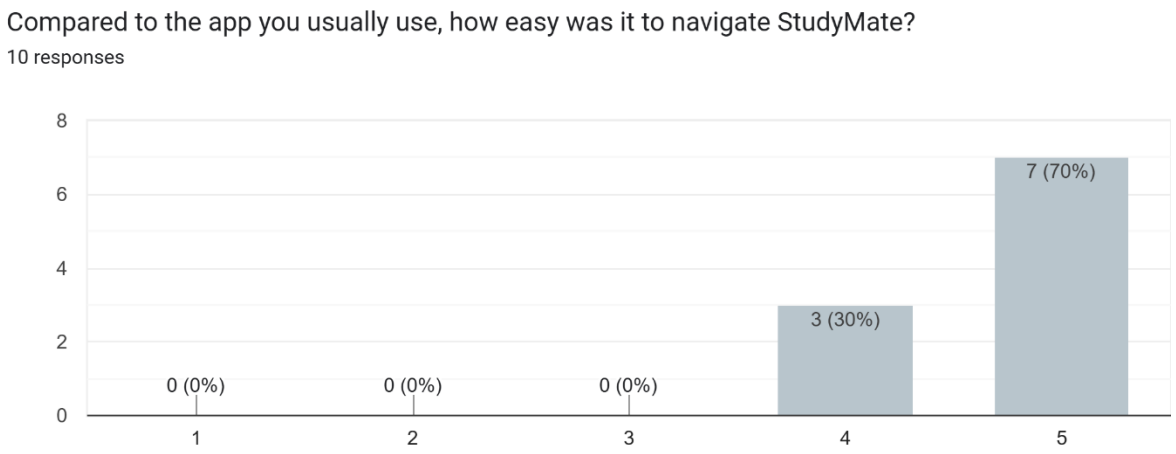

Figure 4.2.11 Section B Question 1

Figure 4.2.11 illustrates the ease of navigation when using StudyMate compared to respondents' usual applications. Seven respondents rated navigation as very easy (5), while three rated it as easy (4). None selected ratings from 1 to 3. This indicates a uniformly positive user experience in terms of ease of navigation.

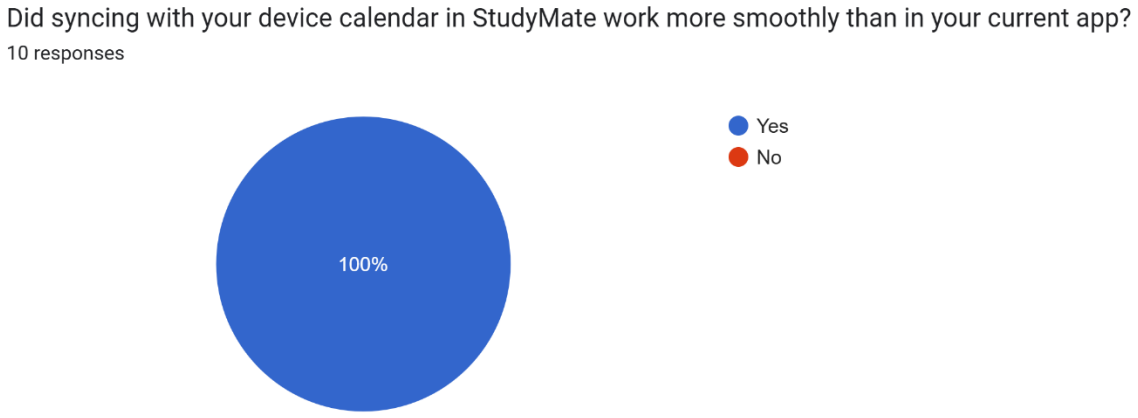

Figure 4.2.12 Section B Question 12

Figure 4.2.12 shows the experience of syncing StudyMate with device calendars compared to other applications. All ten respondents (100%) reported that syncing in StudyMate worked more smoothly. This demonstrates a strong consensus on the reliability and effectiveness of StudyMate's calendar integration.

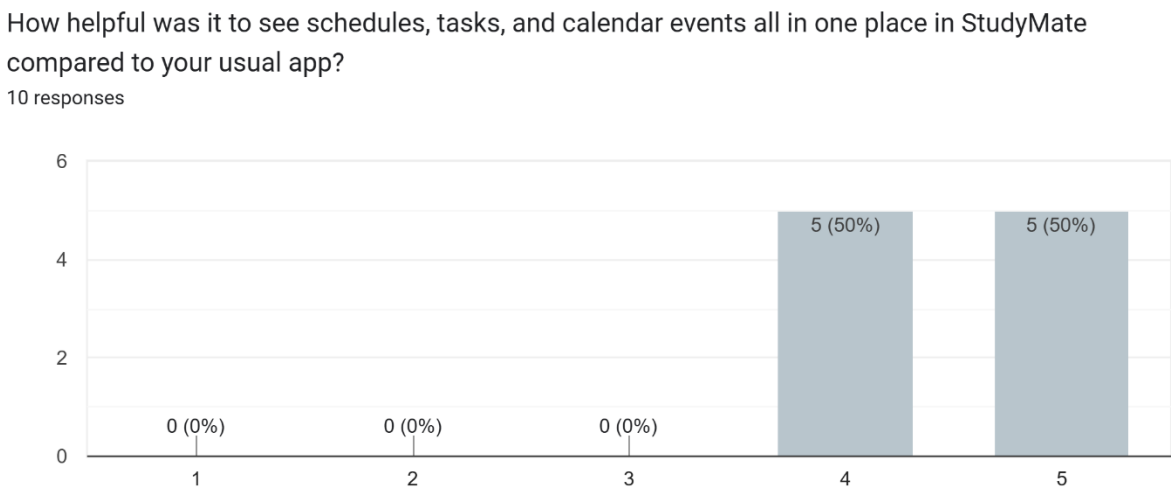

Figure 4.2.13 Section B Question 3

Figure 4.2.13 presents how helpful it was to view schedules, tasks, and calendar events in one place using StudyMate compared to other applications. Half of the respondents rated this feature as very helpful (5), while the other half rated it as helpful (4). This reflects a consistent view that integration of multiple functions in a single interface enhances efficiency.

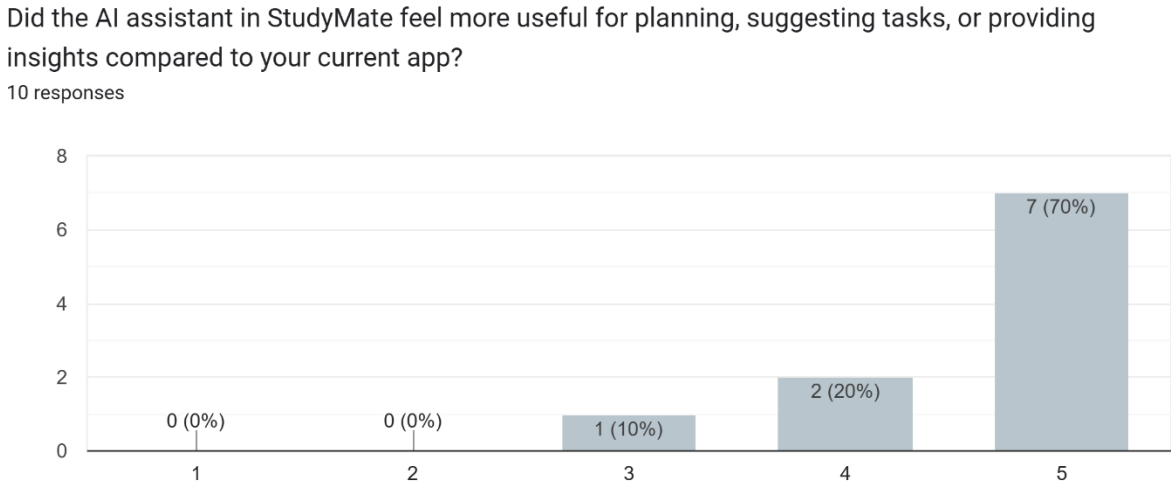

Figure 4.2.14 Section B Question 4

Figure 4.2.14 depicts respondents' views on the usefulness of the AI assistant for planning, suggesting tasks, and providing insights compared to their current applications. Seven respondents rated it as very useful (5), two rated it as useful (4), and one rated it as moderately useful (3). This indicates strong approval of the AI assistant, with only minimal reservations.

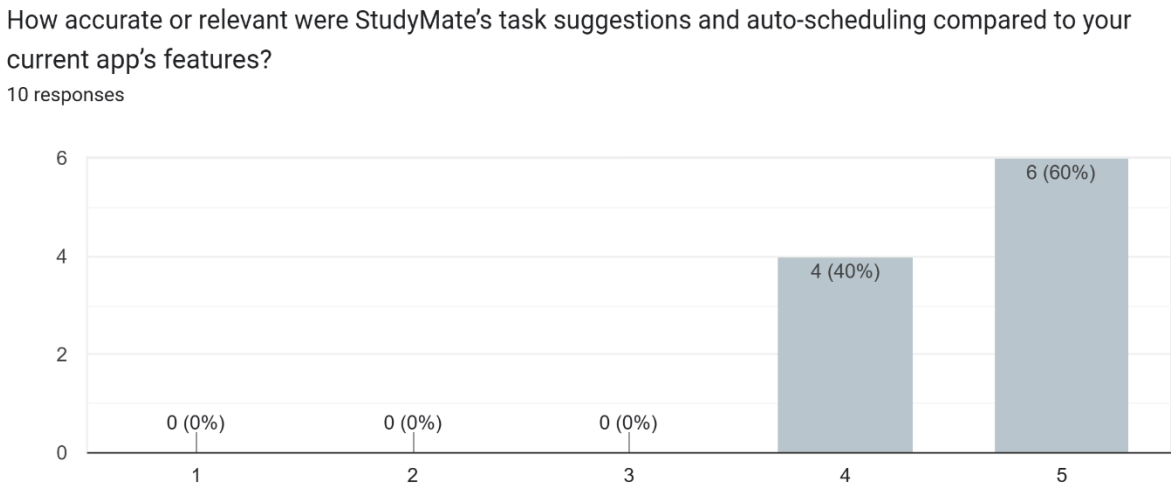

Figure 4.2.15 Section B Question 5

Figure 4.2.15 shows the accuracy and relevance of StudyMate's task suggestions and auto-scheduling compared to other applications. Six respondents rated the feature as very accurate (5), while four rated it as accurate (4). The results highlight a positive reception towards this functionality, with all respondents finding it at least accurate.

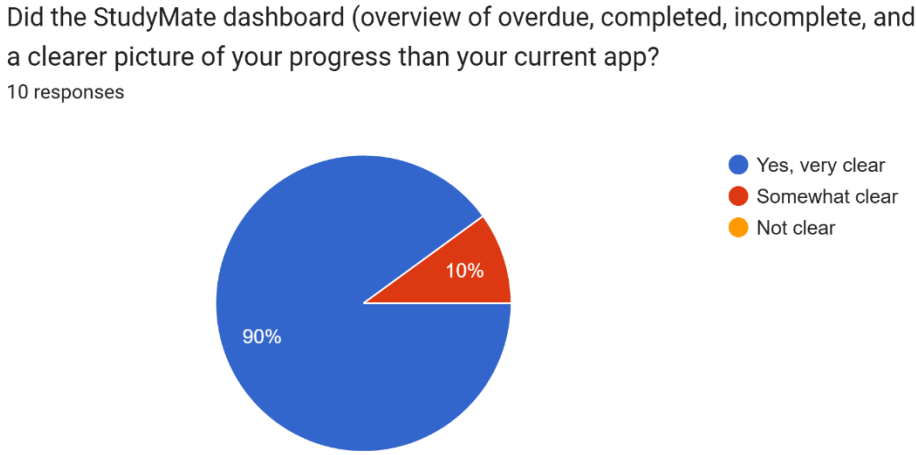

Figure 4.2.16 Section B Question 6

Figure 4.2.16 illustrates the clarity of the StudyMate dashboard in presenting overdue, completed, incomplete tasks, and progress charts compared to other applications. Nine respondents (90%) found the dashboard very clear, while one respondent (10%) found it somewhat clear. This suggests that the dashboard is effective in visualising task progress and overall productivity.

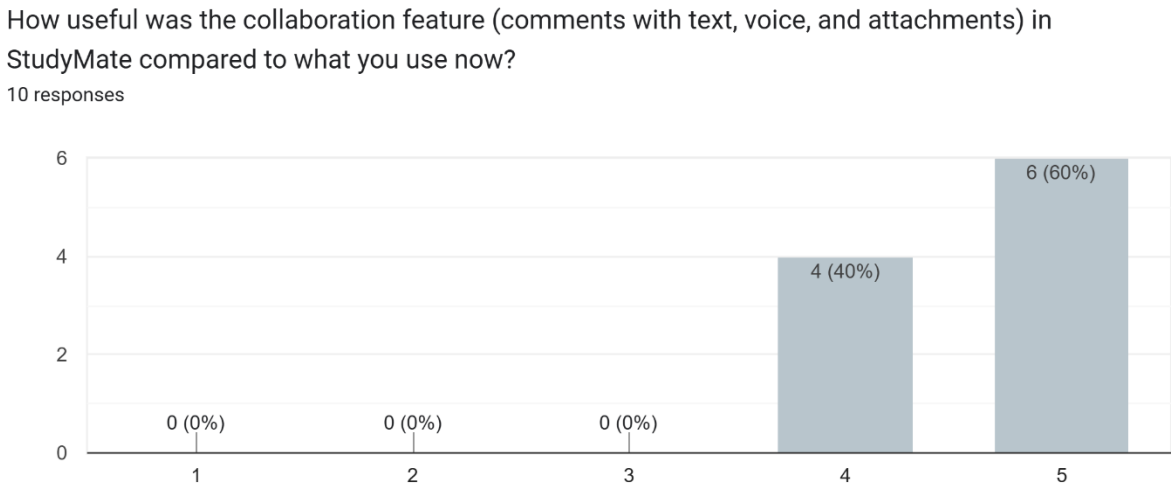

Figure 4.2.17 Section B Question 7

Figure 4.2.17 demonstrates the usefulness of the collaboration feature, including text, voice, and attachments, compared to current applications. Six respondents rated it as very useful (5), and four rated it as useful (4). This reflects widespread satisfaction with the collaboration tools available in StudyMate.

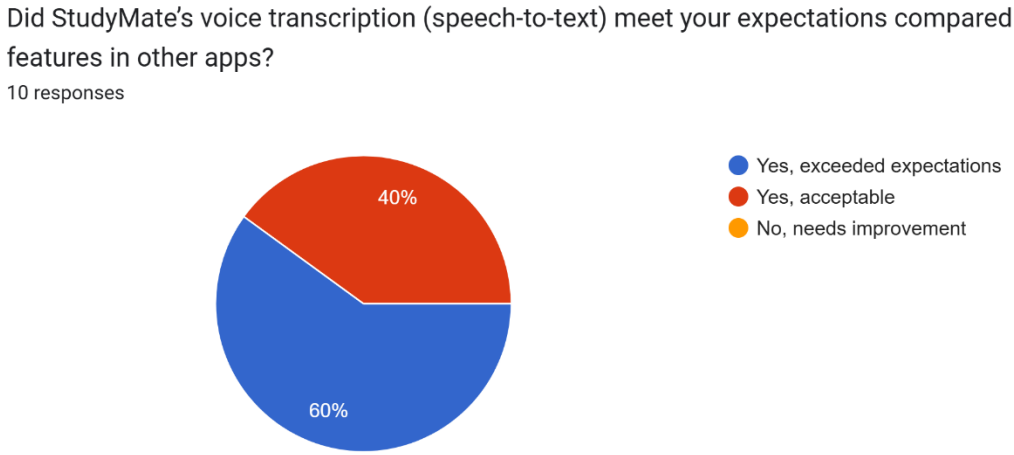

Figure 4.2.18 Section B Question 8

Figure 4.2.18 presents user feedback on the voice transcription feature compared to similar features in other applications. Six respondents (60%) stated it exceeded expectations, while four respondents (40%) found it acceptable. None reported it as needing improvement, suggesting that the transcription system meets or surpasses expectations.

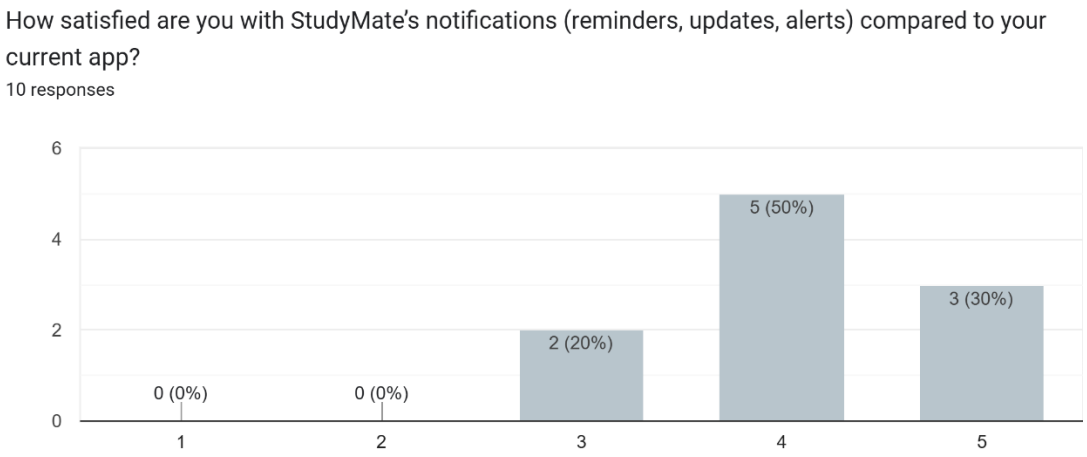

Figure 4.2.19 Section B Question 9

Figure 4.2.19 shows satisfaction with StudyMate's notifications compared to current applications. Five respondents rated satisfaction as 4, three rated it as 5, and two rated it as 3. Overall, the majority of respondents expressed satisfaction, though a small proportion indicated room for further refinement.

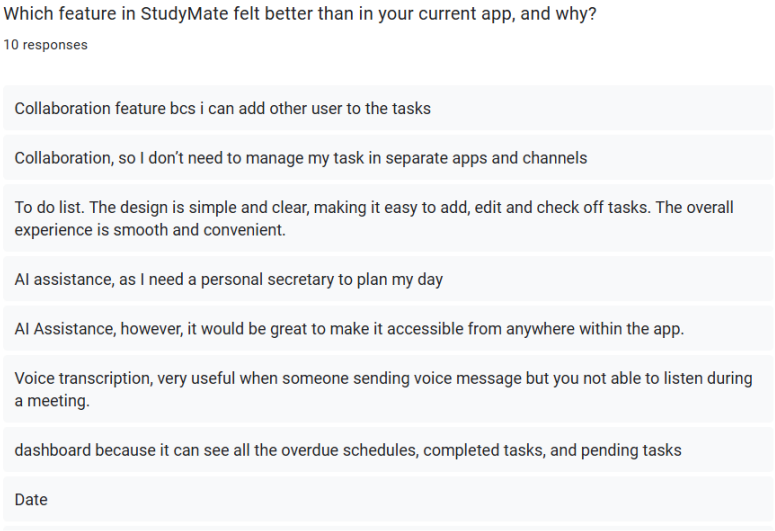

Figure 4.2.20 Section B Question 10

Figure 4.2.20 highlights open-ended feedback on which StudyMate feature was considered better than respondents' current applications. The most frequently cited advantages were the collaboration feature, which consolidates teamwork into one platform, the AI assistant for planning and personal organisation, and the dashboard for clear task tracking. Other mentions included the simplicity of the to-do list design and the usefulness of voice transcription. These responses reinforce the quantitative findings that StudyMate provides clear advantages in integration, collaboration, and intelligent assistance.

Section C: Productivity & Future Needs Compared to Current Applications

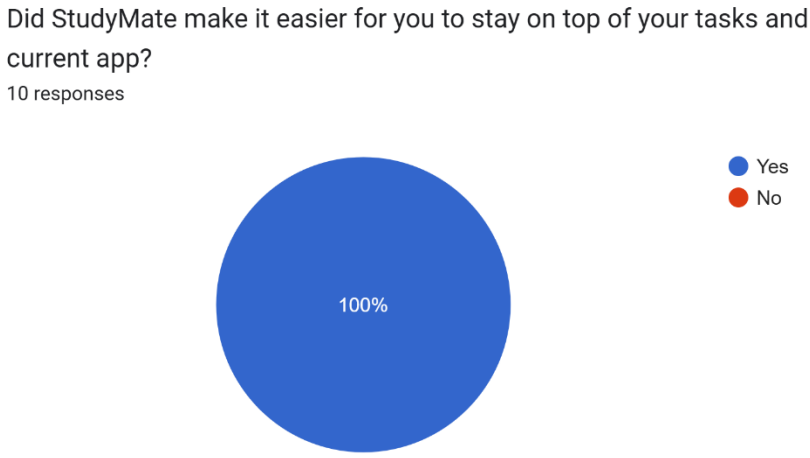

Figure 4.2.21 Section C Question 1

Figure 4.2.21 shows whether StudyMate made it easier for respondents to stay on top of their tasks and activities compared to their current application. All ten respondents (100%) answered "Yes," indicating unanimous agreement that StudyMate provides greater support in managing tasks effectively.

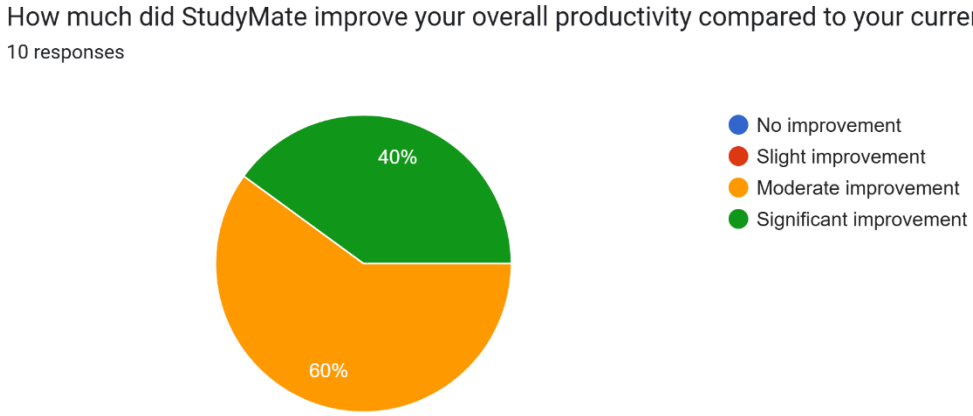

Figure 4.2.22 Section C Question 2

Figure 4.2.22 illustrates the extent to which StudyMate improved overall productivity. Six respondents (60%) reported moderate improvement, while four (40%) reported significant improvement. None selected "no improvement" or "slight improvement." This demonstrates that StudyMate had a clear positive impact on productivity for all participants.

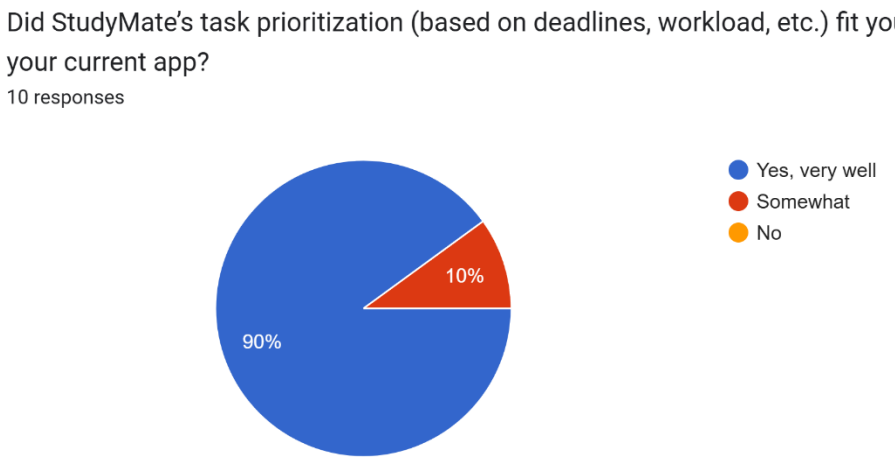

Figure 4.2.23 Section C Question 3

Figure 4.2.23 depicts the effectiveness of StudyMate's task prioritisation compared to respondents' existing applications. Nine respondents (90%) stated that prioritisation fit their needs very well, while one respondent (10%) selected "somewhat." This reflects a high level of satisfaction with the weighted prioritisation approach integrated into the system.

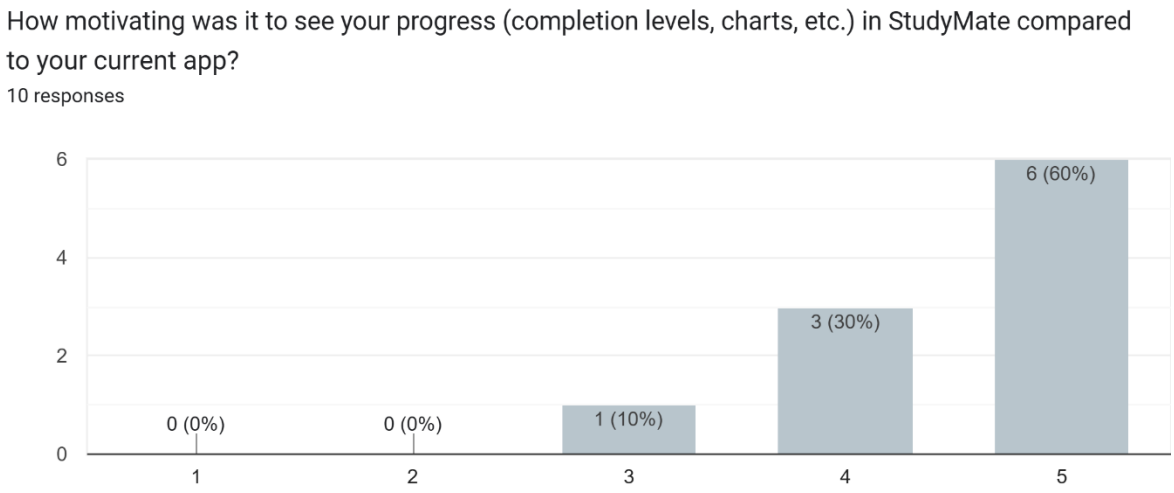

Figure 4.2.24 Section C Question 4

Figure 4.2.24 presents respondents' perceptions of how motivating it was to view progress through completion levels and charts. Six respondents (60%) rated motivation as very high (5), three (30%) rated it as high (4), and one (10%) provided a moderate rating (3). No participants rated motivation at low levels. These findings suggest that StudyMate's visual progress-tracking tools serve as strong motivators.

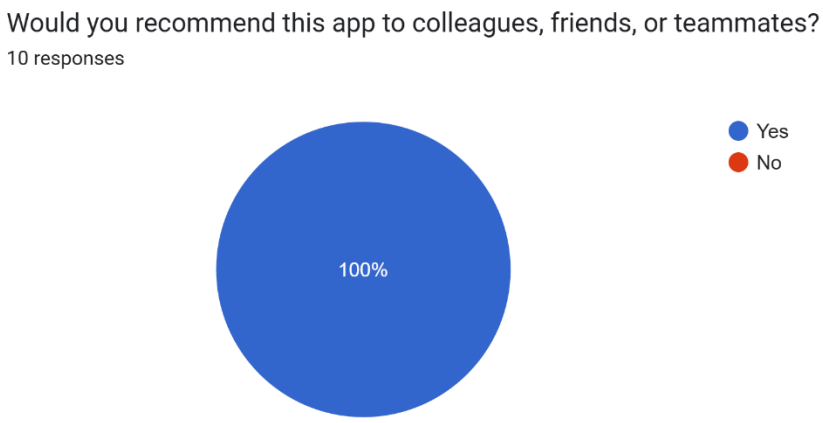

Figure 4.2.25 Section C Question 5

Figure 4.2.25 shows whether respondents would recommend StudyMate to colleagues, friends, or teammates. All ten participants (100%) answered "Yes," reflecting unanimous endorsement of the application.

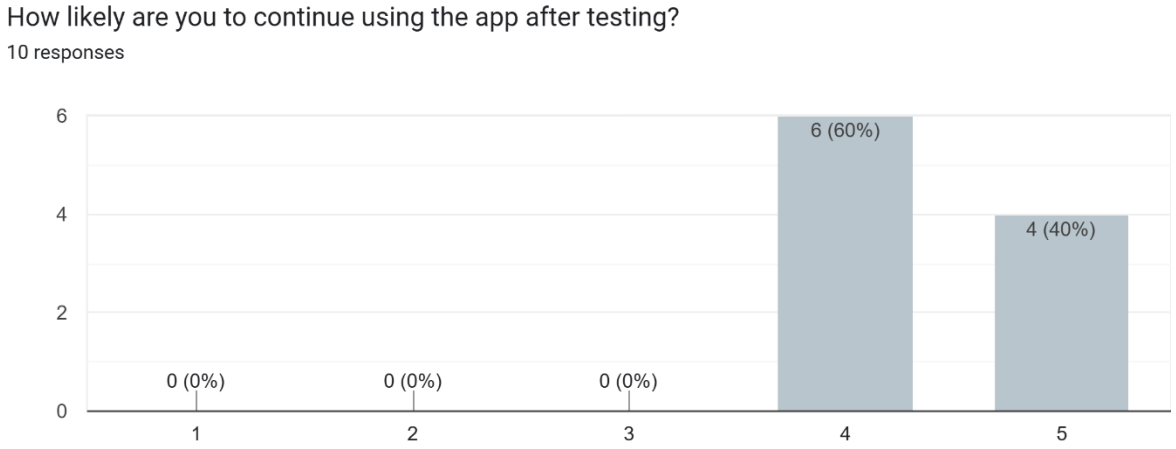

Figure 4.2.26 Section C Question 6

Figure 4.2.26 demonstrates respondents' likelihood of continuing to use StudyMate after testing. Six respondents (60%) rated this likelihood at 4, while four (40%) rated it at 5. None selected values lower than 4. This highlights high retention potential for the application among its users.

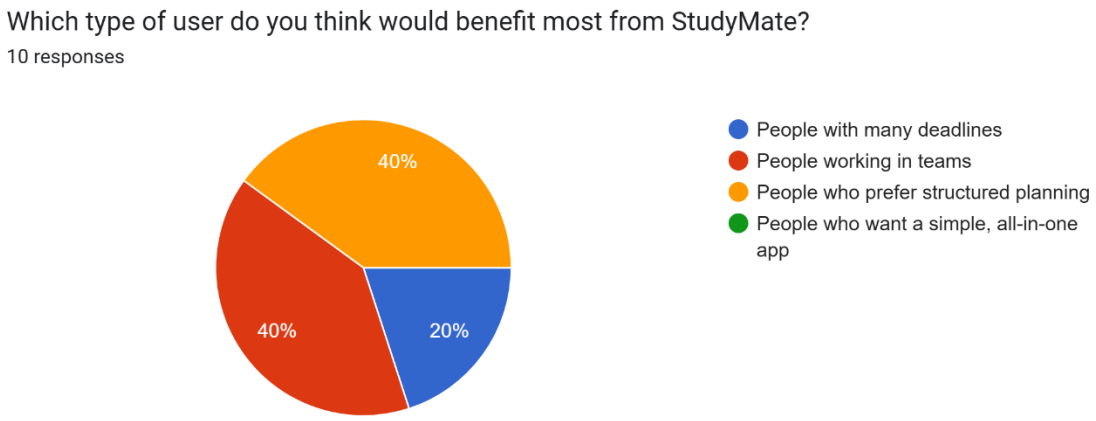

Figure 4.2.27 Section C Question 7

Figure 4.2.27 identifies the type of user respondents believe would benefit most from StudyMate. Four respondents (40%) chose people working in teams, another four (40%) selected people who prefer structured planning, and two (20%) selected people with many deadlines. This suggests that StudyMate is seen as particularly valuable for collaborative and structured task management scenarios.

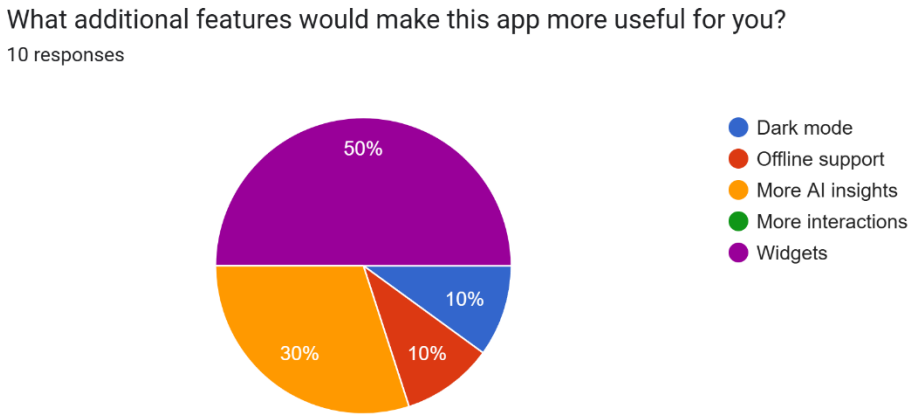

Figure 4.2.28 Section C Question 8

Figure 4.2.28 presents suggestions for additional features that would make StudyMate more useful. Half of the respondents (50%) selected widgets, three respondents (30%) indicated more AI insights, and one respondent each (10%) suggested dark mode and offline support. These responses highlight demand for enhanced customisation and expanded AI-driven functionalities.

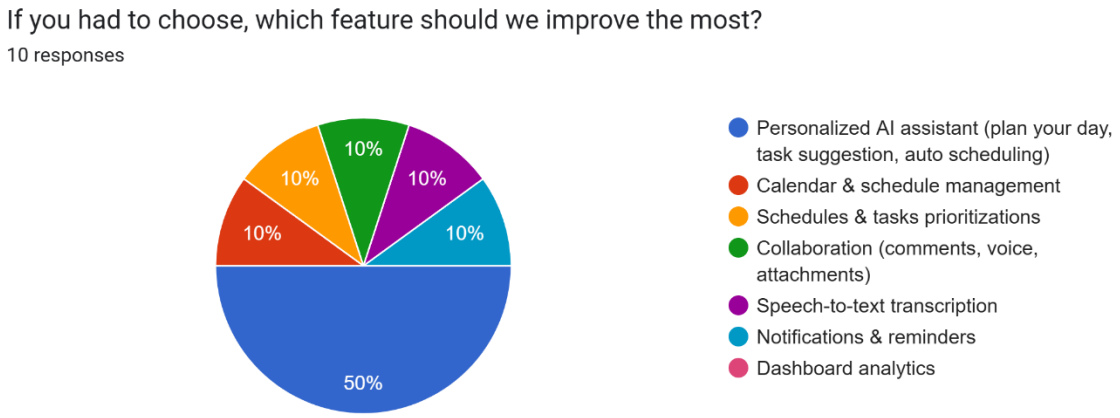

Figure 4.2.29 Section C Question 9

Figure 4.2.29 illustrates which feature respondents believe should be prioritised for improvement. Half of the respondents (50%) selected the personalised AI assistant, while the other five responses (10% each) were spread across calendar management, task prioritisation, collaboration, speech-to-text transcription, notifications, and dashboard analytics. This finding indicates that further development of the AI assistant is the highest priority for users.

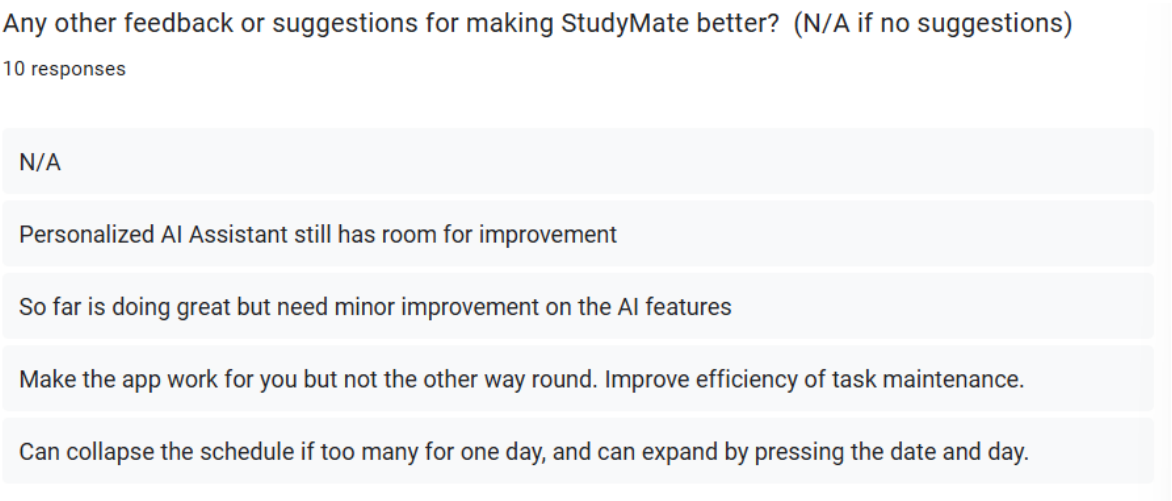

Figure 4.2.30 Section C Question 10

Figure 4.2.30 presents open-ended feedback from respondents. Suggestions included improving the AI assistant's capabilities, enhancing task maintenance efficiency, and providing collapsible schedules for days with many activities. These responses confirm that while StudyMate performs strongly overall, users desire refinements to increase efficiency and flexibility.

4.3 Results and Benchmark

The evaluation results demonstrate that StudyMate not only meets user expectations but also outperforms existing applications in several critical areas of task and schedule management. Respondents indicated that their current tools included Apple Reminders, Google Calendar, WhatsApp, and similar applications, which often require users to combine multiple platforms to achieve complete functionality. By contrast, StudyMate consolidates these features into a single, integrated system.

In terms of usability, navigation and calendar synchronisation were rated superior in StudyMate compared to existing tools. All respondents confirmed that syncing with device calendars worked more smoothly, highlighting a key strength in integration that traditional applications lack. This finding positions StudyMate as more reliable in maintaining consistency between personal devices and application data.

The inclusion of an AI assistant marks a significant benchmark distinction. While current applications provide basic scheduling or reminders. StudyMate's assistant supports planning, task suggestions, and progress insights. Respondents rated these features more positively than those available in their current apps, particularly in terms of relevance and accuracy of task recommendations. This demonstrates the added value of AI-driven functionality compared to traditional reminder-based systems.

Collaboration was another area where StudyMate surpassed existing benchmarks. Respondents highlighted the ability to integrate comments, voice messages, and attachments within tasks as more convenient than relying on external communication platforms such as WhatsApp. The consolidation of communication and task management within a single interface represents a key improvement in collaborative productivity.

Dashboard analytics provided a further advantage, with StudyMate offering visualisations of overdue, completed, and incomplete tasks as well as breakdowns by priority. This level of progress tracking was rated as clearer and more motivating than what respondents typically experienced in their current applications, where such integrated overviews are often absent or fragmented.

From a productivity standpoint, all respondents agreed that StudyMate made it easier to stay on top of their tasks and activities compared to their current applications. Moderate to significant improvements in productivity were reported, confirming that StudyMate delivers

measurable benefits. The prioritisation algorithm was also rated more effective than the approaches used by existing tools, suggesting that dynamic weighting based on deadlines, workload, and priority levels provides a superior method of task ranking.

Finally, the unanimous willingness to recommend StudyMate and the strong likelihood of continued usage reflect its competitiveness in the wider productivity application space. Users particularly valued the personalised AI assistant and expressed interest in further enhancements such as widgets and expanded AI insights. These findings highlight both the present advantages of StudyMate and the directions for future refinement to maintain its competitive edge.

In summary, benchmarking against current applications confirms that StudyMate provides a more comprehensive, integrated, and intelligent solution. It exceeds the capabilities of traditional task and calendar management tools by combining AI-driven features, collaboration, and analytics into a single platform, thereby addressing gaps that users currently experience with fragmented application use.

4.4 Objectives Evaluation

The three objectives of this project were successfully achieved by providing an integrated and intelligent productivity solution. First, task organization and prioritization were enhanced through the Dynamic Weighted Task Prioritization Algorithm, which dynamically sorted tasks based on deadlines, workloads, and urgency while offering real-time visualizations such as progress bars and charts. This ensured that users could allocate their time efficiently and avoid overload. Second, real-time collaboration was significantly improved with unified messaging that supported both text and voice, automatic transcription for accessibility, attachment and URL integration, and a smart search function capable of scanning across all communication formats. These features minimized fragmentation and centralized group interactions, promoting transparent and efficient teamwork. Third, the Smart AI Assistant served as a virtual secretary by providing personalized daily planning, proactive task suggestions, reminders, and project oversight through dashboards and timeline views. The assistant's ability to interact naturally through text or speech further increased usability and convenience. Together, these features enabled users to maintain clear visibility of their responsibilities, collaborate effectively, and receive actionable guidance to remain organized and productive. Ultimately, the objectives collectively ensured that StudyMate surpasses traditional task management tools by combining intelligent prioritization, seamless collaboration, and AI-driven support.

4.5 Concluding Remark

To sum up, this project has successfully delivered a smart mobile task management application that addresses the common challenges of disorganized task handling, fragmented collaboration, and limited personalization found in existing tools. By integrating a dynamic prioritization algorithm, unified collaboration features, and a smart AI assistant, the system provides a comprehensive solution that not only improves individual productivity but also strengthens teamwork efficiency. The evaluation results confirmed that users experienced greater clarity, motivation, and convenience when managing their schedules and tasks with the application compared to their current tools. Furthermore, the positive reception and unanimous willingness to recommend the system highlight its practical value and potential for real-world adoption. While future improvements such as enhanced AI insights, widget support, and further customization were suggested, the objectives of the project have been met effectively.

Overall, this project demonstrates that integrating intelligent automation, collaborative functionality, and user-centered design can significantly enhance task management, ultimately enabling users to manage their responsibilities more efficiently and with reduced stress. Minor technical challenges such as occasional Gemini API overload and slower transcription speeds in multilingual contexts were addressed through retry handling and optimized language settings, ensuring that these limitations did not compromise the overall performance or user experience. Looking ahead, the adoption of fallback AI models and the integration of offline or on-device speech-to-text could further improve system resilience, guaranteeing reliable assistance even during network congestion or service disruptions.

CHAPTER 5
Conclusion

This project has successfully delivered a mobile-based task management system, StudyMate, designed to improve productivity and efficiency for students, educators, and professionals alike. By addressing the recurring challenges of ineffective task prioritization, fragmented collaboration, and limited visibility into progress, the system integrates three key innovations including a Dynamic Weighted Task Prioritization Algorithm, a seamless real-time collaboration framework, and a Smart AI Assistant for personalized planning and oversight. Together, these features provide an integrated platform that consolidates scheduling, task management, collaboration, and intelligent assistance into a single application.

The evaluation through black box testing and client satisfaction surveys confirmed that the system achieved its objectives. Users reported improved clarity in task organization, smoother collaboration with centralized communication tools, and greater motivation through progress tracking dashboards and visual aids. Benchmarking further demonstrated that StudyMate surpasses widely used applications such as Apple Reminders, Google Calendar, Microsoft To-Do, Todoist, TickTick, and Trello by offering dynamic prioritization, unified task and schedule integration, AI-driven recommendations, and advanced collaboration features within one platform.

Although challenges were encountered during development including Flutter framework mastery, storage limitations, and notification reliability these were effectively managed, and the system was successfully completed with scalable integration of Firebase and Google Cloud services. Feedback from respondents also highlighted potential enhancements, including widget support, additional AI insights, and further personalization, which provide valuable direction for future development. Furthermore, expanding cross-platform integration with tools such as Microsoft Outlook, Slack, and Google Workspace would allow seamless adoption in broader academic and professional contexts.

In conclusion, StudyMate demonstrates that the integration of intelligent automation, collaboration, and personalized assistance can significantly enhance productivity and reduce stress across academic and professional environments. The project has met its stated objectives and delivered a functional, user-centered, and intelligent task management solution that holds strong potential for real-world adoption and continued refinement.

REFERENCES

[1] D. Ariely and K. Wertenbroch, "Procrastination, Deadlines, and Performance: Self-Control by Precommitment," *Psychological Science*, vol. 13, no. 3, pp. 219–224, May 2002, https://doi.org/10.1111/1467-9280.00441.

[2] "Organize reminders on your iPhone or iPad," *Apple Support*. https://support.apple.com/en-us/119953#:~:text=Time%3A%20Add%20reminders%20to%20the

[3] "Introducing List Sharing and Steps in Microsoft To-Do," *TECHCOMMUNITY.MICROSOFT.COM*. https://techcommunity.microsoft.com/t5/microsoft-to-do-blog/introducing-list-sharing-and-steps-in-microsoft-to-do/ba-p/200109

[4] "Features," *Todoist*. https://todoist.com/features

[5] S. K. Nayak, S. K. Padhy, and S. P. Panigrahi, "A novel algorithm for dynamic task scheduling," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 709–717, May 2012, doi: https://doi.org/10.1016/j.future.2011.12.001.

[6] M. Spuri and G. Buttazzo, "Scheduling aperiodic tasks in dynamic priority systems," *Real-Time Systems*, vol. 10, no. 2, pp. 179–210, Mar. 1996, doi: https://doi.org/10.1007/bf00360340.

[7] R. W. White, Ahmed Hassan Awadallah, and R. Sim, "Task completion detection: A study in the context of intelligent systems," *In Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 405–414, Jul. 2019, doi: https://doi.org/10.1145/3331184.3331187.

[8] Nielsen, J. (2012). *Usability 101: Introduction to Usability.* Nielsen Norman Group. https://www.nngroup.com/articles/usability-101-introduction-to-usability/

[9] Interaction Design Foundation (IDF). (2023). *Prototyping: Learn Eight Common Methods and Best Practices.* https://www.interaction-design.org/literature/topics/prototyping

[10] S. Davis and P. Mermelstein, "Comparison of Parametric Representations for Monosyllabic Word Recognition," IEEE Trans. Acoustics, Speech, and Signal Processing, vol. 28, no. 4, pp. 357–366, 1980.

[11] L. Rabiner and R. Schafer, Theory and Applications of Digital Speech Processing, Pearson, 2010.

[12] X. Anguera, “Acoustic Feature Extraction for Automatic Speech Recognition,” in Encyclopedia of Speech and Language Processing, 2010.

[13] K. Rao, H. Sak, and R. Prabhavalkar, “Exploring architectures for streaming end-to-end speech recognition with RNN-Transducer,” ASRU, 2017.

[14] A. Gulati et al., “Conformer: Convolution-augmented Transformer for speech recognition,” Interspeech, 2020.

[15] S. Toshniwal et al., “A comparison of techniques for language model integration in encoder-decoder speech recognition,” SLT, 2018.

[16] J. Sohn, N. S. Kim, and W. Sung, “A statistical model-based voice activity detection,” IEEE SPL, 1999.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is All You Need,” *Advances in Neural Information Processing Systems (NIPS)*, pp. 5998–6008, 2017.

[18] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. V. Le, G. Hinton, and J. Dean, “Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer,” *arXiv preprint arXiv:1701.06538*, 2017.

[19] OpenAI, “GPT-4 Technical Report,” arXiv preprint arXiv:2303.08774, 2023.

[20] Anthropic, “Claude 3 Model Card,” Anthropic Documentation, 2024. [Online]. Available: https://www.anthropic.com

[21] Google DeepMind, “Gemini: A Family of Highly Capable Multimodal Models,” Google DeepMind Blog, Dec. 2023.

[22] Google, “Cloud Generative AI API Documentation,” Google Cloud, 2024. [Online]. Available: https://cloud.google.com/generative-ai

[23] Google Cloud, “Speech-to-Text v2 Documentation,” *Google Cloud*, 2024. [Online]. Available: https://cloud.google.com/speech-to-text

[24] M. Jeffress, “speech_to_text: A Flutter Plugin,” *pub.dev*, 2024. [Online]. Available: https://pub.dev/packages/speech_to_text

[25] A. Radford *et al.*, “Robust Speech Recognition via Large-Scale Weak Supervision,” *arXiv preprint arXiv:2212.04356*, 2022.

[26] A. Kochetov *et al.*, “Vosk Speech Recognition Toolkit,” *alphacephei.com*, 2020. [Online]. Available: https://alphacephei.com/vosk

[27] Microsoft, “Azure Speech to Text Documentation,” *Microsoft Azure*, 2024. [Online]. Available: https://azure.microsoft.com/en-us/products/ai-services/speech-to-text

[28] Amazon, “Amazon Transcribe Documentation,” *AWS*, 2024. [Online]. Available: https://aws.amazon.com/transcribe

[29] TickTick, “TickTick,” *ticktick.com*. https://ticktick.com/?language=en_US

[30] Trello, “Trello Tour,” *Trello.com*, 2019. https://trello.com/tour

[31] Apple Inc., “Limits for iCloud Calendar and Reminders,” *Apple Support*, 2024. https://support.apple.com/en-us/103188

[32] Microsoft Corporation, “Can’t change order of tasks on ‘My Day’ if they are recurring,” *Microsoft Community*, 2023. https://answers.microsoft.com/en-us/msoffice/forum/all/cant-change-order-of-tasks-on-my-day-if-they-are/0f43b74d-decd-4e66-a675-c12c5f4d0e54

[33] Microsoft Corporation, “Reordering To-Do does not save across devices,” *Microsoft Community*, 2023. https://answers.microsoft.com/en-us/msoffice/forum/all/reordering-to-do-does-not-save-across-devices/8dc716b1-1a6a-4191-8cc2-eaa95a75248b.

[34] Microsoft Corporation, “Custom order in To-Do not reflected in Outlook task,” *Microsoft Community*, 2023. https://answers.microsoft.com/en-us/msoffice/forum/all/custom-order-in-todo-not-reflected-in-outlook-task/444ecbfd-0205-4d2b-ad65-180234f1225e

[35] H. Singh, “Todoist vs Apple Reminders: Which to-do app is better for you?” *Hulry.com*, 2023. https://hulry.com/todoist-vs-apple-reminders

[36] Microsoft Corporation, “Microsoft To-Do sorting by due date is not working properly,” *Microsoft Community*, 2023. https://answers.microsoft.com/en-us/msoffice/forum/all/microsoft-to-do-sorting-by-due-date-is-not/b4a79bf5-f8eb-4af0-b8a8-09df605285d6

[37] R. S. Pressman, *Software Engineering: A Practitioner’s Approach*, 8th ed. New York, NY, USA: McGraw-Hill Education, 2015. ISBN: 978-0078022128.

[38] I. Sommerville, *Software Engineering*, 10th ed. Harlow, U.K.: Pearson Education, 2016. ISBN: 978-0133943030.

[39] Nielsen, J. (2012). *Usability 101: Introduction to Usability.* Nielsen Norman Group. https://www.nngroup.com/articles/usability-101-introduction-to-usability/

[40] Interaction Design Foundation (IDF). (2023). *Prototyping: Learn Eight Common Methods and Best Practices.* https://www.interaction-design.org/literature/topics/prototyping

APPENDIX A

Poster

APPENDIX B

Operating Manual

B-1.1 Login

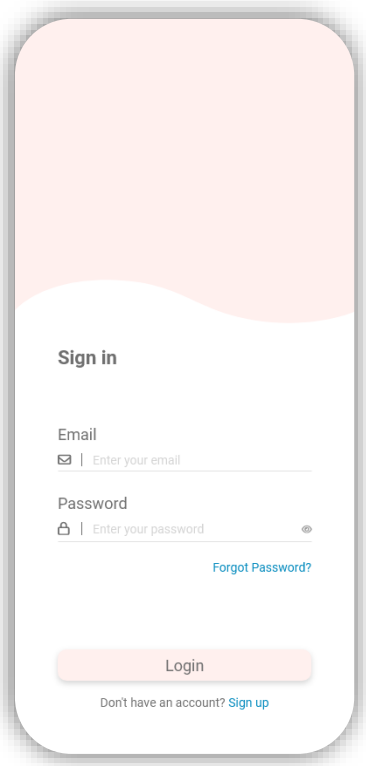

Figure B1.1 Login Page

Based on Figure B1.1 above, users can log in to the app by entering their registered email and password. New users can create an account by clicking the "Sign-up" button, while users who have forgotten their password can reset it by selecting the "Forgot Password" option. Afterward, they will receive an email with instructions to set a new password.

B-1.2 Sign Up

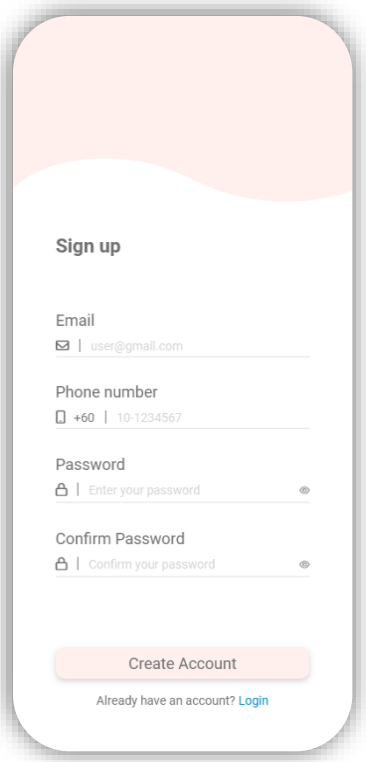

Figure B1.2 Sign Up Page

As shown in Figure B1.2, new users can create an account by entering their email, phone number, and password. They are required to confirm their password to ensure accuracy. Once the details are provided, users can proceed by clicking the "Create Account" button to complete the registration process. For users who already have an account, they can simply click the "Login" link to access their existing account.

B-1.3 AI Assistant (AI panel)

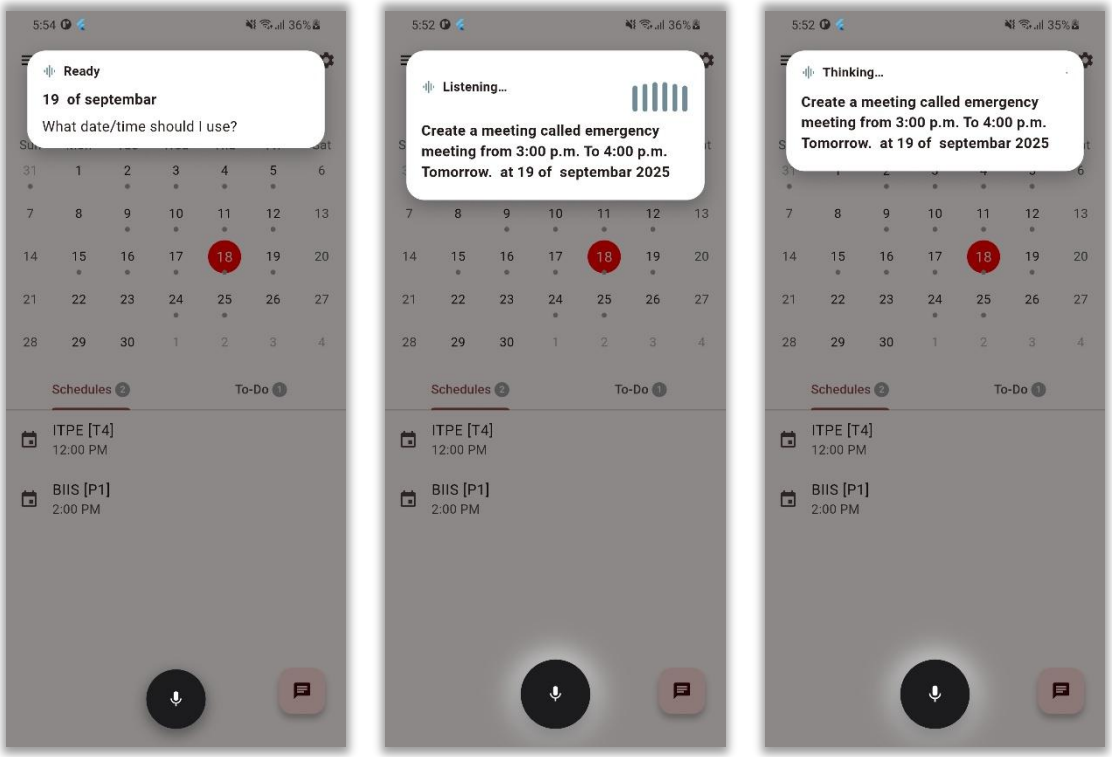

Figure B1.3 AI Assistant (AI panel)

In Figure B1.3, the voice assistant allows users to quickly add schedules by speaking instead of typing. To trigger it, press and hold the microphone button at the bottom of the screen. When the system shows “Listening…”, speak the event details, such as the title, date, and time. The spoken text will appear on screen for confirmation, followed by a short “Thinking…” stage where the assistant processes the command. If needed, it may ask a follow-up question to clarify details like the date or time. Once confirmed, the schedule is automatically created and displayed in the calendar. This feature provides a fast, hands-free way to capture new events and is especially useful when multitasking.

B-1.4 AI Assistant (Chatbot)

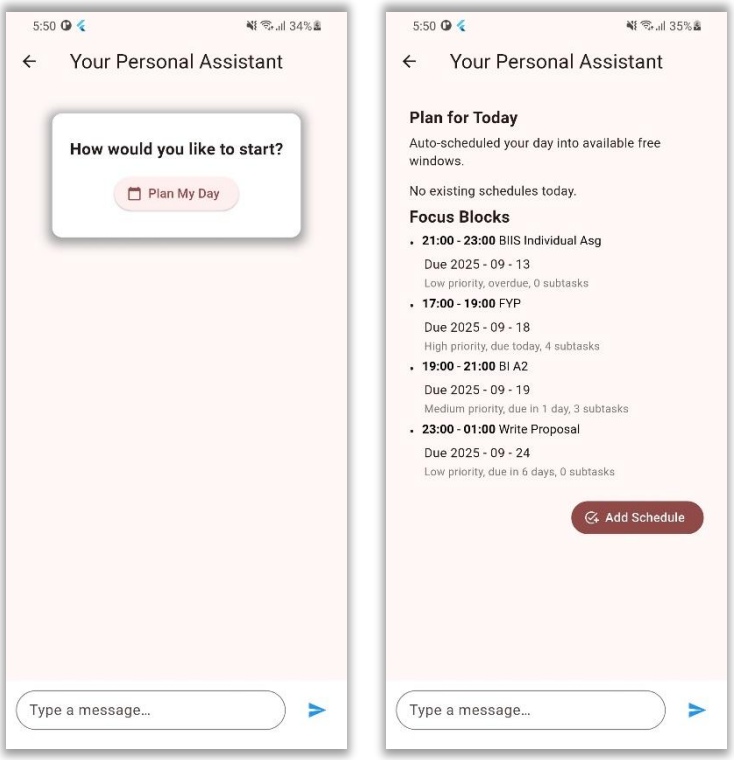

Figure B1.4 AI Assistant (Chatbot)

Figure B1.4 Personal Assistant screen is the central place where users can interact with the AI assistant for smart task and schedule management. At the start, the assistant asks, *"How would you like to start?"* and provides a Plan My Day button as a quick option. Tapping this button allows the system to automatically review the user's pending tasks, deadlines, and available free time, then generate a suggested daily plan with focus blocks.

Once a plan is created, the screen displays a clear breakdown under Plan for Today. Each focus block shows the allocated time, the task title, its due date, priority level, and the number of subtasks. Overdue and urgent tasks are scheduled earlier, while lower-priority ones are slotted later. This gives the user a structured, time-based plan that balances deadlines and priorities. Users can also add schedules directly from this screen by tapping the Add Schedule button, making it easy to insert additional commitments alongside the AI's generated plan.

Beyond the Plan My Day feature, users can interact with the assistant by typing into the message box at the bottom. By entering free-text queries or instructions (e.g., "Show my upcoming tasks," "Suggest when to study for BIIS," or "Add a reminder for tomorrow"), the assistant responds with personalized insights, suggestions, or actions.

B-1.5 Home

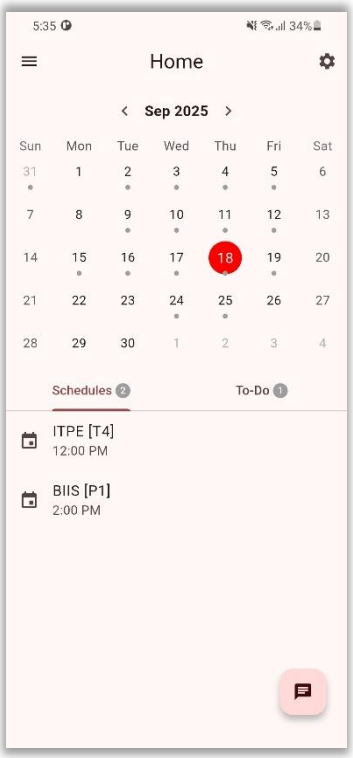

Figure B1.5 Home Page

As shown in Figure B1.5, the home screen features a calendar view for the month of September 2025. The user can switch between different months in the calendar view to navigate through their schedule. Next, the "Schedules" tab will list all schedules created by the user that are due for the selected date and will also sync with the user's device calendar if permission is granted. Meanwhile, the "To-Do" tab will display all to-dos that are due for the selected date. These provide an organized view of both scheduled events and tasks. Besides, long pressed the "+" icon may trigger the AI assistant panel or tap to open chatbot.

B-1.6 Settings

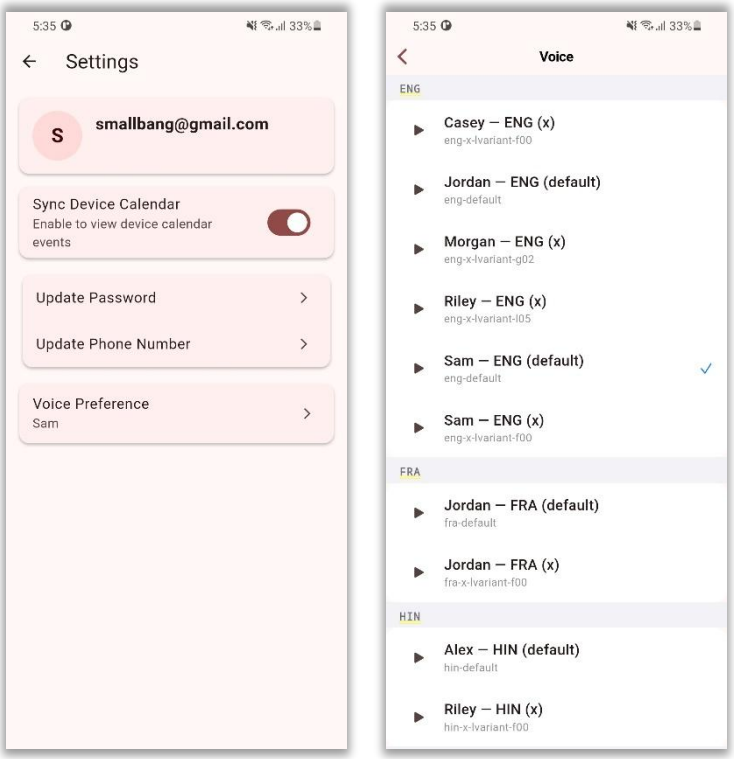

Figure B1.6 Settings

Based on Figure B1.6 Settings, users can manage their core account preferences and app integrations. At the top, the account email is displayed to confirm the signed-in identity. This cannot be changed here since it serves as the unique login identifier. Below it, a switch allows users to enable or disable synchronization with their device's calendar. When turned on, the app is permitted to read calendar events directly from the phone, which then appear alongside in-app schedules and tasks. This ensures a unified view of both personal and academic or work events. The screen also provides options to update the password and phone number, allowing users to maintain account security and keep their contact information current.

The Voice Preference screen lists available voices categorized by language, such as English, French, or Hindi, each with multiple tonal variants. Users can preview a voice by pressing the play button and then select it by tapping the voice entry, which will mark it with a check icon. The currently active voice is displayed in the Settings screen under the Voice Preference option. By adjusting this setting, users can personalize how the AI assistant and read-aloud features sound, making the app more comfortable and familiar for daily use.

B-1.7 Schedule List

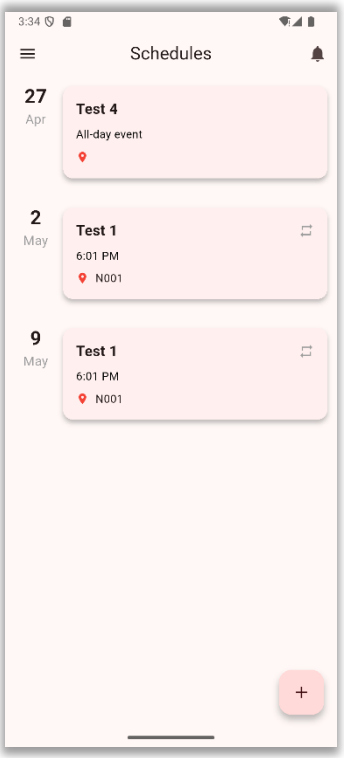

Figure B1.7 Schedule List

As shown in Figure B1.7, the "Schedules" screen displays all the schedules created by the user, listed in chronological order. The dynamic prioritization algorithm is applied based on the deadline set for each event. If a schedule is due today, its date will be highlighted in red to visually indicate urgency. Additionally, the system automatically archives schedules that have passed their due date, but they can be restored later in the "Dashboard" under the "Overdue Schedules" section. The user can click on any scheduled event to view more details, and a "+" button at the bottom allows the user to quickly add a new schedule. Besides, long pressed the "+" icon may trigger the AI assistant.

B-1.7.1 Schedule View

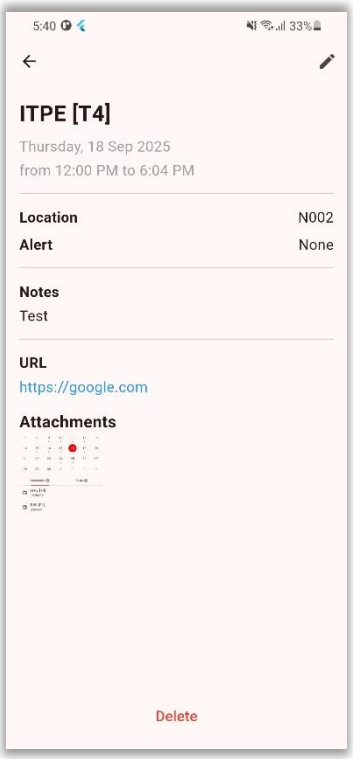

Figure B1.7.1 Schedule Details

As shown in Figure B1.7.1, users can view the details of their schedules by selecting a specific event. The screen displays all the relevant information, including the event's location, alert type, notes, and any associated URL. Additionally, users can view attachments linked to the event, such as PDFs or images. The user can also edit or delete the event using the corresponding buttons at the bottom and top of the screen. This provides a comprehensive view and easy management of each scheduled event.

B-1.7.2 Schedule Add/ Edit

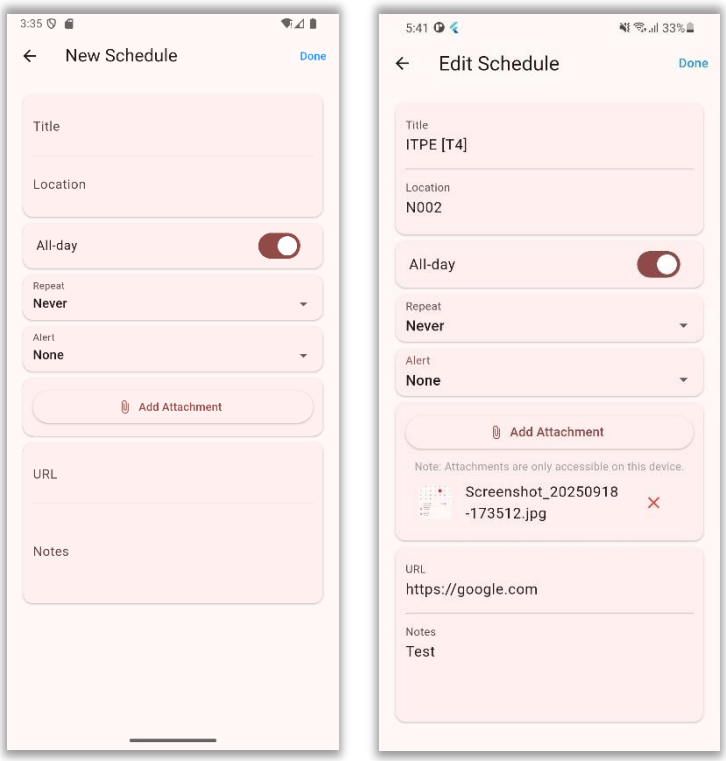

Figure B1.7.2 Schedule Add/ Edit

Figures B1.7.2 show the "New Schedule" and "Edit Schedule" screens. The layout for both screens is nearly identical, with the primary difference being that the "Edit Schedule" screen fetches and displays the data of an existing event, while the "New Schedule" screen is used to create a new event.

In both screens, the user can input the title, location, and set the event to be "All-day" if necessary. The "Repeat" option allows users to set the frequency of the event, while the "Alert" dropdown allows for setting reminders. Additionally, users can add attachments by clicking on the "Add Attachment" button, which is available on both screens.

Both screens also allow the user to enter a URL and notes related to the schedule. Once the required information is filled in, the user can either save or update the event by clicking the "Done" button. If the event being edited is part of a recurring series, the system will prompt the user with an option to either edit only this specific schedule or all occurrences in the series. This gives the user flexibility in managing recurring events.

B-1.8 To-Do List

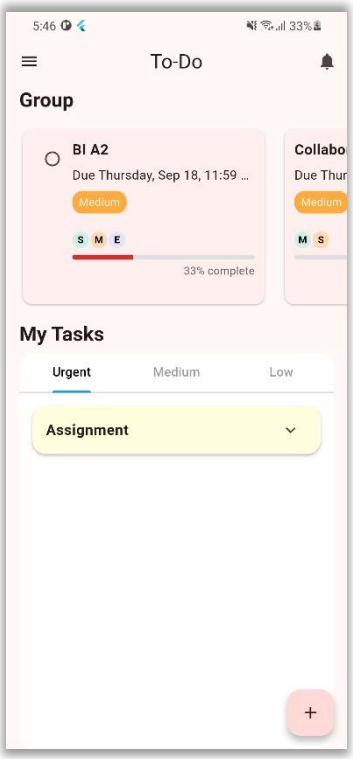

Figure B1.8 To-Do List

As shown in Figure B1.8, the **To-Do** screen is divided into two main sections "Group" and "My Tasks". Each task card displays important details such as the due date, priority level (High, Medium, or Low), and a progress bar that reflects task completion. Only the task creator and assigned admins have the ability to edit or mark tasks as complete using the checkmark option. Regular members are limited to viewing the tasks and cannot make changes or updates. The progress bar is dynamically updated based on the completion status of subtasks within each task. When a user marks a subtask as done, the system automatically recalculates the overall progress percentage and adjusts the bar. For instance, if a task has four subtasks and two are completed, the progress bar will indicate 50% completion.

The dynamic weighted prioritization algorithm operates in both the "Group" and "My Tasks" sections. It evaluates deadlines, subtask counts, alert settings, and priority levels to assign urgency scores. Tasks with higher scores appear under the Urgent tab to ensure timely attention. In My Tasks, items are further grouped into user-defined categories, making organization easier. Once a task is marked as done, users are prompted to archive it, moving it into the Completed Tasks section in the Dashboard, where it remains recoverable. Additionally, the **"+"** button at the bottom enables new task creation, while a long press activates the AI assistant for smart task input.

B-1.8.1 To-Do View (Individual)

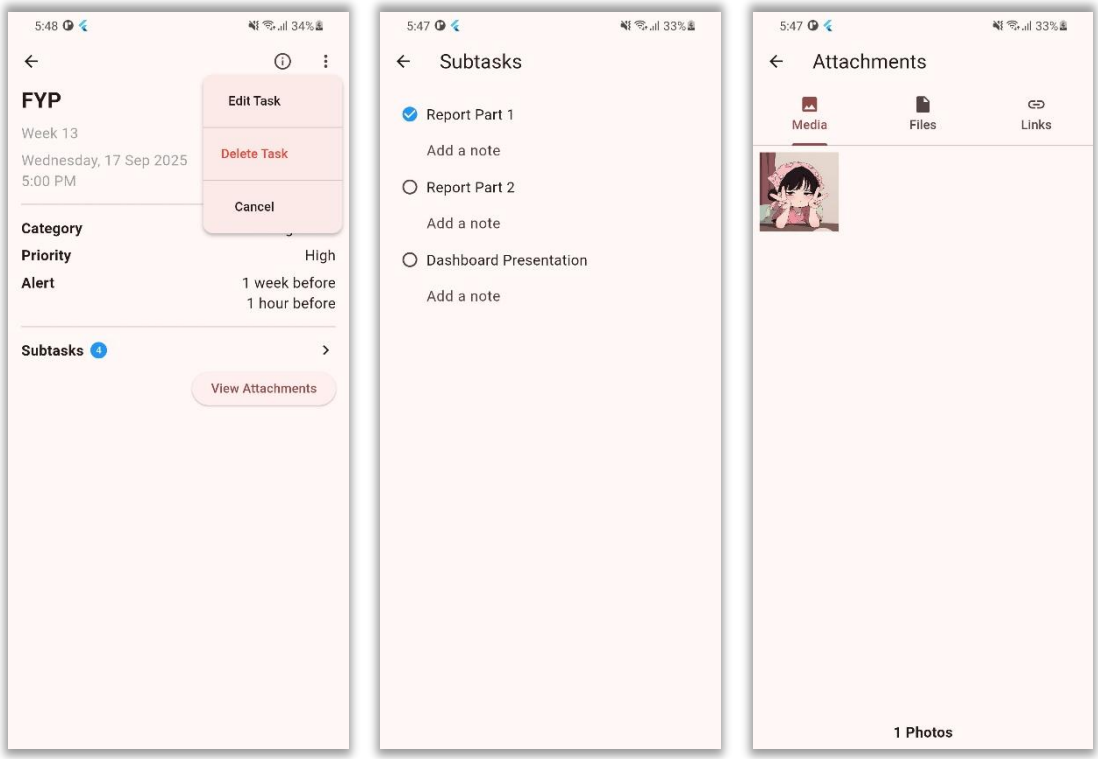

Figure B1.8.1 To-Do Details (Individual)

Figure B1.8.1 shows the detailed view of a task, where the user can view and edit the task's attributes. The task includes information such as the title, category, priority level, due date, and alert settings. Additionally, there is a list of subtasks, with the option to add notes for each subtask. Users can also see whether there are any attachments associated with the task.

Additionally, there're options to either "Edit Task" or "Delete Task." If the task is being edited, the user can modify its details, while the "Delete Task" option allows for the task's removal. The "Cancel" button allows users to exit the task info window.

This allows users to manage their tasks efficiently, including making edits, viewing details, and handling subtasks with ease.

B-1.8.2 To-Do View (Collaborated)

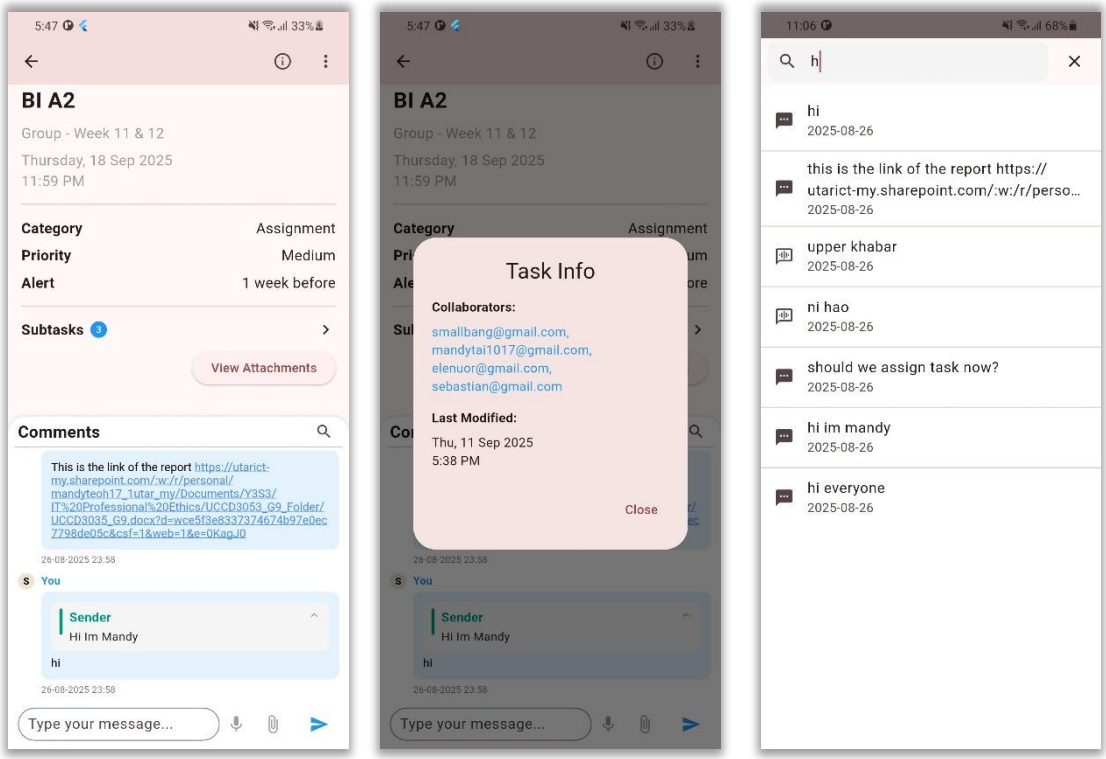

Figure B1.8.2 To-Do Details (Collaborated)

Figure B1.8.2 Collaborated To-Do Details screen provides a complete view of shared tasks where multiple users can contribute. At the top, the task title, group, and due date are displayed, along with its assigned category, priority level, and any alert reminders set. If the task contains subtasks, the progress bar automatically updates based on the number of subtasks completed, giving all collaborators a real-time indication of how far the task has advanced. Attachments related to the task can also be accessed directly through the View Attachments button.

Below, the Comments section functions as a live discussion board for collaborators. Users can exchange text messages, share external links, or upload attachments, ensuring smooth communication within the context of the task. They can even transcribe the voice message sent to text. Every message is timestamped, allowing collaborators to track the sequence of updates and discussions. A search bar at the top enables quick retrieval of past comments by keywords, making it easy to revisit important information without scrolling through the entire chat history.

Additional task information is available through the Task Info panel, which lists all collaborators' email addresses and the last modified date. This transparency ensures that every member is aware of who is involved and when the task was last updated.

B-1.8.3 To-Do Add/ Edit

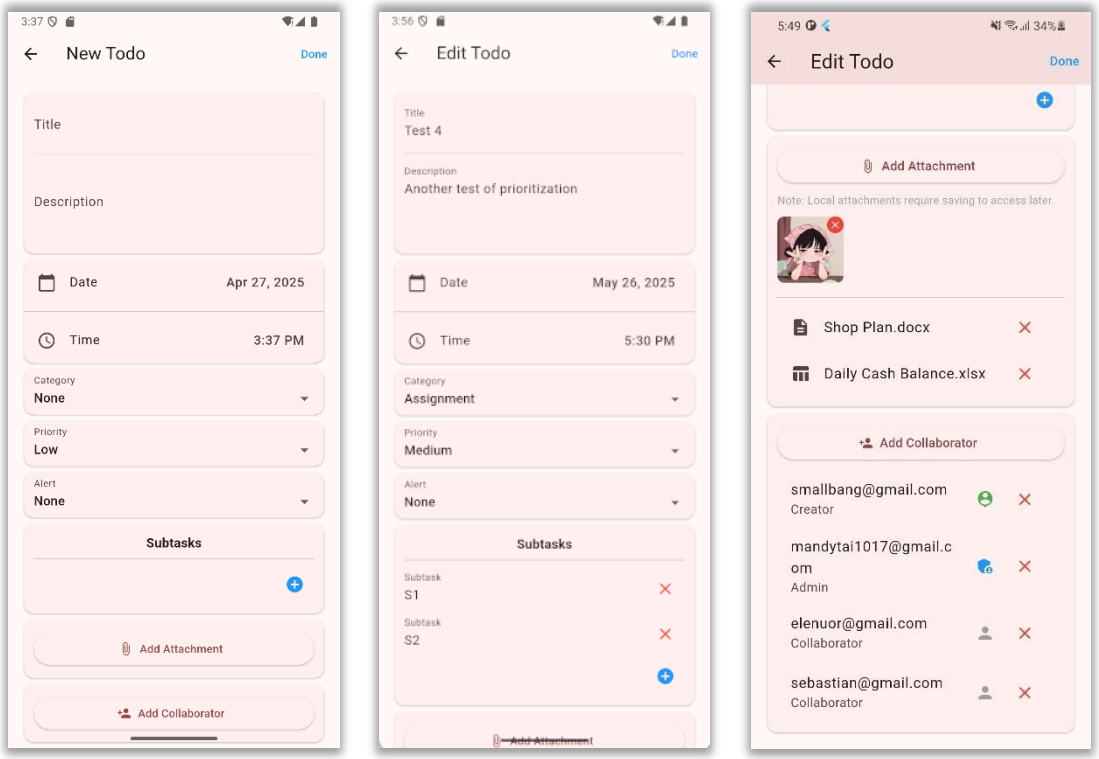

Figure B1.8.3 To-Do Add/ Edit

Figure B1.8.3 shows the "New Todo" screen where users can create a new to-do. The screen allows users to input essential information, including the task title, description, due date, and time. Users can also categorize the to-do, assign a priority level (e.g., Low, Medium, High), and set an alert for reminders. The "Subtasks" section enables the user to add subtasks to break down the main task into smaller steps. Additionally, users can attach files or add collaborators to the to-do by clicking the respective buttons. Once the necessary details are filled, users can save the new to-do by clicking "Done."

The "Edit Todo" screen, which is nearly identical to the "New Todo" screen. The key difference is that this screen is used for editing an existing to-do. In this case, the task's details are pre-filled with data such as the title, description, due date, and time. The user can edit the category, priority level, and alert settings. In the "Subtasks" section, users can modify existing subtasks (such as adding or deleting them) and update any other details of the to-do. The "Add Attachment" and "Add Collaborator" options remain available for any changes. The creator is allowed to set any team members to admin to manage the collaborated task. After making the necessary edits, the user can save the updated to-do by clicking "Done."

B-1.9 Dashboard

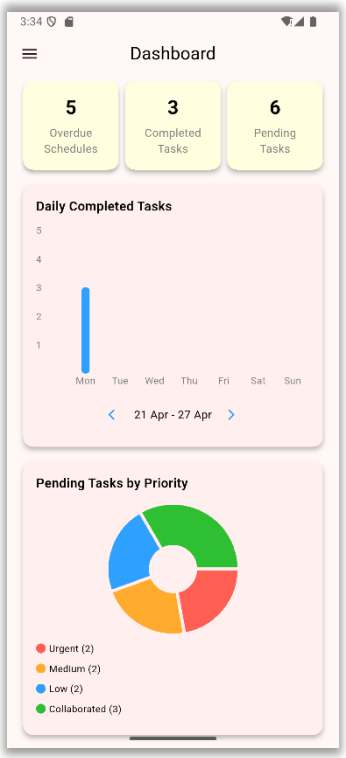

Figure B1.9 Dashboard

As shown in Figure B1.9, the "Dashboard" screen provides a quick overview of the user's task and schedule data. At the top of the screen, there are three cards displaying the number of overdue schedules, completed tasks, and pending tasks. These cards not only show the total count but are also clickable. Clicking on the "Overdue Schedules" or "Completed Tasks" cards will navigate the user to the respective list, where they can view, delete, or restore events and tasks as needed. This feature allows users to manage overdue or completed items directly from the dashboard.

Below that, the "Daily Completed Tasks" graph presents a visual representation of tasks completed over the week, with a bar chart showing the number of completed tasks for each day from April 21 to April 27. Users can switch between different weeks by clicking the back-and-forth buttons to navigate through past or upcoming weeks, helping them track productivity over time.

The "Pending Tasks by Priority" pie chart categorizes tasks based on their priority level: Urgent, Medium, Low, and Collaborated. This visual breakdown helps users easily see how their tasks are distributed according to priority and allows them to focus on high-priority items as needed.

B-1.10 Notification

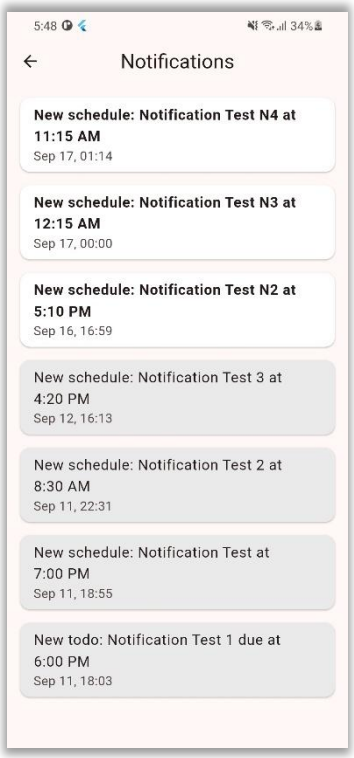

Figure B1.10 Notification

Figure B1.10 Notification provides a centralized view of all alerts generated by the system. Each notification card clearly indicates whether it is related to a new schedule or a new task, followed by the title and the scheduled or due time. Beneath the title, a timestamp shows the exact date and time the notification was issued, allowing users to trace when changes or reminders were created. Notifications are arranged with the most recent appearing at the top for quick reference.

In addition to simply listing alerts, this screen also supports interaction. By tapping a notification card, the user is taken directly to the corresponding task or schedule, where full details such as descriptions, collaborators, or reminders can be reviewed. This eliminates the need to search manually and ensures that users can respond quickly to new or updated items. Serving both as a reminder log and a navigation shortcut, the Notifications screen helps users stay organized and up to date with minimal effort.