Software Engineering Project
CS383

Qassim University جامعة القصيم
College of Computer | كلية الحاسب

Welcome to

StudyMate

Your friendly app that keeps your studies on track-easy, focused, and stress-free

Remaz Almarzouq
Hessah Alharbi
Thekra Alyahya
Reef Albarakat
Elaf Aldubayan
Budur Aljohani
Kadi alseadoun
Munirah Alharbi
ahad Alharbi
leena Almutairi

Supervisor: Dr.Alaa Alsehail

Table of Contents

Introduction

The **StudyMate** application represents an innovative digital tool that supports students in managing their academic tasks effectively. through task creation, scheduling, reminders, and calendar management. It provides a centralized platform where students can create and organize assignments, track deadlines, set reminders, and monitor their progress. By offering a clear overview of all academic tasks in one interface, StudyMate enables students to prioritize their workload, plan study time effectively, and reduce stress caused by disorganized schedules.
The application strongly aligns with the vision of digital transformation in education, introducing technology-driven solutions to enhance learning productivity and efficiency. Its smart features support not only individual task management but also facilitate better monitoring and communication between students and educators.

Key Contributions

- Enables easy creation and management of tasks, assignments, and project deadlines.
- Provides reminders and notifications to ensure timely completion of tasks.
- Offers a calendar overview for better planning and organization.
- Allows monitoring of task progress, promoting accountability and efficiency.
- Enhances communication and coordination in academic workflows.
- Encourages adoption of structured, technology-driven study and work habits.
- Contributes to overall academic productivity and supports a smooth digital learning environment.

By combining task creation, reminders, calendar management, and progress monitoring, StudyMate serves as a comprehensive tool for all academic users, helping them stay organized, productive, and engaged in a technology-enhanced educational experience.

Project Management

meeting No.	Platform	Agenda	Minutes Summary
1	Zoom	Discussed project idea and tool.	Project idea approved and repo created.
2	Zoom	Reviewed requirements and SRS start.	Requirements finalized and tasks assigned.
3	Zoom	Reviewed functional requirements.	FRs completed and SRS sections divided.
4	Zoom	Merged SRS and checked formatting.	SRS finalized with diagrams added.
5	Zoom	Chose system architecture.	Architecture approved and SDD tasks assigned.
6	Zoom	State and sequence diagrams completed.	Worked on UML diagrams.
7	Zoom	SDD finalized with all UML diagrams.	Reviewed full SDD.
8	Zoom	Combined SRS + SDD.	Final PDF prepared and uploaded.

Software Engineering Ethics

Software Engineering Ethics

To develop software systems that are safe, equitable, and reliable, adherence to ethical principles and professional responsibilities is essential. The **StudyMate** application incorporates the following ethical duties and principles to maintain system integrity, foster trust, and protect user interests.

1. User Privacy

Principle: Software engineers must respect users' privacy and confidentiality, as well as the integrity of their personal information.

Application: The Student Task Tracker collects only essential data, such as email addresses, usernames, and task specifics. Restricted access and authentication techniques are employed to securely store and protect all user data. Sensitive information is not disclosed to third parties. Furthermore, the system complies with ethical data handling standards, ensuring the confidentiality of students' personal and academic information.

2. Data Accuracy

Principle: Engineers are responsible for ensuring that all data processed and displayed by the system is accurate, consistent, and up-to-date.

Application: The Student Task Tracker implements validation checks to prevent erroneous entries, such as invalid dates or duplicate tasks. User updates are synchronized in real-time to maintain data integrity. Additionally, testing procedures are established to ensure that task progress, deadlines, and notifications are presented accurately, providing students with reliable information for managing their workload.

Software Engineering Ethics

3. Reliability

Principle: Software must be dependable and consistently function as intended.

Application: The Student Task Tracker undergoes regular system testing, debugging, and performance assessments to ensure reliability. Backup and recovery features are integrated to mitigate data loss or system failures. These precautions enable students to depend on the system for tracking their academic progress and managing deadlines without concerns about unexpected malfunctions.

4. Transparency

Principle: Developers should fully inform users about the system's functionality, the data it collects, and any associated risks or limitations.

Application: The Student Task Tracker clearly communicates its objectives, features, and privacy policies. Users are informed about the types of data collected and their intended purposes. Any limitations, such as data storage or maintenance schedules, are explicitly stated. To promote transparency and accountability throughout the development process, all design decisions and updates are thoroughly documented

Software Process Model

Selected Software Process Model

Selected Model: Incremental Process Model

The **Incremental Process Model** is a structured yet adaptable approach to software development that divides the project into smaller, manageable parts called *increments*.Each increment delivers a working version of the system, expanding its features progressively until the full software is complete.

It blends the clarity and organization of the **Waterfall Model** with the flexibility of **Iterative Development**, allowing continuous improvement through repeated cycles of requirements gathering, design, implementation, and testing.

We chose the **Incremental Model** for our project, *Student Task Tracker*, because it aligns perfectly with the nature of student productivity applications.University students often have changing priorities, new academic tasks, and evolving schedules throughout the semester — which makes flexibility crucial.
Unlike the rigid **Waterfall Model**, the Incremental approach allows the system to grow gradually, adapting to new requirements or improvements as development progresses.
Additionally, it maintains a clear and structured plan, making it ideal for academic projects that value both **organization and adaptability**.

Process Activities Overview

Our project, Studymate, is designed to help university students manage their daily academic activities, track tasks, monitor progress, and maintain a balanced study routine. The system is divided into several increments, each introducing new tools and improvements that enhance the student's personal organization experience.
Every increment undergoes all essential software process activities requirements, design,implementation, and testing ensuring consistent quality and functionality integration.

Project Increments
The development of StudyMate will be carried out in four main increments, each focusing on a specific set of features for the student user.

Increment 1 – Account Setup and Dashboard

Focus: Establishing the foundation of the system.
Description: This increment includes basic user registration and the creation of a personal dashboard. The student can log in, view a summary of tasks, and access all major sections of the system.
Outcome: A personalized, functional interface that forms the basis for all future increments.

Increment 2 – Task Management

Focus: Core functionality.
Description: The student can add, edit, categorize, and delete academic tasks or assignments. Tasks can be sorted by priority, subject, or due date.
Outcome: A structured and intuitive system for organizing daily academic responsibilities.

Process Activities Overview

Increment 3 – Progress Tracking and Reminders
Focus: Productivity and self-monitoring.
Description: The system enables students to mark tasks as “in progress” or “completed,” visualize their progress indicators, and receive reminders before deadlines. Additionally, the system provides an integrated calendar view, allowing students to see upcoming deadlines and scheduled tasks in a clear weekly or monthly layout.
Outcome: Increased motivation and awareness of academic performance.

Increment 4 – Reports and Goal Setting
Focus: Reflection and improvement.
Description: Adds weekly and monthly reports showing completed tasks, upcoming deadlines,and productivity trends. The student can also set academic goals and measure progress toward them.
Outcome: A comprehensive self-tracking system that supports continuous improvement and time management.

Process Activities Overview

Incremental Development Activities
Each increment follows the same disciplined sequence of software engineering activities:

1. Requirements Analysis: Define the goals and functional requirements of the new increment.

2. Design: Create or update design diagrams and user interface prototypes.

3. Implementation: Code and integrate the new features into the existing system structure.

4. Testing: Verify that the increment works properly and does not interfere with previous functionality.

5. Integration: Merge the increment with earlier components, forming a more complete and reliable version of the system.

The **Incremental Model** provides a flexible yet structured path for developing the Student Task Tracker system.
By delivering partial but functional versions of the application early,it ensures continuous improvement, early testing, and consistent user feedback.
This method minimizes risk, promotes adaptability, and results in a **practical, user-centered tool that empowers university students to organize their academic life effectively.**

Software Requirements

System Overview

StudyMate is a study planner application designed to assist students in managing their academic tasks and deadlines. The system allows users to add, edit, and organize tasks such as assignments, exams, and projects. Also providing automatic reminders before each due date.

The main objective of the **StudyMate** app is to help students improve their time management and academic performance by keeping all study -related tasks in one place. Users can create accounts, record their tasks, and view their
upcoming tasks through a calendar interface. The app also supports categorizing tasksbased on subjects or task types, filtering completed and pending tasks, and visualizing progress through a simple dashboard.

The target users of the system are university and high school students who need a simple and reliable way to keep track of study tasks and reminders. The application will run on both **Android** and **IOS** devices using a cross-platform framework(e.g., **Flutter** or **React Native**), to reduce development time and effort. For data storage, the system will be using **SQLite** for local
caching and **Firebase** for cloud synchronization and multi-device access, thereby enhancing user experience and minimizing cost where possible.

In conclusion, the **StudyMate** app provides a user-friendly and reliable solution that allows students to plan their academic workload efficiently, meet deadlines, and stay organized throughout their studies.

Requirements Engineering Plan

This section outlines the process of gathering, analyzing, validating, and modeling the requirements for the StudyMate application

Gathering Requirements

To identify user needs, the team conducted short interviews and online surveys with students from various majors. The goal was to understand how they currently manage their academic tasks and what problems they face. Most students expressed the need for timely notifications, easy task categorization, and academic goals to monitor their progress towards them. This information was used to create an initial list of functional and non-functional requirements.

Analyzing Requirements

The collected requirements were analyzed to determine their priority, feasibility, and impact on the system design. Requirements were divided into three categories:

- **Functional Requirements** : features that the system must perform (e.g., add task, edit task, set reminder).
- **Non-Functional Requirements** : performance, usability, and reliability aspects.
- **External Interface Requirements** : how the system connects with external tools such as Notification APIs and Calendar integration.

Each requirement was assigned to a specific increment to simplify implementation and testing.
Example:

- Increment 1 → User Account & Dashboard
- Increment 2 → Task Management
- Increment 3 → Progress Tracking & Reminders
- Increment 4 → Reports & Goal Setting

Requirements Engineering Plan

Validating Requirements
Validation was performed at the end of each increment. Early prototypes (e.g., Increment 1 and 2) were demonstrated to a small group of students to verify functionality and usability. Feedback on the interface and reminder accuracy was incorporated before proceeding to later increments.
This incremental validation minimized rework and ensured continuous user involvement.

Modeling Requirements
The system requirements were modeled using **UML diagrams** to visualize user interactions and data flow.

- A **Use Case Diagram** shows how the user interacts with features like Login, Add Task, Edit Task, and View Calendar.
- An **Activity Diagram** demonstrates the logical flow of task creation and reminders.
- Additional diagrams, including **Sequence** and **State Diagrams** describe the communication among system components and the lifecycle of each task.

These diagrams were created using **Lucidchart** and included in the Software Design Document(SDD) section for better understanding of the system structure.

By applying the Incremental Development Model, requirements are gathered once, analyzed, and then implemented in manageable increments. Each stage produces a tested, usable version of the **StudyMate** app, ensuring quality, reliability, and close alignment with student needs. This structured plan forms a strong foundation for the subsequent system-design and implementation phases

Functional Requirements

Development Approach The system will be developed using the **Incremental Process Model**, which divides the project into multiple increments.
Each increment will introduce a specific set of features that are built upon the previousversion until the full system is completed

increment1 : User Account and Dashboard

- The system shall allow users to register and log in securely using a username and password.
- The system shall display a personalized dashboard showing the student's tasks summary.
- The system shall allow the user to log out safely.

increment 2 : Task Management

- The user shall be able to add new tasks with title, subject, due date, and description.
- The user shall be able to edit existing tasks to update information.
- The user shall be able to delete tasks that are no longer needed.
- The system shall allow the user to categorize tasks (by subject or priority).

Functional Requirements

increment 3 : Progress Tracking and Reminders

- The system shall allow users to mark tasks as completed or in progress.
- The system shall display progress charts or indicators to help students visualize their completion rate.
- The system shall send reminders or notifications before deadlines.

Increment 4 : Reports and Goal Setting

- The system shall generate weekly and monthly reports summarizing completed and pending tasks.
- The user shall be able to set academic goals and monitor their progress toward them

Summary

By following the Incremental Model, the system's functionality will grow step by step starting with basic login and dashboard features, then moving toward full task management, progress tracking, and reporting capabilities

Non-Functional Requirements

Non-functional requirements represent the quality and performance standards that ensure the system's effectiveness and stability. These requirements focus on how the system performs rather than what it does. They cover aspects such as performance, usability, reliability, security, compatibility, and maintainability, which are crucial for delivering a high-quality user experience.

1. Performance Requirements

- The system shall load the user dashboard within 3 seconds upon login under normal network conditions.
- Task creation, editing, or deletion operations shall be processed and reflected in the user interface within 2 seconds.
- Notifications for upcoming deadlines shall be triggered at least 1 hour before the due time to ensure timely user awareness.

2. Usability Requirements

- The user interface shall be intuitive enough for a first-time user to add a new task within 3 minutes without external guidance.
- The application shall support both English and Arabic languages to accommodate the target user base.
- Navigation between core features (e.g., Dashboard, Task List, Progress View) shall require no more than 3 clicks.

Non-Functional Requirements

3. Reliability Requirements

- The system shall maintain 99% uptime during critical academic periods (e.g., exam weeks).
- User data shall be automatically synchronized with the cloud (Firebase) to prevent data loss in case of device failure.
- Local data (SQLite) shall remain consistent with cloud data even under poor or intermittent network connectivity

4. Security Requirements

- User passwords shall be stored in a hashed and salted format in the database.
- Each user shall only access their own tasks and data; no user shall see another user's information.

5. Compatibility Requirements

- The application shall run on iOS 13+ and Android 9.0+ operating systems.
- The user interface shall adapt seamlessly to different screen sizes (mobile phones and tablets).
- The app shall function in both online and offline modes, with synchronization occurring once the connection is restored.

6. Maintainability Requirements

- The source code shall be modular and well-documented to facilitate future updates and feature additions.
- The database schema shall be version-controlled to support seamless updates without data loss

External Interface Requirements

This section describes the interfaces between the Student Task Tracker system and external systems or services that facilitate notifications, calendar synchronization, data storage,and report generation.

- **Notification Interface**

The system should integrate with the device's Notification API (Android Notification Manager / iOS Notification Center) to deliver real-time reminders and alerts
Purpose: To notify users about upcoming deadlines, incomplete tasks, and progress reminders even when the app is not running
Inputs: Notification data (task title, due date, reminder time)
Outputs: Push notification displayed on the user's device
Error Handling: If the user denies notification permission, the system should display an in-app alert reminding them to enable notifications.

- **Calendar Integration Interface**

The system should connect with the device's calendar API (Google Calendar / iOS Calendar) to synchronize academic tasks and deadlines
Purpose: To allow users to view academic tasks alongside personal events in their native calendar applications
Inputs: Task information (title, due date, time)
Outputs: Calendar entries corresponding to the user's tasks
Error Handling: The system should handle cases where calendar access is denied by showing a notification and disabling sync functionality.

External Interface Requirements

- **Cloud Database Interface (Firebase)**

The system should interface with Firebase for cloud-based data management and user authentication
Purpose: To enable secure user login/registration and synchronize data across multiple devices
Inputs: User credentials, task data, progress updates
Outputs: Stored and retrieved data from Firebase servers
Error Handling: In case of network or authentication errors, the system should store unsynced data locally until connection is restored.

- **Local Database Interface (SQLite)**

The system should use SQLite for local data caching and offline access
Purpose: To provide fast, local access to user data when an internet connection is unavailable
Inputs: User-created tasks, updates, deletions
Outputs: Locally stored records in the device database
Error Handling: The system should ensure data integrity during synchronization between local and cloud storage.

- **Report Export Interface**

The system should include an interface for report generation and export through the device's file system or sharing services
Purpose: To allow users to generate and share weekly or monthly academic progress reports in formats such as PDF or CSV
Inputs: Task data, completion status, goal summaries
Outputs: Downloadable or shareable report files
Error Handling: If file generation fails, the system should display an error message and provide retry options.

UML Use Case Diagram

The **Student Task Tracker** is an application designed to help students manage their academic tasks. The application provides functionality to **add, edit, delete, prioritize** tasks**, set reminders, track progress,** and **generate reports**. The system is also integrated with **external systems.**

Actors :

1. Student:

Description: The **student** is the **primary user** of the system. They **interact** with the system to perform various actions such as logging in, adding tasks, setting reminders, and viewing reports.

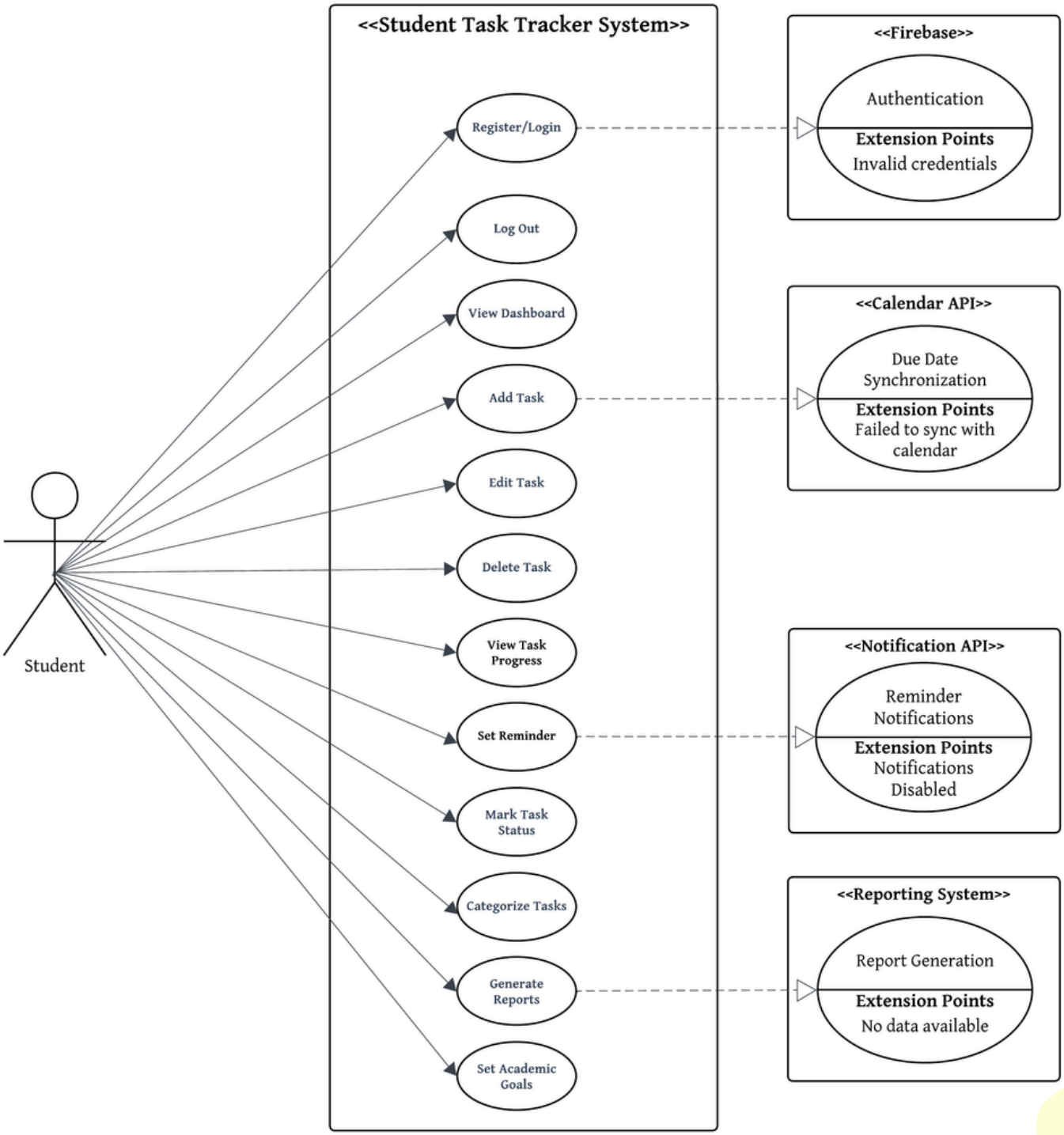
2. System:

Description: The **Student Task Tracker** is the software responsible for processing the user's inputs, managing tasks, sending notifications, and generating reports.

3. External Systems:

- **Firebase:** Handles user authentication and stores task data.
- **Notification API:** Sends reminders and notifications for tasks.
- **Calendar API:** Syncs task due dates with the student's personal calendar.
- **Reporting System:** Generates detailed reports , using tools like Google Sheets or Tableau.

UML Use Case Diagram



UML Use Case Diagram

- **Use Cases**

Here are the **key Use Cases** for the **Student Task Tracker system**:

1. Login:
- **Actor:** Student
- **Description:** The student logs into the application using their username and password.
- **External System: Firebase** (for authentication)
- **Failures:** Occurs during authentication if credentials are invalid.

2. Add Task:
- **Actor:** Student
- **Description**: The student adds a new academic task, specifying title, description, subject, due date, and priority.
- **External System: Calendar API** (for syncing the task's due date)
- **Failures:** Occurs if required fields are missing or input is invalid or synchronization fails

3. Edit Task:
- **Actor:** Student
- **Description:** The student modifies an existing task to update information.
- **External System:** None
- **Failures:** Occurs if task ID is not found or database connection fails

4. Delete Task:
- **Actor**: Student
- **Description**: The student deletes tasks that are no longer needed.
- **External System**: None
- **Failures**: Occurs if task does not exist or deletion fails due to data integrity issues.

UML Use Case Diagram

5. Set Reminder:

- **Actor:** Student
- **Description:** The student **sets a reminder** for a task before its due date.
- **External System: Notification API** (for sending reminders).
- **Failures:** Occurs if notifications cannot be set due to user settings or API failure.

6. Generate Report:

- **Actor:** Student
- **Description:** To allow users to generate and share weekly or monthly academic progress reports in formats such as PDF or CSV.
- **External System: Reporting System** (for generating and displaying reports).
- **Failures:** Occurs if there is no data to report or if report generation fails.

7.View Dashboard

- **Actor:** Student
- **Description:** The student views their personalized dashboard to see an overview of their tasks and progress.
- **External System:** None
- **Failures:** Occurs if dashboard data cannot be loaded due to network or database issues.

8. View Task Progress

- **Actor:** Student
- **Description:** The student views the progress of their tasks, showing whether they are completed, in progress, or pending.
- **External System:** None
- **Failures:** Occurs if task progress data cannot be retrieved or displayed.

UML Use Case Diagram

9. Mark Task Status

- **Actor:** Student
- **Description:** The student marks the status of a task as completed, in progress, or pending.
- **External System:** None
- **Failures:** Occurs if the task status cannot be updated due to system failure.

10. Categorize Tasks

- **Actor:** Student
- **Description:** The student categorizes tasks by subject, priority, or due date.
- **External System:** None
- **Failures:** Occurs if the categorization fails or if the system cannot store the new category.

11. Set Academic Goals

- **Actor:** Student
- **Description:** The student sets academic goals, such as completing a certain number of tasks or achieving a target score.
- **External System:** None
- **Failures:** Occurs if the goal cannot be saved or retrieved due to system failure.

12. Log Out

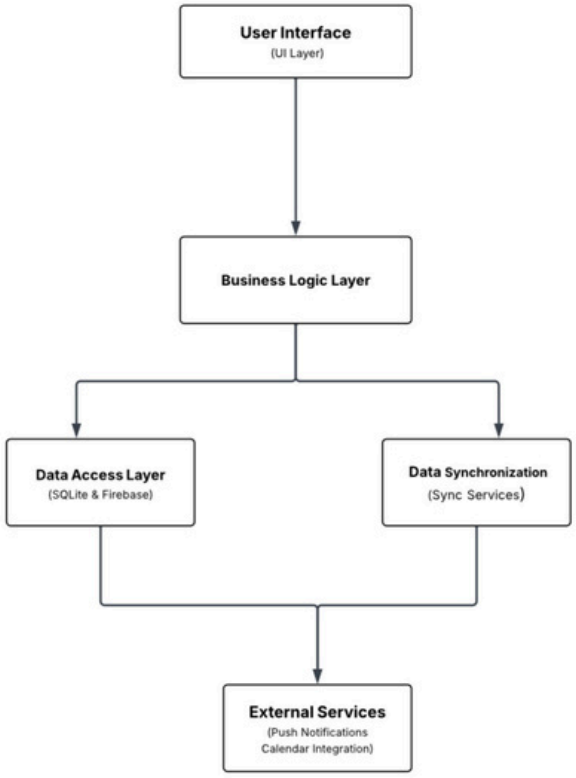
- **Actor:** Student
- **Description:** The student logs out of the application to end their session. Once logged out, the system clears any session data, and the student is redirected to the login page to authenticate again.
- **External System:** None
- **Failures:** Occurs if session data cannot be cleared or if the student cannot be redirected to the login page.

System Design & Modeling

System Architecture Model

high level architecture model

System Architecture and Components Design
The Student Task App is built so that each part of the system has a clear job and works smoothly with the others. The system follows a layered structure to keep everything organized and make future updates easier. The app runs on both iOS and Android using a cross-platform framework such as Flutter or React Native, which helps save development time because the code is written once for both platforms.

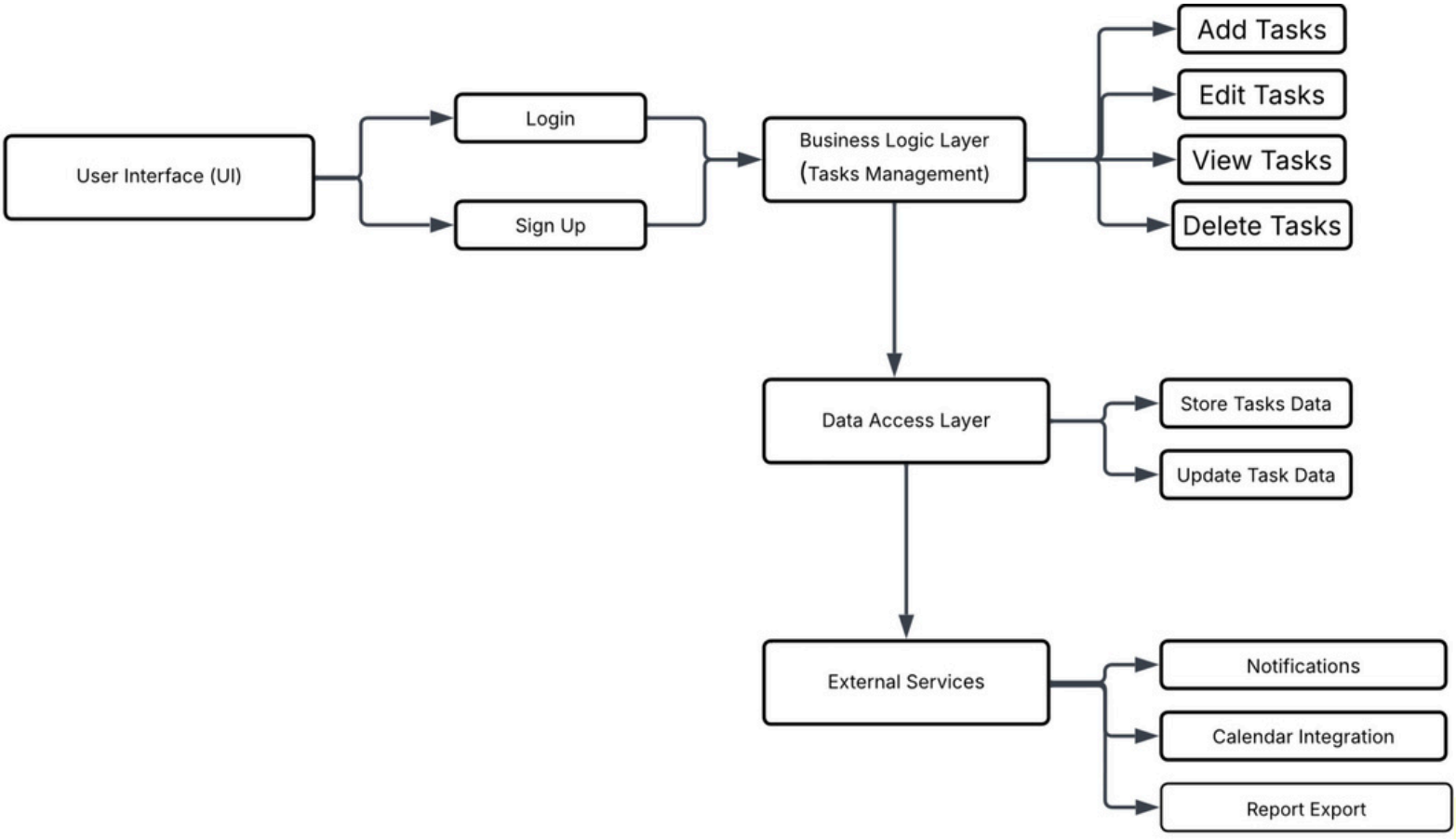


The system is divided into layers:

- **User Interface Layer:** lets students add tasks manage deadlines and track progress.
- **Business Logic Layer:** Processes user actions and connects the UI with data and services.
- **Data Layer:** uses SQLite for offline storage and Firebase for cloud syncing with a synchronization mechanism that keeps both local and cloud data updated
- **External Services:** includes notifications, calendar integration, and report export.

System Architecture Model

Component Decomposition Description



System Architecture Model

Detailed Components Design Description:

The app is divided into smaller components, and each one has a specific role to keep thesystem organized and easier to maintain:

• **Account Management Component:** Handles user registration, login, profile updates, and password recovery. It uses Firebase Authentication for login and works with the data layer to store and retrieve user profiles.

• **Task Management Component:** Allows students to add, update, delete, and categorize tasks. They can set due dates, priorities, and task reminders. This component works with the data layer to save tasks locally and in the cloud.

• **Reminder Management Component:** Manages notifications for upcoming deadlines.It works with the task component to send alerts at the right time, even if the app is not open.

• **Synchronization Component:** Keeps local and cloud storage aligned. It updates offline changes once the device is online again and handles any conflicts that may happen between versions of the data.

• **Data Access Layer:** Provides a simple way for components to store and access data. It manages communication with SQLite for offline storage and Firebase for cloud storage.

• **External Services**: Includes services such as push notifications, calendar integration, and report export functionality.

These services send reminders allow students to link tasks with their device calendar and enable the generation and sharing of academic progress reports.

System Architecture Model

Design Rationale:

1. User Experience & Flexibility

- **Cross-Platform**: Works on both iOS and Android with one codebase. Faster to build and everything feels the same.
- **Easy Interface:** Adding editing, deleting, and organizing tasks is simple and stress-free.
- **Hybrid Storage & Sync:** Tasks save offline and sync online automatically so data is always up-to-date.

2. External Services

- **Notifications**: Alerts come even if the app is closed so deadlines are never missed.
- **Calendar & Reports:** Sync with Google Calendar and export reports (PDF/CSV) to track progress easily.

3. Security & Reliability

- **Data Safety:** Everything is encrypted and password-protected.
- **Consistent Data:** Offline changes sync properly keeping all devices updated.

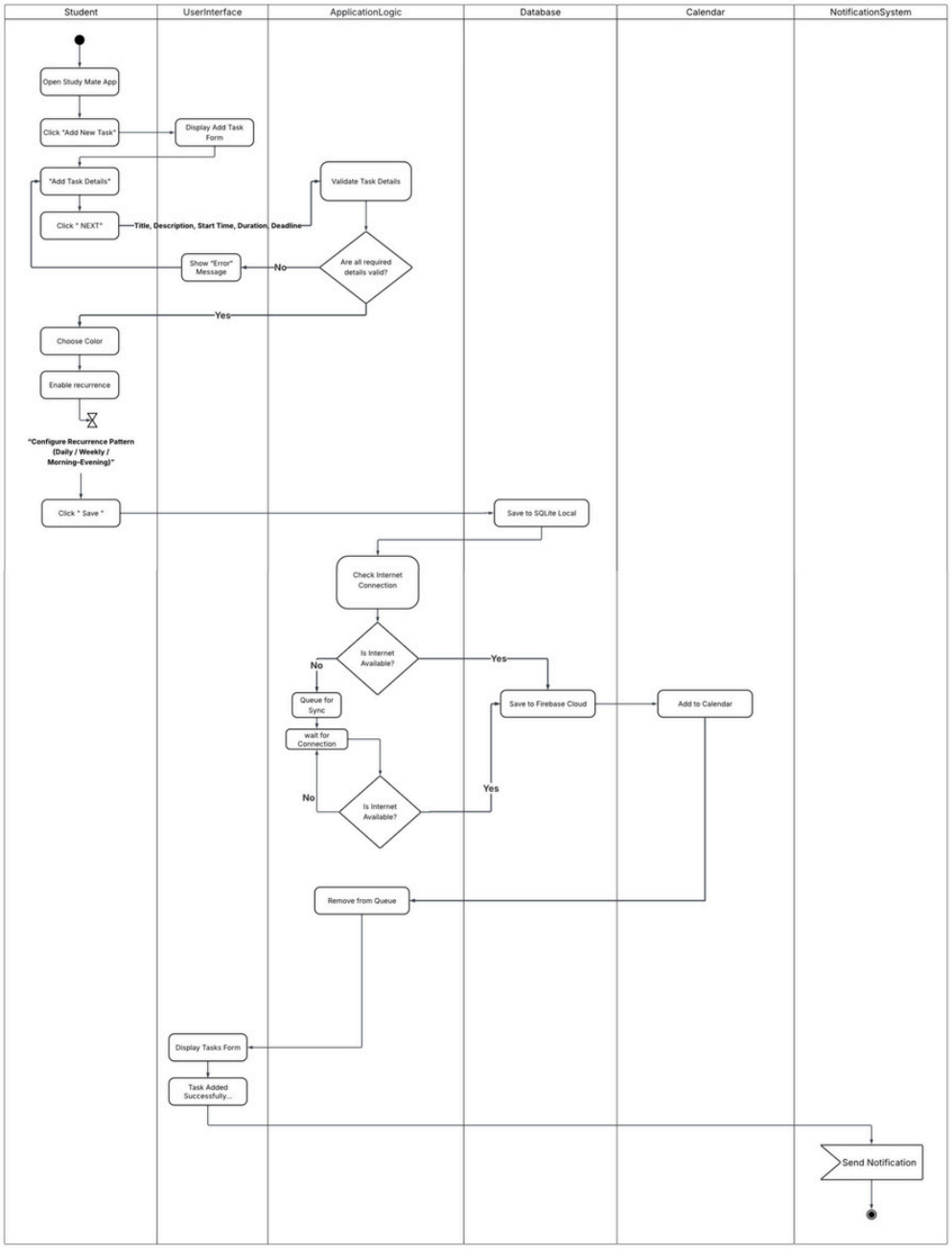
4. Performance & Usability

- **Fast & Simple**: Quick access to tasks, clean interface easy to use.
- **Reminders**: Timely notifications help students stay on top of their tasks.

Activity Diagram

The user opens the app, enters the basic task details, and submits the form.
The system validates the required inputs, then processes the priority color selection and optional recurrence settings, saving the task locally .and synchronizing it with the cloud if internet is available
Finally, the app updates the task list, sorts the tasks by their priority colors, and schedules the required notifications

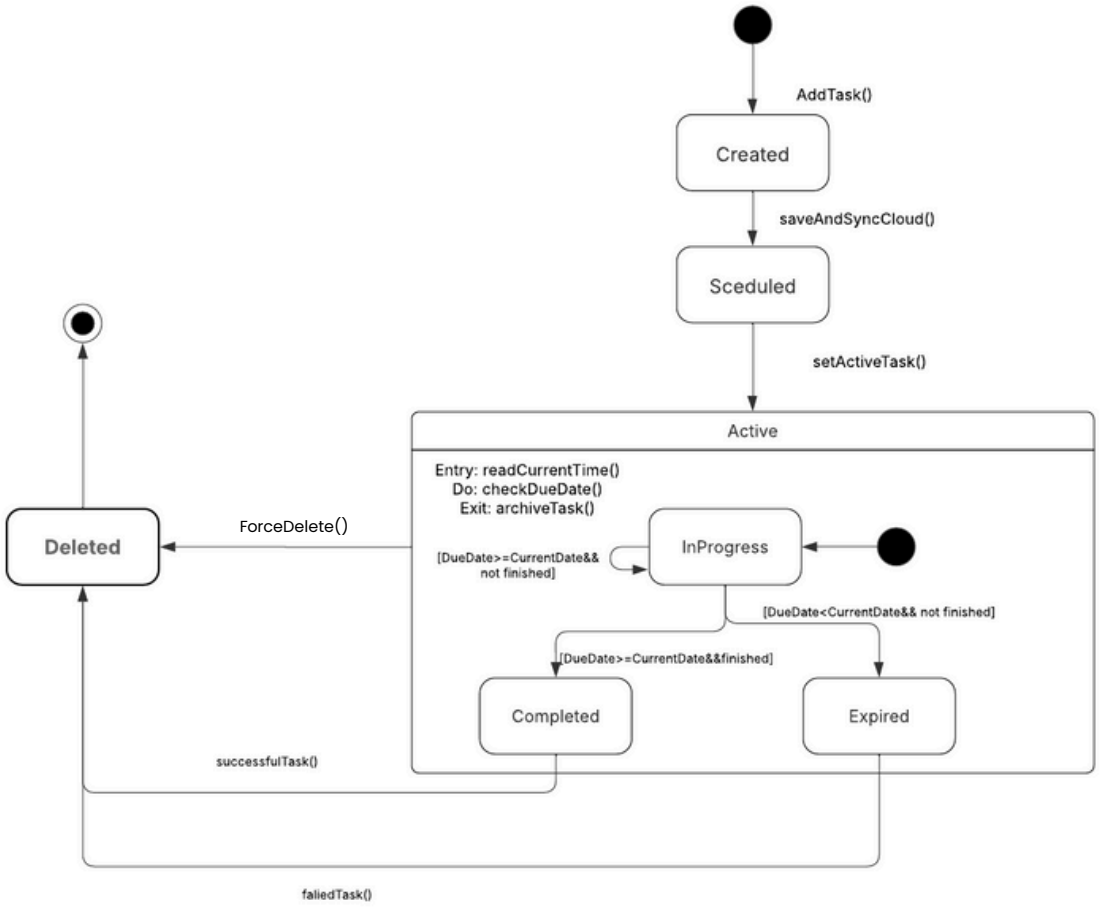
click to view clearly



State Diagram

Scenario: task states

The task in the Student Task Manager System progresses through several states as the student interacts with it. Initially, a task is in the Created state when the student adds it to the system. Once it is saved and synchronized with the cloud or calendar service, it moves to the Scheduled state. When the task becomes Active, the system continuously checks the due date and the student's progress. If the student completes the task before or on the due time, the task moves to the Completed state. If the due date passes without completion, the task transitions to the Expired State. All completed or expired tasks are then Archived automatically for record-keeping. Additionally, the student can manually archive a task at any time or restore an archived task to make it active again.



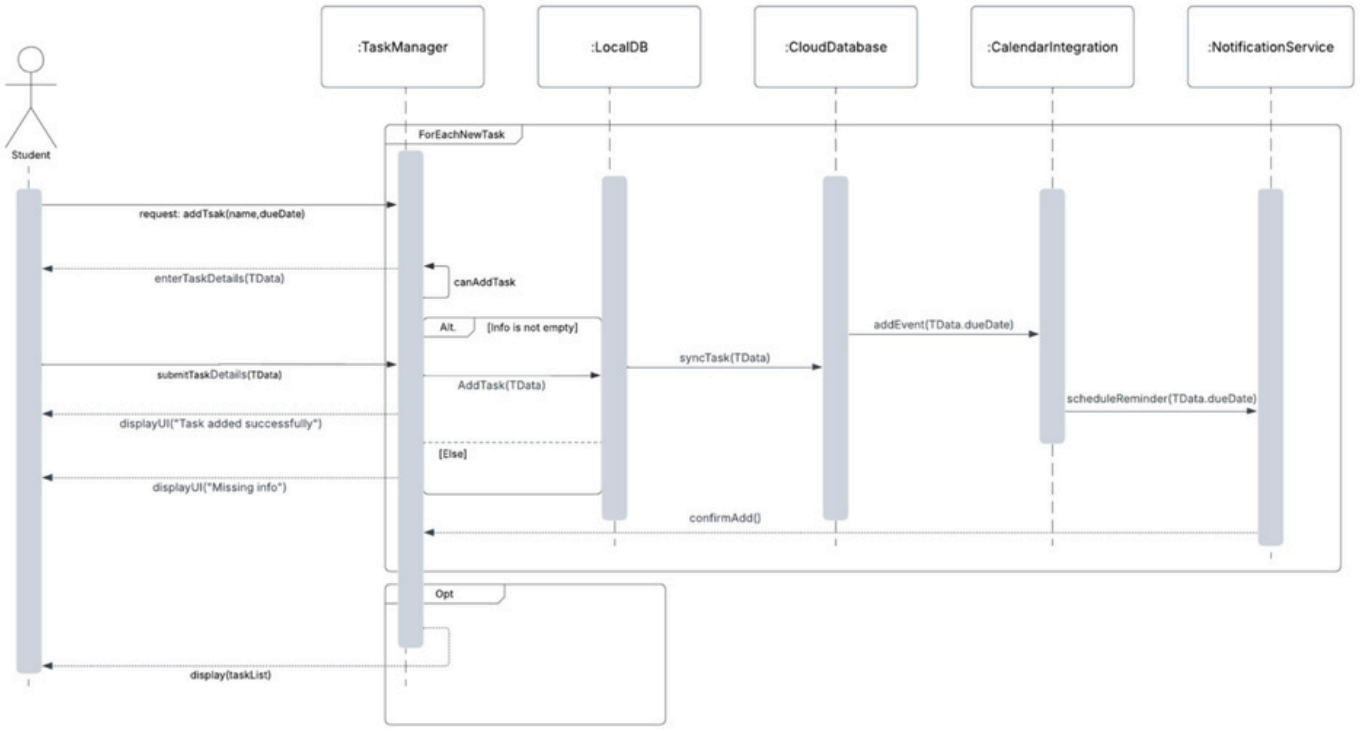
Sequence Diagram

Scenario: Adding a task

A student uses the task management system to add a new task. Once the student submits the task details, the system first stores the task locally. Then it synchronizes the data with the cloud server. After that, the system adds the task to the student's calendar and schedules notifications. Finally, the system confirms that the task was added successfully, and the student can get the task list.

Actors & System objects:

- **Actor:** Student
- **System objects:** TaskManager, LocalDB, CloudDatabase, CalendarIntegration, NotificationService



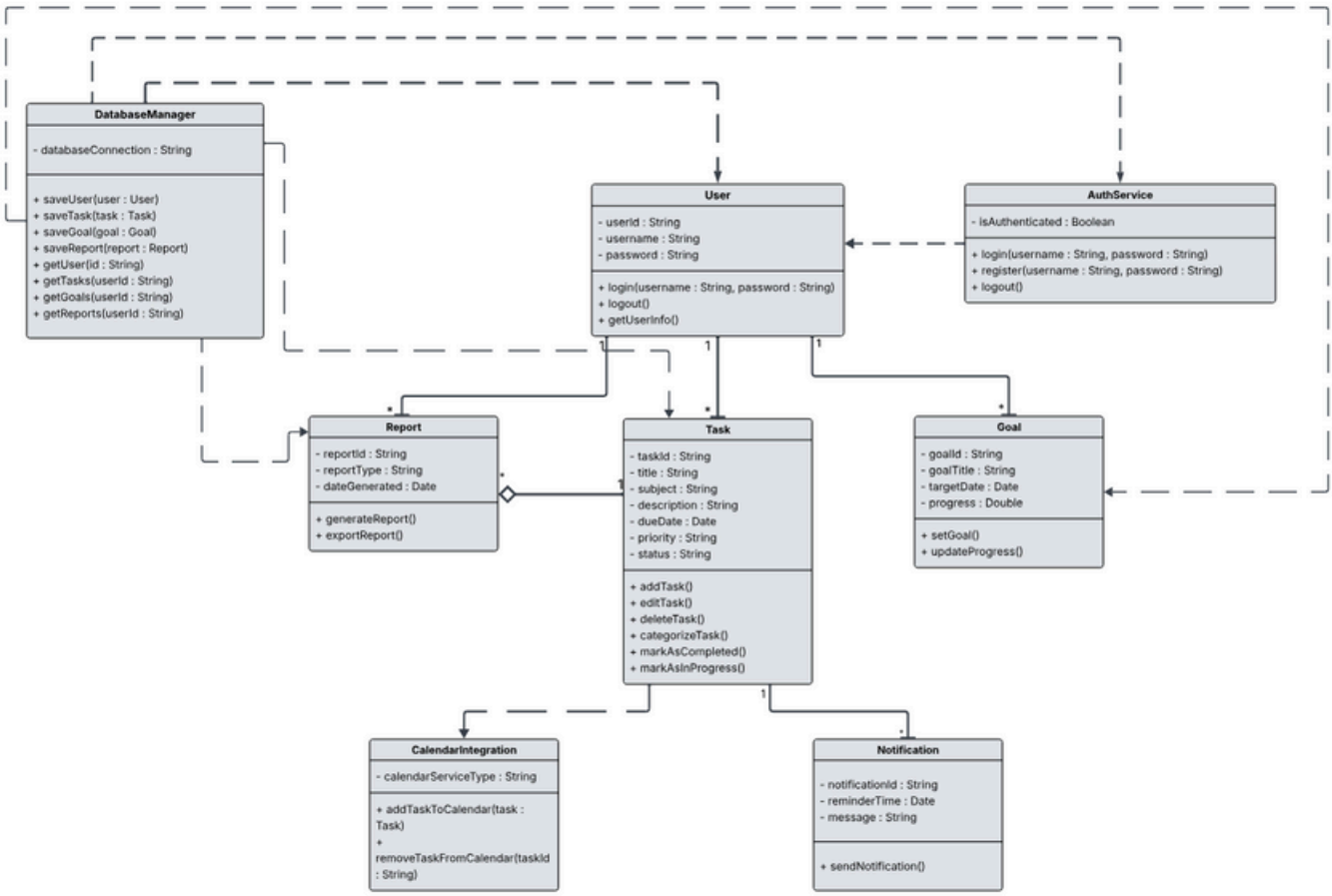
Class Diagram

Scenario:

In Study Mate, the User is the main actor. After logging in through the AuthService, the user can create and manage different academic elements. Each user can create multiple Tasks, where every task stores details like title, due date, priority, and status. Tasks may also generate Notifications to remind the user about deadlines. If the user chooses to sync a task with their device calendar, the task briefly uses the CalendarIntegration external service to add or remove events.

The user can also set Goals, which help track long-term study progress. When the user wants to see an overview of their performance, the Report class collects relevant tasks to generate a summary, such as completed tasks and upcoming deadlines.

All data—users, tasks, goals, and reports—is saved and retrieved through the DatabaseManager, which acts as the system's data handler. Each class depends on the DatabaseManager only when it needs to store or access information



Human Interface Design

Overview of the User Interface

The user interface of StudyMate is designed to provide students with a clear, intuitive, and well-organized digital environment to manage their academic responsibilities. The application focuses on simplicity, visual clarity, and accessibility, ensuring students of different technical backgrounds can easily interact with the system.

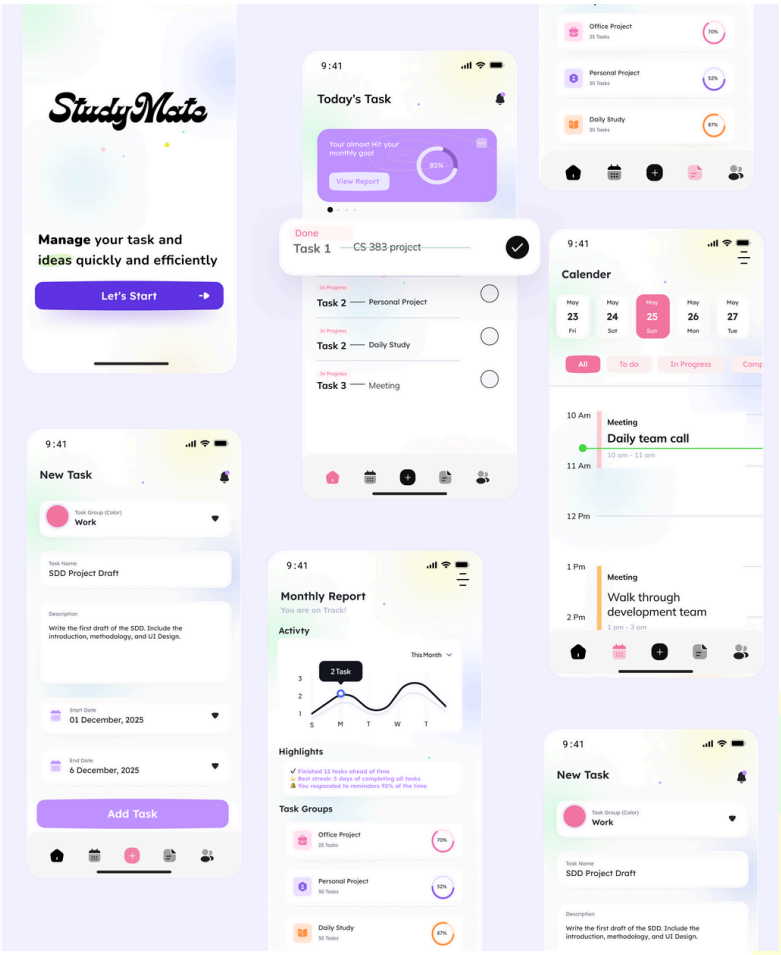
From the end user's perspective, StudyMate supports the following main functionalities:

- Creating and managing academic tasks such as assignments, quizzes, and study reminders.
- Scheduling deadlines and events using a built-in calendar.
- Receiving notifications and reminders for upcoming tasks and overdue items.
- Monitoring progress through charts and task-completion indicators.

Detailed Design of the User Interface

StudyMate

Your friendly app that keeps your studies on track-easy, focused, and stress-free



Conclusion

The StudyMate project delivers a well-engineered and fully integrated solution designed to enhance students' academic productivity, streamline task management, and improve the overall learning experience. Throughout its development, the team applied structured software engineering practices—from organized project management and clear task distribution to systematic planning and documentation—resulting in a workflow that reflects real industry standards.

Ethical principles such as privacy, data accuracy, reliability, and transparency were embedded into the system to ensure a secure and user-centered environment. The Incremental Process Model played a key role in the project's success, enabling gradual development through well-defined increments, continuous testing, early feedback, and smooth integration of new features. Each increment introduced essential functionality, including account management, task creation, progress tracking, and automated reporting.

Through a rigorous Requirements Engineering process, the team gathered, analyzed, validated, and modeled requirements using professional techniques. Functional, non-functional, and external interface requirements were clearly defined and supported with comprehensive UML diagrams that accurately represented system behavior and interactions.

The System Design phase transformed these requirements into a scalable, modular, and maintainable multi-layered architecture. With features such as offline–online synchronization, calendar integration, automated notifications, and secure data management, the system provides a robust and reliable user experience. Usability and performance were also carefully considered, ensuring that StudyMate is intuitive, efficient, and practical for daily academic use.

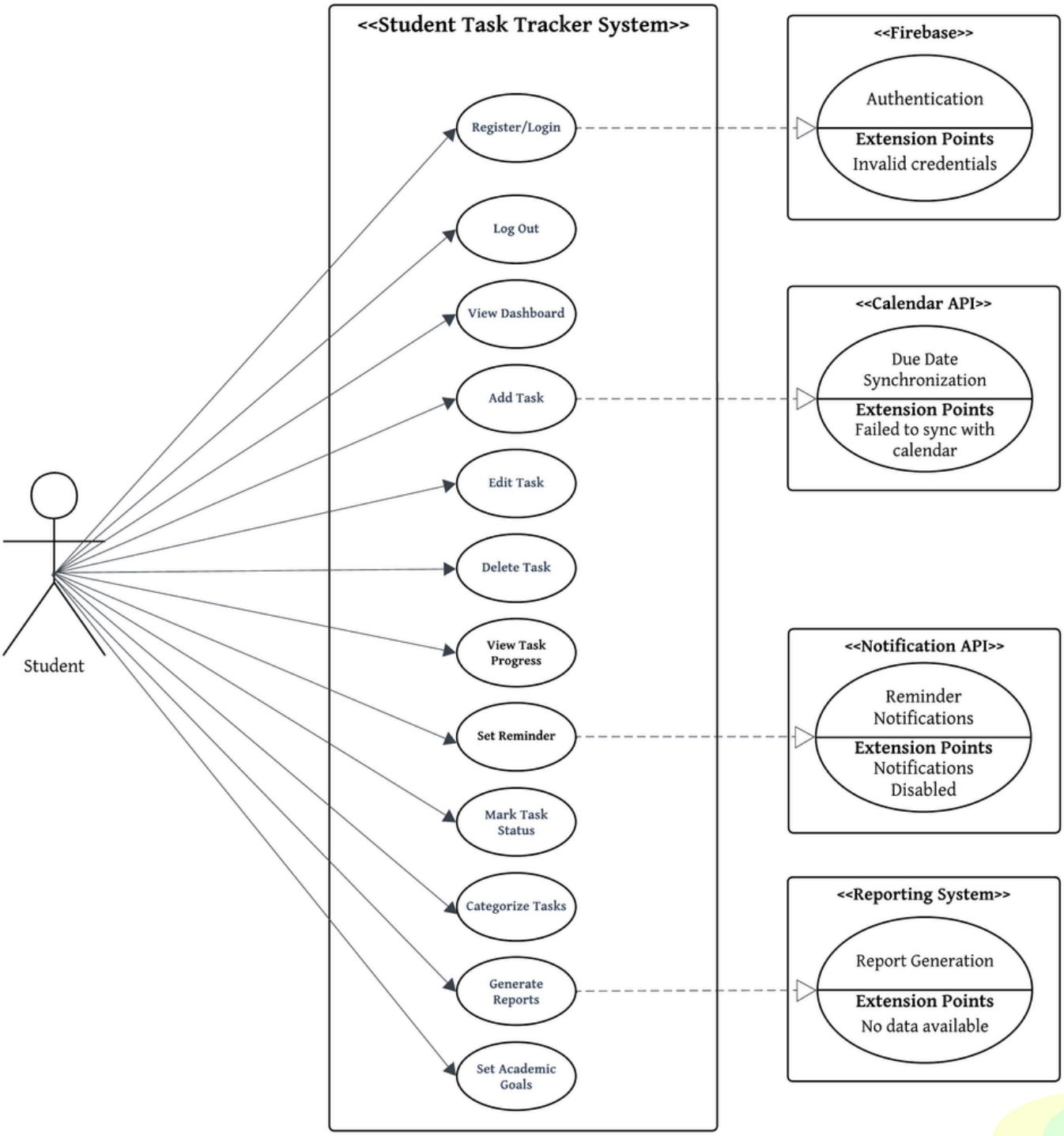
Overall, the StudyMate project successfully meets its objectives and demonstrates strong command of software engineering principles. It offers a complete, reliable academic task-management solution that helps students stay organized, reduce stress, and improve their academic performance. The project stands as a clear example of teamwork, thoughtful planning, and effective application of engineering methodologies.

Appendix

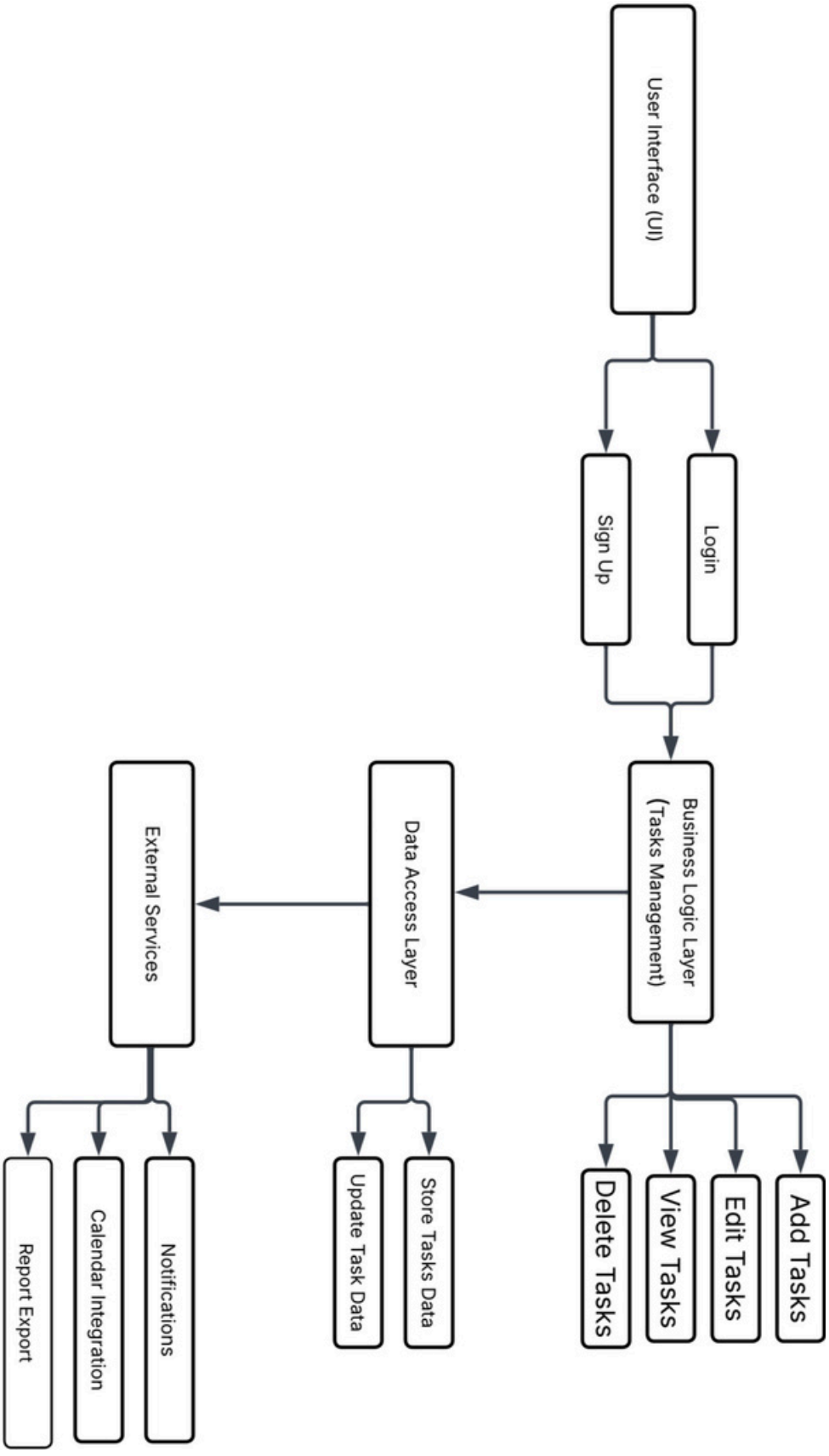
Contributions Table

MemberName	Contribution
Ahad alharbi	Prepared Part A-F including project management, ethics, process model, requirements, and modeling.
Munirah Alharbi	Completed Part B Software Engineering Ethics & Responsibility.
Reef Albarakat	Completed Part C by selecting the Model and defining its application, and also supported the architecture section in Part E.
Remaz Almarzouq	Completed System Overview and Requirements Engineering Plan (Part D)
Leena Almutairi	Contributed to Functional Requirements (Part D) and prepared Class Diagrams in Part E.
Kadi Alseadoun	Contributed to Functional Requirements (Part D) and prepared the Sequnce and State Diagram in Part E. Prepeared the Human interface design
Budur Aljohani	Completed Non-Functional Requirements (Part D) and prepared the Activity Diagram in Part E.
Elaf Aldubayan	Completed External Interface Requirements (Part D).
Thekra Alyahya	Prepared UML: Use Case Diagram (Part D), including layered architecture diagram explanation.
Hessah Alharbi	Worked on Architecture Design (Part E) including structure, layering, and diagram development.
Leena Almutairi Kadi Alseadoun	Report review, proofreading, and editing before submission

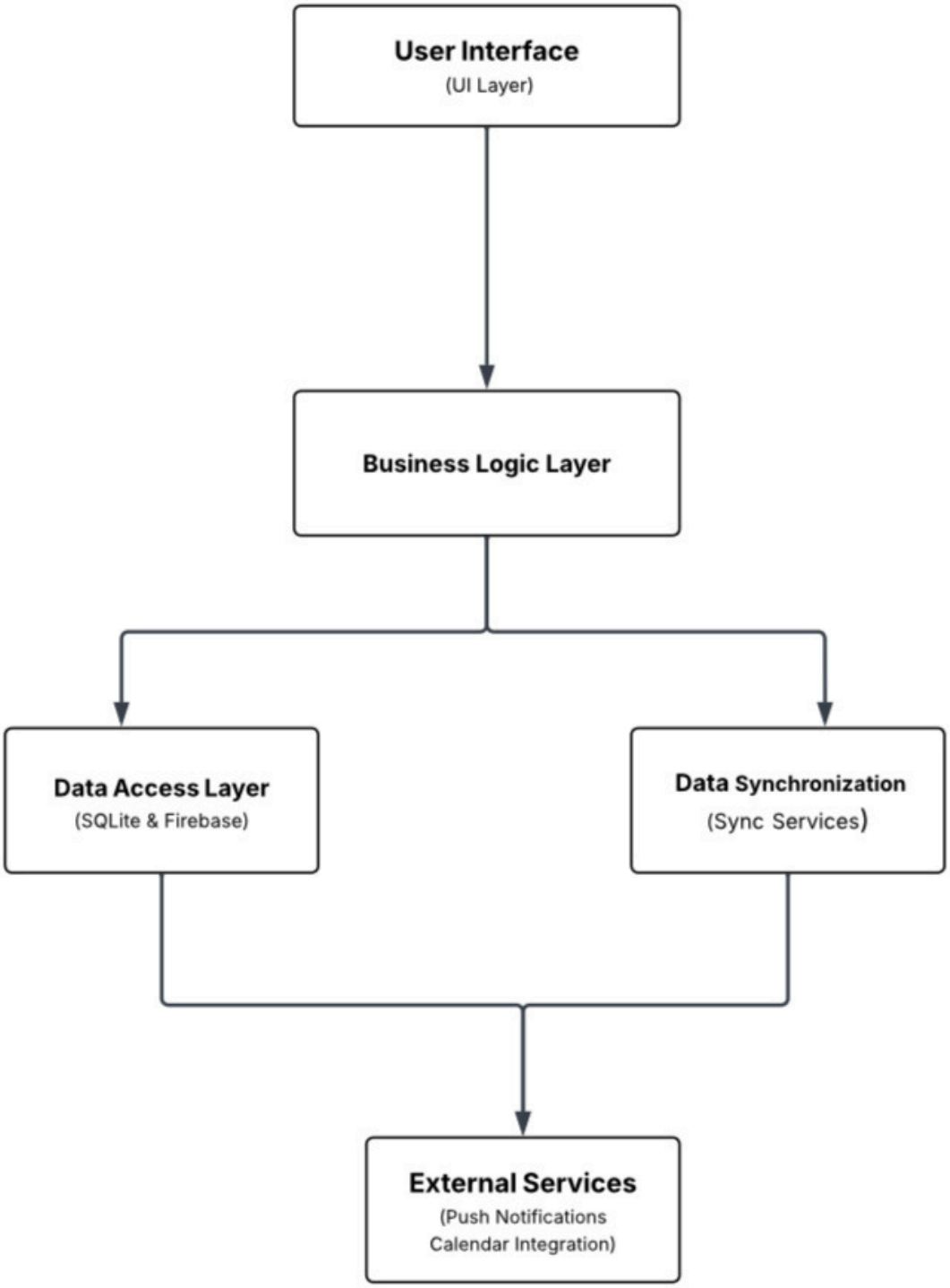
Appendix


<<Student Task Tracker System>>
Register/Login
Log Out
View Dashboard
Add Task
Edit Task
Delete Task
View Task Progress
Set Reminder
Mark Task Status
Categorize Tasks
Generate Reports
Set Academic Goals
Student
<<Firebase>>
Authentication
Extension Points
Invalid credentials
<<Calendar API>>
Due Date Synchronization
Extension Points
Failed to sync with calendar
<<Notification API>>
Reminder Notifications
Extension Points
Notifications Disabled
<<Reporting System>>
Report Generation
Extension Points
No data available

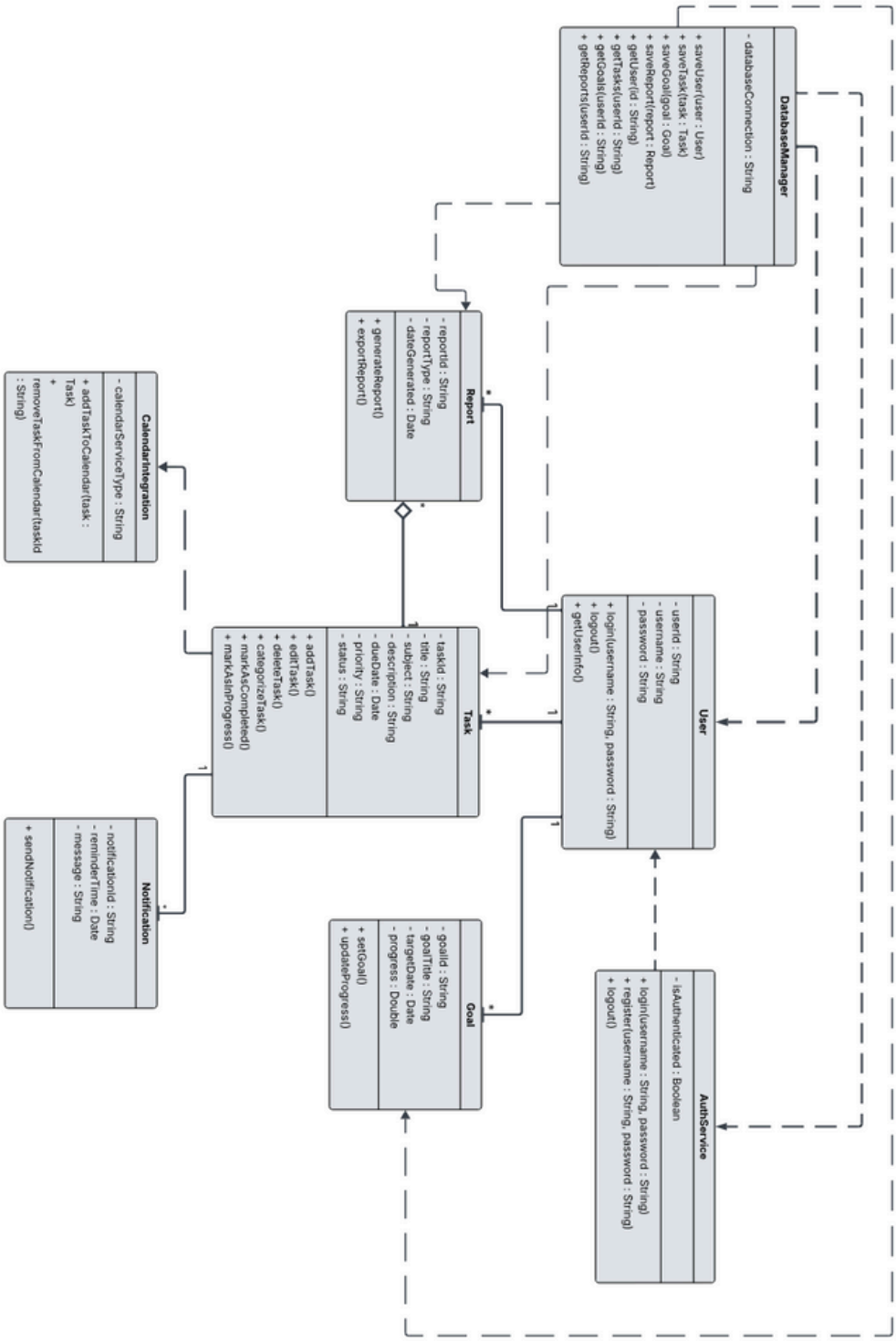
Appendix


User Interface (UI)
Sign Up
Login
Business Logic Layer (Tasks Management)
Data Access Layer
External Services
Add Tasks
Edit Tasks
View Tasks
Delete Tasks
Store Tasks Data
Update Task Data
Notifications
Calendar Integration
Report Export

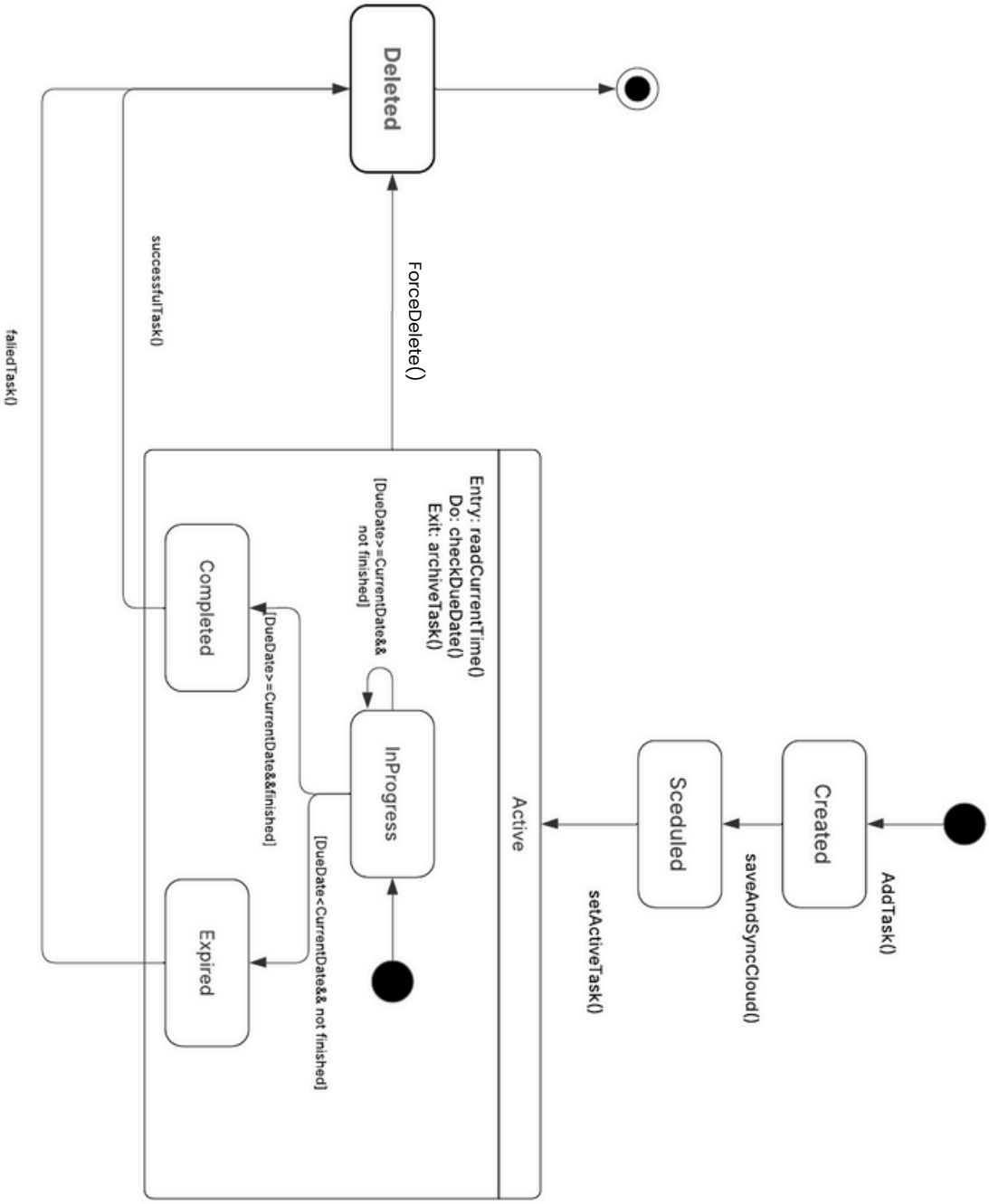
Appendix


User Interface
(UI Layer)
Business Logic Layer
Data Access Layer
(SQLite & Firebase)
Data Synchronization
(Sync Services)
External Services
(Push Notifications
Calendar Integration)

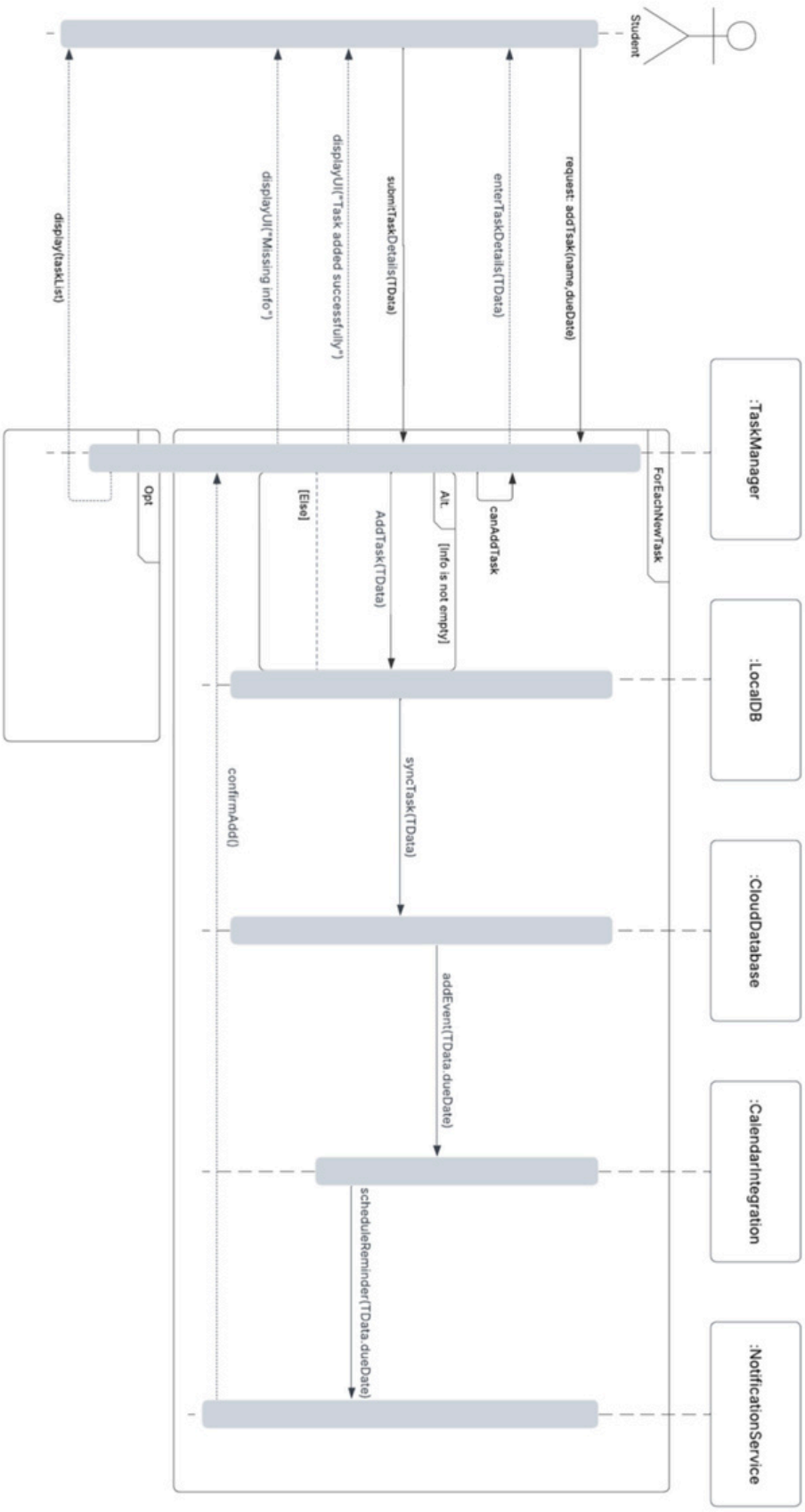
Appendix


DatabaseManager
- databaseConnection : String
+ saveUser(user : User)
+ saveTask(task : Task)
+ saveGoal(goal : Goal)
+ saveReport(report : Report)
+ getUser(id : String)
+ getTasks(userId : String)
+ getGoals(userId : String)
+ getReports(userId : String)
Report
- reportId : String
- reportType : String
- dateGenerated : Date
+ generateReport()
+ exportReport()
User
- userId : String
- username : String
- password : String
+ login(username : String, password : String)
+ logout()
+ getUserInfo()
Task
- taskId : String
- title : String
- subject : String
- description : String
- dueDate : Date
- priority : String
- status : String
+ addTask()
+ editTask()
+ deleteTask()
+ categorizeTask()
+ markAsCompleted()
+ markAsInProgress()
CalendarIntegration
- calendarServiceType : String
+ addTaskToCalendar(task : Task)
+ removeTaskFromCalendar(taskId : String)
Notification
- notificationId : String
- reminderTime : Date
- message : String
+ sendNotification()
Goal
- goalId : String
- goalTitle : String
- targetDate : Date
- progress : Double
+ setGoal()
+ updateProgress()
AuthService
- isAuthenticated : Boolean
+ login(username : String, password : String)
+ register(username : String, password : String)
+ logout()

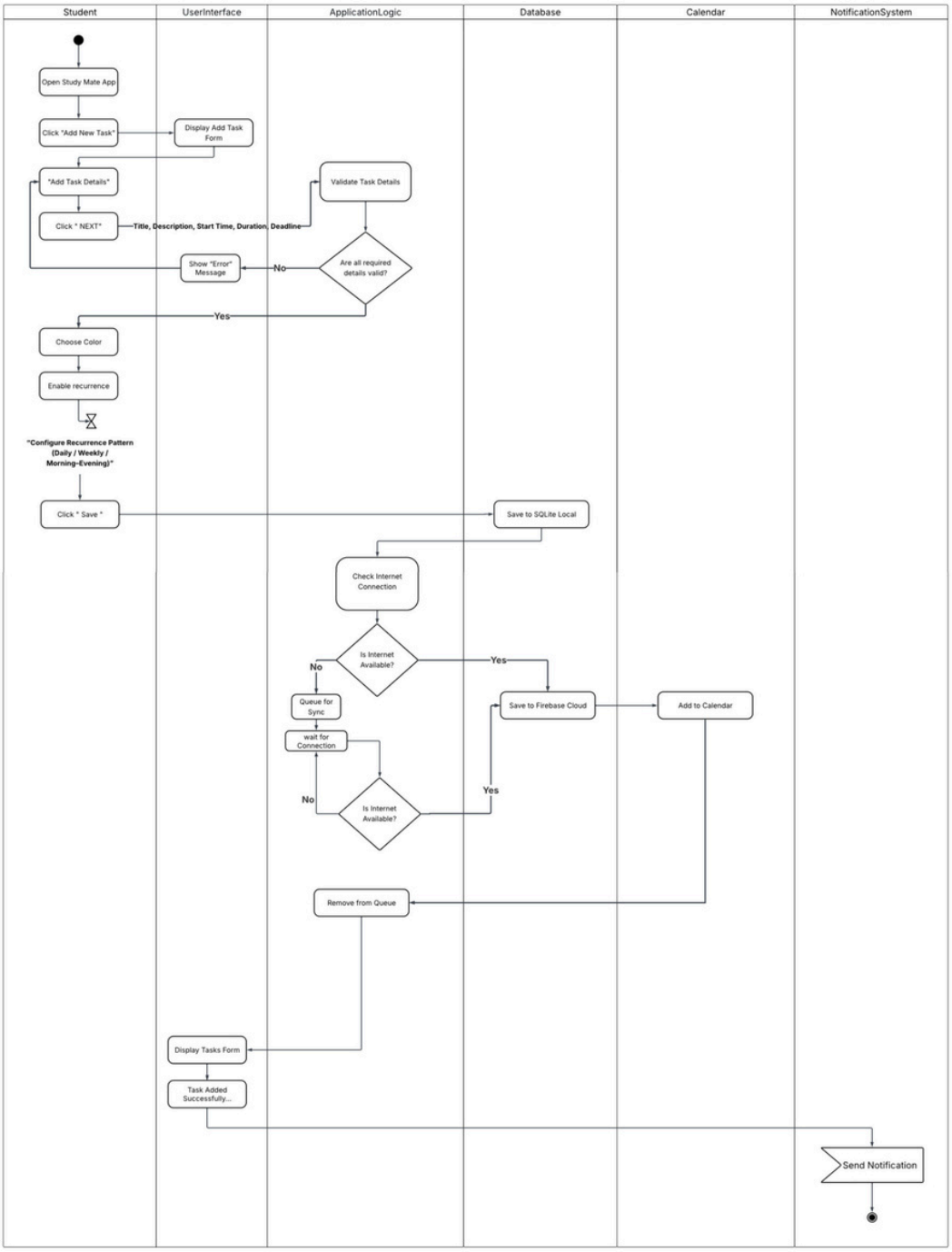
Appendix


Created
AddTask()
saveAndSyncCloud()
Sceduled
setActiveTask()
Active
Entry: readCurrentTime()
Do: checkDueDate()
Exit: archiveTask()
[DueDate>=CurrentDate&&
not finished]
InProgress
[DueDate>=CurrentDate&&finished]
[DueDate<CurrentDate&& not finished]
Completed
Expired
successfulTask()
failedTask()
ForceDelete()
Deleted

Appendix


Student
:TaskManager
:LocalDB
:CloudDatabase
:CalendarIntegration
:NotificationService
request: addTask(name,dueDate)
enterTaskDetails(TData)
submitTaskDetails(TData)
displayUI("Task added successfully")
displayUI("Missing info")
display(taskList)
ForEachNewTask
canAddTask
Alt
[Info is not empty]
AddTask(TData)
[Else]
Opt
syncTask(TData)
confirmAdd()
addEvent(TData.dueDate)
scheduleReminder(TData.dueDate)

Appendix



click to view clearly