Final Project Report

CEN4010.01 ENGR
Software Engineering

Notetaker Multitool Application

Team members:
Juhyung Kim
Solomon Storts
Frank Bafford

Table of Contents

Deliverable I

Objective

Create an Android application that functions as a note-taking multi-tool. Its functions are to provide a basic note-taking app, a planner, and reminders.

Premise

It is an application targeted toward students to help them organize their schoolwork through their mobile device (Android).

Infrastructure

The app will use local storage to store documents and support Android mobile devices within API 20 - 30. The app will be downloaded through the Google Play Store, and wifi will only be required to download the application. A small amount of storage is also needed on the device.

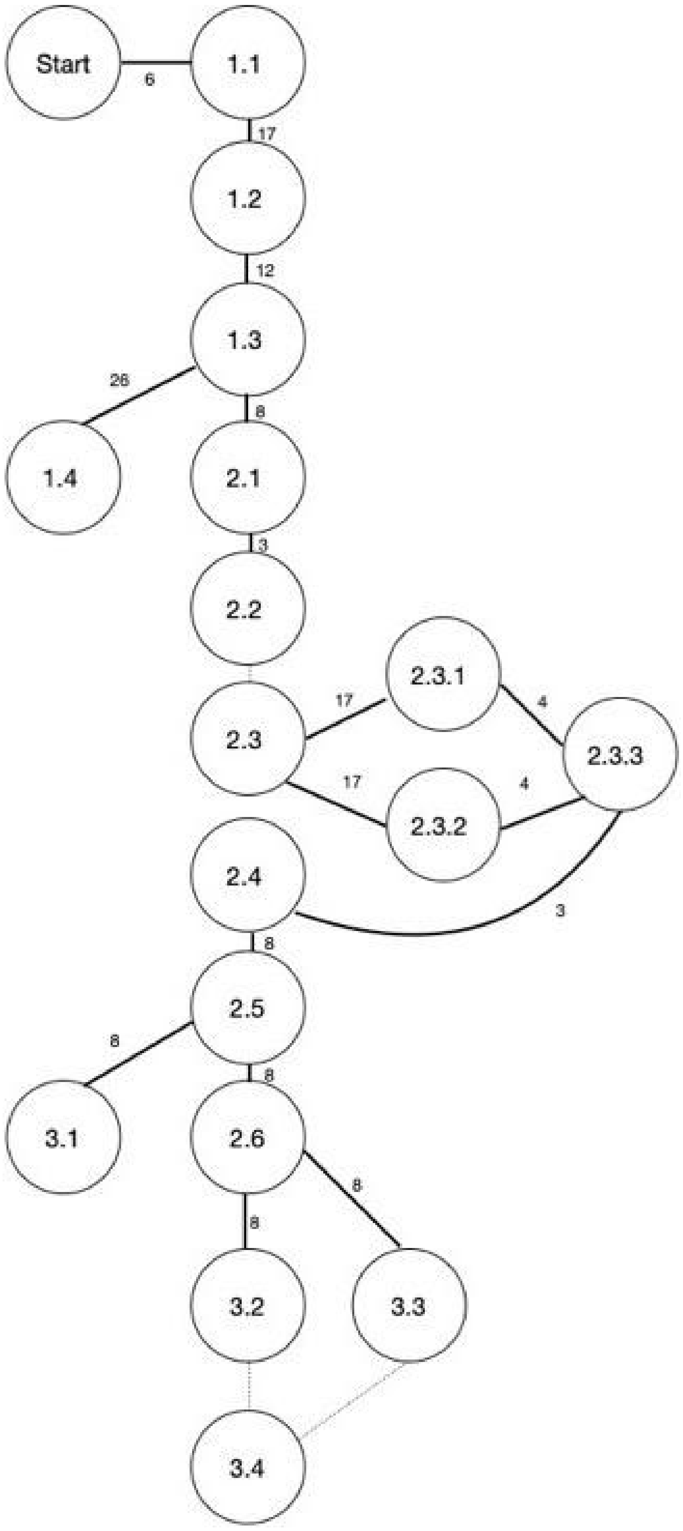
1. Problem/Project Description
    - Our project description is provided above
2. What SDLC Will You Follow and Why?
    - We'll use the Prototyping Model because it will allow us to prototype individual parts of the app and then re-review the specifications to ensure they fit.
3. Project Scope:

    System boundaries, what is wanted from the project:
    - The project is limited to Android mobile devices within API 20 - 30 and will use the Google Play Store and the user's local storage. Other than these, no outside integration will be required.
4. Project Schedule:

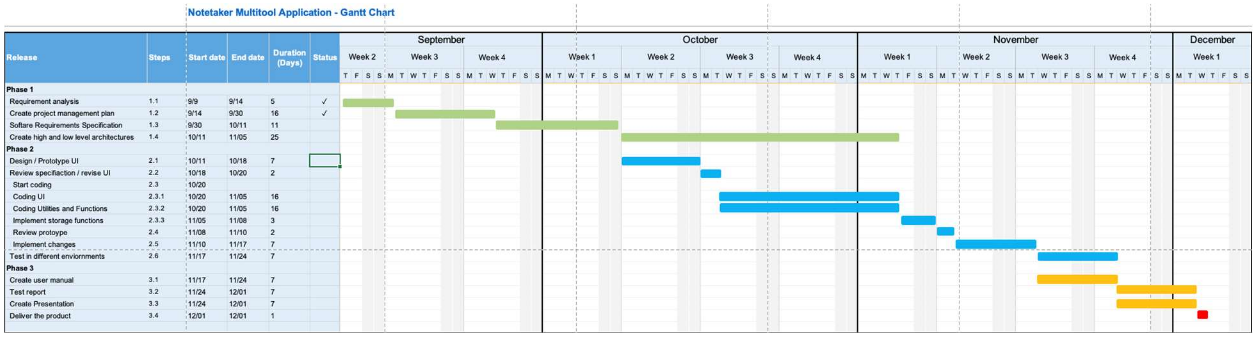
WBS	
Activity	**Estimated time**
Phase 1: Project analysis	
1.1 Requirement analysis	9/09-9/14
1.2 Create a project management	9/14-9/30

plan	
1.3 Provide a Software Requirements Specifications (SRS)	9/30-10/11
1.4 Create high and low-level architectures	10/11-11/05
Phase 2: Development and testing	
2.1 Design/Prototype UI	10/11 - 10/18
2.2 Review specifications / revise UI	10/18-10/20
2.3 Start coding	10/20
2.3.1 Coding UI	10/20 -11/05
2.3.2 Coding Utilities and Functions	10/20 - 11/05
2.3.3 Implement storage functions	11/05-11/08
2.4 Review prototype	11/08-11/10
2.5 Implement changes	11/10-11/17
2.6 Test in different environments	11/17-11/24
Phase 3: Final deliverable	
3.1 Create a user manual	11/17-11/24
3.2 Test report	11/24-12/01
3.3 Create presentation	11/24-12/01
3.4 Deliver the product	By 12/01

- **Activity graph**



- **Timeline Gantt chart**



Notetaker Multitool Application - Gantt Chart

Release	Steps	Start date	End date	Duration (Days)	Status
Phase 1					
Requirement analysis	1.1	9/9	9/14	5	✓
Create project management plan	1.2	9/14	9/30	16	✓
Softare Requirements Specification	1.3	9/30	10/11	11	
Create high and low level architectures	1.4	10/11	11/05	25	
Phase 2					
Design / Prototype UI	2.1	10/11	10/18	7	
Review specifiaction / revise UI	2.2	10/18	10/20	2	
Start coding	2.3	10/20			
Coding UI	2.3.1	10/20	11/05	16	
Coding Utilities and Functions	2.3.2	10/20	11/05	16	
Implement storage functions	2.3.3	11/05	11/08	3	
Review protoype	2.4	11/08	11/10	2	
Implement changes	2.5	11/10	11/17	7	
Test in different enviornments	2.6	11/17	11/24	7	
Phase 3					
Create user manual	3.1	11/17	11/24	7	
Test report	3.2	11/24	12/01	7	
Create Presentation	3.3	11/24	12/01	7	
Deliver the product	3.4	12/01	12/01	1	

5. Project Team Organization:

Who is working on the project and what will they do, team hierarchy, and resource allocation chart:

- Frank - developer, testing
- Solomon - developer, documentation, testing
- Juhyung - documentation, testing

6. Technical Description of the System:

h/w, s/w (compilers special s/w). Restriction on cabling, execution time, response time, security, or any other aspects of functionality or performance:

- There is no restriction on cabling (because our app doesn't require a physical cable connection), execution time, response time (because our app is not a time-sensitive service), or security (because the app will not be using anything such as passwords or sensitive information).

7. Project Standards and Procedures:

Algorithms, tools, review or inspection techniques, design language, coding languages, testing technique:

- Java (Android Studio)
- Wireframe UI Designs
- Debugging with logcat

8. Quality Assurance Plan:

Reviews and inspections to meet customers' needs:

- We are students who would need an app like this, so if it can work for our needs, we know it will work for other students as well. Our testers and inspectors are also students, so we know that we will provide a product for the needs of our audience.

9. Configuration Management Plan

- N/A - For this project, a configuration management plan is not needed because this is not a very big project, and most of the things covered in that document we are doing are in the form of deliverables. Also, most of the information provided in that document, based on sample documents from NASA, will not be required for this project.

10. Documentation Plan:

What documents will be produced, by whom, and when:

- The documents that will be produced during this project will be the deliverables due throughout this semester. We will all make these documents together, which will be worked on starting 1-2 weeks before the due date.

11.Data Management Plan:

How data will be gathered, stored, manipulated, and archived:

- Data will be collected and stored through emails, Discord, or Microsoft Teams. We will be manipulating the code on Android Studio on our personal computers, and we will also archive them on our personal computers. We also have the option of keeping them on emails, Discord, Microsoft Teams, and USB Drives.

12. Resource Management Plan:

Managing physical resources:

- Part of our resource management plan is a Gantt chart. This has a timeline of what we will do and our assignments to complete the product on time. The only resource we will need is a computer and Android Studio installed on that computer.

13. Test Plan:

What and how tests will be run. How the program modules will be tested and how the test data will be generated:

- The tests will be run inside of Android Studio using the debug feature and logcat. With this, we will be able to view all potential bugs, errors, and warnings that may show up during testing and document them. We will also be testing the app on different APIs to see if the software is consistent between our set range of APIs.

14. Training Plan:

How training will be conducted, the expertise required, documents and manuals to be produced:

- Developer - We do not need a training plan because we are already knowledgeable with Java and were already given the necessary training to complete this project.
- Customer - We will be providing a user manual as a deliverable and an in-app feature that will show the user everything they need to know to use the app.

15. Security Plan:

Secure the system (e.g. password management):

- The system will be secured with the user's password to get into their phone. To get into the app, the phone will need to be unlocked, and the app just needs to be selected. The app can also be protected with whatever security protocols that the user puts in place to get into their phone. There will be no additional protocols in the app because nothing such as passwords will be required.

16. Risk Management Plan:

- Risk Identification: The risks that may occur during this project are constraints on time because of other projects we are working on, functions of the app not working for unforeseen reasons, or not having enough time to finish the project with all of our classes and projects members of the group are doing.
- Risk Responsibilities: The responsibility of managing all of the risks that may occur is given to all members of the group.
- Risk Response: If we are ever met with a risk, we will have a meeting about what should be done and fix our schedule accordingly so that we will be able to finish the project on time and get everything that we need to finish by our deadline.
- Risk Mitigation: To mitigate these risks, we are doing extensive planning and organizing what we need to do and when. By organizing our time, we can reduce the risk of being busy with other projects and not having enough time to complete a task on schedule. We can also mitigate the possibility of our app not working by doing efficient and thorough testing while setting the API in a wide enough range that will work with a majority of phones used today.

17. Maintenance Plan:

Responsibility for changing code while maintaining the system, repairing hardware, and updating the training materials:

- N/A - We will not be maintaining the app after we finish the product, so there is no need for us to have a maintenance plan for this deliverable

Deliverable II - Software Requirements Specification

1. Introduction

1.1 Purpose

- The purpose of this document is to provide a detailed description of the Notetaker Multitool Application. This document will be a comprehensive guideline for both users and developers. It contains the description of the project, external interface requirements, system features, and other non-functional requirements.

1.2 Document Conventions

- This document follows the IEEE standard for System Requirements Specification.

1.3 Intended Audience and Reading Suggestions

- Users who want to learn about this program. Casual users may be interested in reading the Overall Description and System Features, and more advanced users can explore more in-depth about the project.
- Testers who need to know about the requirements and features of the system.
- Developers who are interested in developing the program further.
- Customers are the students to whom we will be presenting the project.

1.4 Product Scope

- Notetaker Multitool Application is an Android application that provides a basic note-taking feature, as well as a planner and reminders. It will remove the inconvenience of using separate applications for each feature.

1.5 References

- GitHub page: https://github.com/solomonstorts/SoftwareEngineeringProject
- IEEE Template for System Requirement Specification Documents: Provided by the professor.

2. Overall description

2.1 Product Perspective

- The product we are developing is a follow-on member of a product family. There are currently apps that are diaries or just plain notetakers that are geared toward general use, but we want our product to be tailored toward student use. Since students need a more organized tool for note-taking that other products do not provide, our product chooses to solve this problem by making note-taking more organized for students.

2.2 Product Functions

- Save and delete notes taken in local storage
- Organize notes in folders

2.3 User Classes and Characteristics

- We expect users of this app to have some sort of education since this app is being developed for students. We can also expect workers to use this app if their jobs require note-taking and organization.
- To use this app, users will need to have the basic knowledge of how to navigate and use a smartphone. The app will be designed to have a similar UI and flow used in most apps on an Android system.
- The most important user class will be students. With the use of an organization service that would favor note-taking for different classes, we want this user class to be the most important. The least important users are workers who require note-taking because other products are tailored toward them with features more favorable to their needs.

2.4 Operating Environment

- This software will operate on modern Android mobile devices. Our target API will be API 25 with a minimum API level of 20 and a maximum of 30. The app won't require internet access to use the app.

2.5 Design Implementation Constraints

- Some constraints that may limit the options available to use are:
  - Hardware Limitations - storage space and memory
  - API Level
  - Unexpected bugs when storage is full

2.6 User Documentation

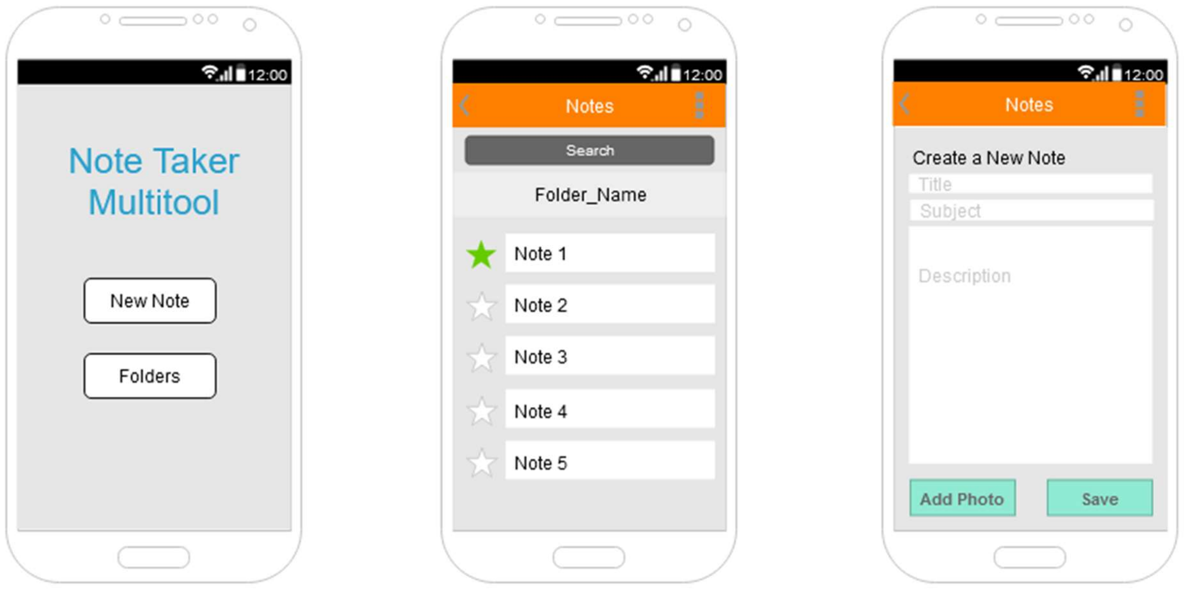
- Brief in-app tutorial

2.7 Assumptions and Dependencies

- Since the scope of this project is very small and simple we don't have any assumed factors. We are not using third-party or commercial components, and the only external factors that affect dependencies for this project will be the API level of the phone that the customer will be using.

3. External Interface Requirements

3.1 User Interface



3.2 Hardware Interface

- Android Mobile Device
  - The hardware that will be needed to use the app will be that of Android mobile devices. Our app is programmed to work for these types of devices only so it cannot be used on hardware that is provided by Apple devices.
- External Storage
  - With the app having the capability to store data on the device there will also be functions to store data in external storage such as an SD card if the model of the device allows the customer to use this storage component.

3.3 Software Interface

- This software will be installed on Android devices and their respective operating system and will only need to be able to write and save data to local storage as .docx files or .pdfs. No internet connection will be required so no software will be needed to store any data in a database. Only basic internet functions from the device will be needed to download the application from the App Store.

3.4 Communication Interface

- The only communication that will be required for this app is an internet connection for installation. The communication protocols that will be used are the same protocols that are needed to use the Play Store. The data transfer rate will vary based on internet connection speed.

4. System features

4.1 Taking Notes

4.1.1 Description and Priority

- The app will provide an empty canvas for writing down notes, using the mobile device's keyboard. This is a high-priority feature.

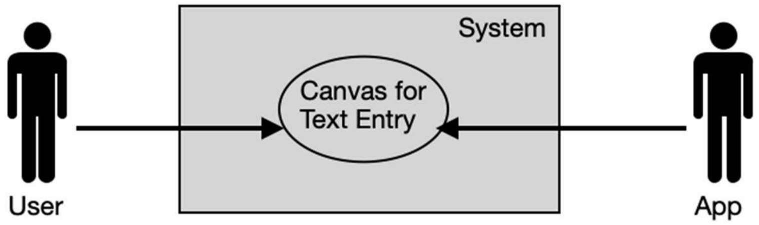
4.1.2 Stimulus/Response Sequences

- System provides a blank canvas
- User clicks on a section of canvas to begin typing
- System provides a keyboard for text entry

4.1.3 Functional Requirements

- REQ-1: Must be able to accept any form of alphanumeric input
- Images, multiple fonts, and different colors support TBD

4.1.4 Use Case Diagram



4.2 Saving Notes

4.2.1 Description and Priority

- The app will enable users to save and retrieve their notes to/from local storage. This is a high-priority feature.

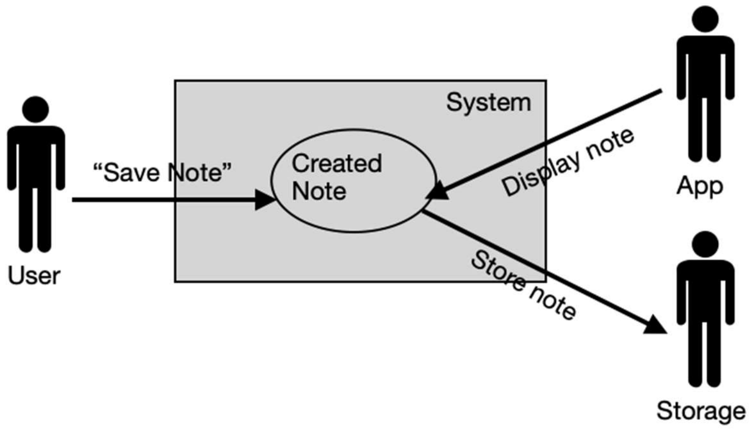
4.2.2 Stimulus/Response Sequences

- After creating a note, the user presses the "Save" button
- System stores the data in local storage
- For retrieval, the user selects their note from system files
- System displays notes in the editor for analysis from the user

4.2.3 Functional Requirements

- REQ-1: Must be able to store any document created by functional requirement 1
- REQ-2: Must provide a user interface for selecting a storage location
- REQ-3: Must provide a user interface for selecting a document from storage
- REQ-4: Must accurately retrieve any previously saved documents
- Maximum document size TBD

4.2.4 Use Case Diagram



4.3 Creating planner events

4.3.1 Description and Priority

- The app will allow users to create notes for a planner, by attaching a date to a note. This is a medium-priority feature.

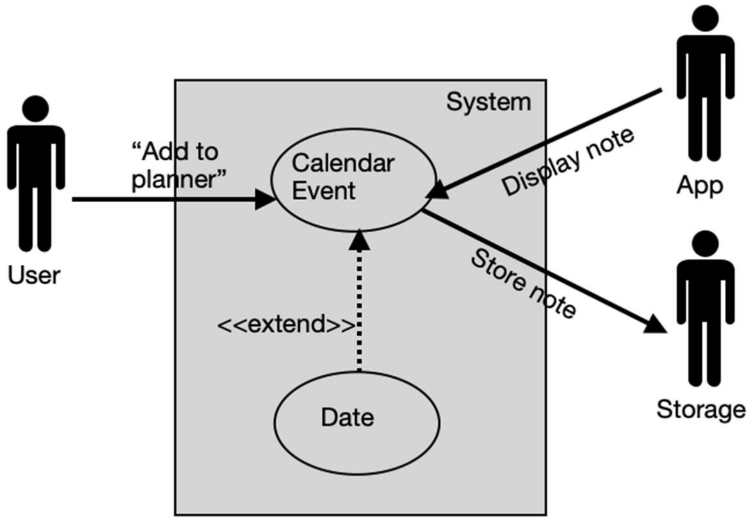
4.3.2 Stimulus/Response Sequences

- After creating a note, the user presses the "Add to Planner" button
- System provides a user interface for date selection
- System stores notes, as well as the associated data in the storage
- System provides a user interface for viewing/editing notes with dates attached (planner events)

4.3.3 Functional Requirements

- REQ-1: Must be able to attach and store associated dates with notes
- REQ-2: Must provide a user interface for viewing/editing planner events

4.3.4 Use Case Diagram



4.4 Creating reminders

4.4.1 Description and Priority

- The app will allow users to create reminders, by attaching the time and date that it will be due. It will then allow users to mark a reminder as “completed” once they have finished with it. This is a medium-priority feature.

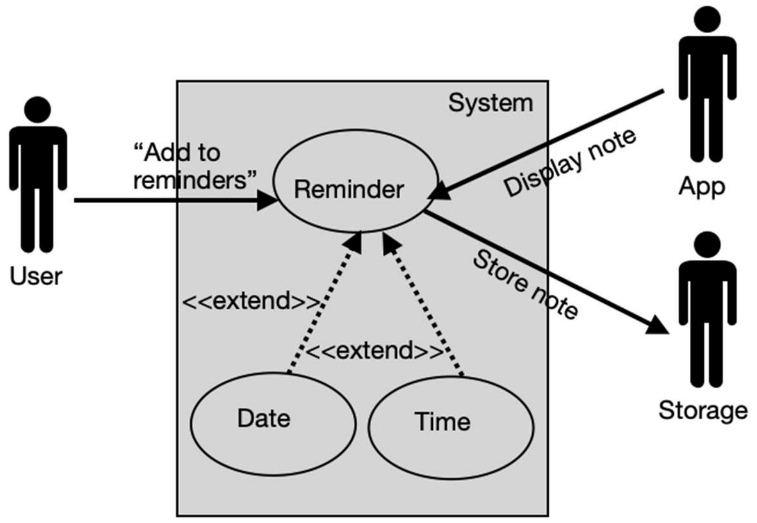
4.4.2 Stimulus/Response Sequences

- After creating a note, the user presses the “Add to Reminders” button
- System provides a user interface for date and time selection
- System stores note, as well as the associated date and time in storage
- System provides a user interface for viewing, editing, and marking reminders as completed

4.4.3 Functional Requirements

- REQ-1: Must be able to attach and store associated dates and times with notes
- REQ-2: Must provide a user interface for viewing/editing/completing reminders
- Sorting features TBD

4.4.4 Use Case Diagram



5. Other Nonfunctional Requirements

5.1 Performance Requirements

- Under 50 MB (TBD) of storage is required to download the application, and additional storage space will be needed as the user saves new files.
- This application is a very lightweight program, so most devices will run without any performance issues.
- This application supports Android mobile devices within API 20 - 30.

5.2 Safety Requirements

- Unusual usage of the program such as a frequent forced stop of the program should be avoided by the users since it may increase the chance of file corruption.
- This application does not provide extra security features such as password lock, so users must employ personal security measures such as locking the phone.

5.3 Security Requirements

- This program needs permission to access the local storage of the mobile device. Once the permission is granted by the user, they can use every feature of the program.

5.4 Software Quality Attributes

- Notetaker Multitool Application provides note, planner, and reminder features to keep the user organized yet it is easy to use. The focus of this application is ease of use, flexibility, portability, and reliability. Anyone who can read and write will be able to use this application.

5.5 Business Rules

- This application is designed for personal use by individuals. The users can use the features of this application in any way that suits their needs. If the user has any type of storage device that we did not anticipate to be used, the user may still be able to use that function because of native Android protocols.

Appendix A: Glossary

- Alphanumeric: Text that includes both letters and numbers.
- Canvas: The area for text entry where users can select and edit the text using the built-in keyboard from their mobile device.
- Mobile Device: Any device that runs the Android mobile operating system, and that will run the app.
- Note: A text file that can be saved, viewed, and edited by the user.
- Planner Event: A note that has an associated date with it.
- Reminder: A note that has an associated due date as well as a time with it.

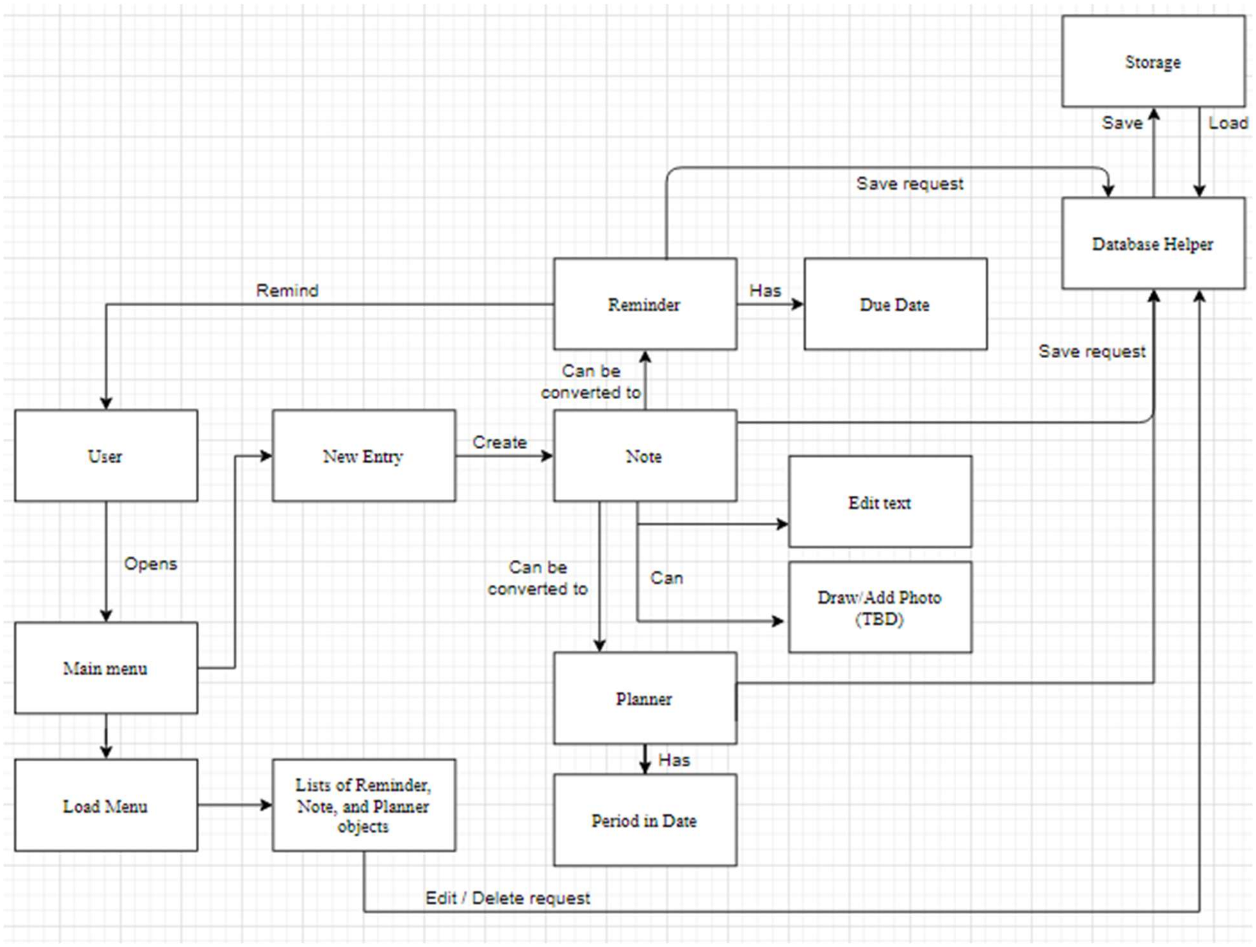
Appendix B: To Be Determined List

- Image attachment, multiple fonts, and different colors support
- Maximum document size
- Sorting features for reminders
- Calculator provided within the app

Deliverable III - Architecture Styles

High-level Architecture:

- Provide a diagram of the major components of your system



- Describe each of the major components given in the diagram

1. User - The person who uses this application.
2. Storage - The local storage of the user's device.
3. Date - An object used to contain the time data.
4. Note - An object that can store the text entry from the user. It can be saved in Storage with the help of Database Helper.
5. List - The list of saved Notes, Calendar Events, and Reminders. The RecyclerView from the Android API will be used to display the items.
6. Reminder - It extends from Note by adding a due Date on it. It will remind the User about the event when the due date arrives.
7. Planner - It extends from Note by adding the start time and end time using the Date.

8. Database Helper - This is the feature that saves the Notes, Calendar Events, and Reminder to the Storage.
9. Text Entry - The text input from the User.
10. Create - The feature to create Notes, Calendar Events, and Reminder so the User can start adding the Text Entry.
11. Delete - The feature to delete the saved Notes, Calendar Events, and Reminder.
12. Edit- The feature to delete the saved Notes, Calendar Events, and Reminder.
13. Remind - The feature that notifies the User when the saved Reminders hit their due date.

- Consider the various architectural styles discussed in the course. Do any of them apply? Make sure to explain the reasoning of your choice of architectural style(s) within the context of the project

Client-Server

- This application only utilizes the device's local storage and it does not communicate with a server. So, the Client-Server architecture style is inappropriate.

Peer-to-Peer

- As discussed previously, this application does not support connections with external entities. So, the Peer-to-Peer architecture style is not a good choice.

Publish-Subscribe

- This application neither provides data to others nor requires data from others. So, the Publish-Subscribe architecture style will not work.

Repositories

- The repositories architecture will not work for the same reason as the Publish-Subscribe architecture's reason.

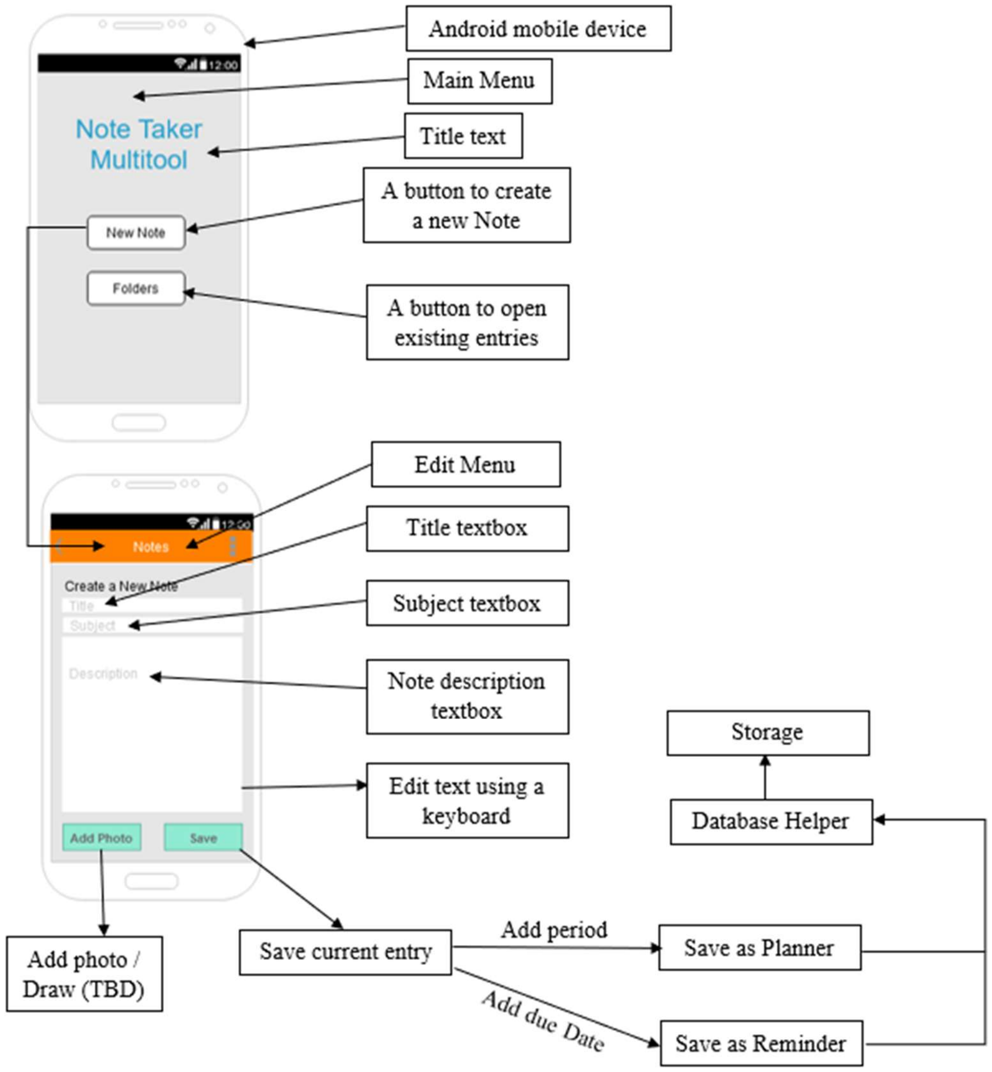
Layering

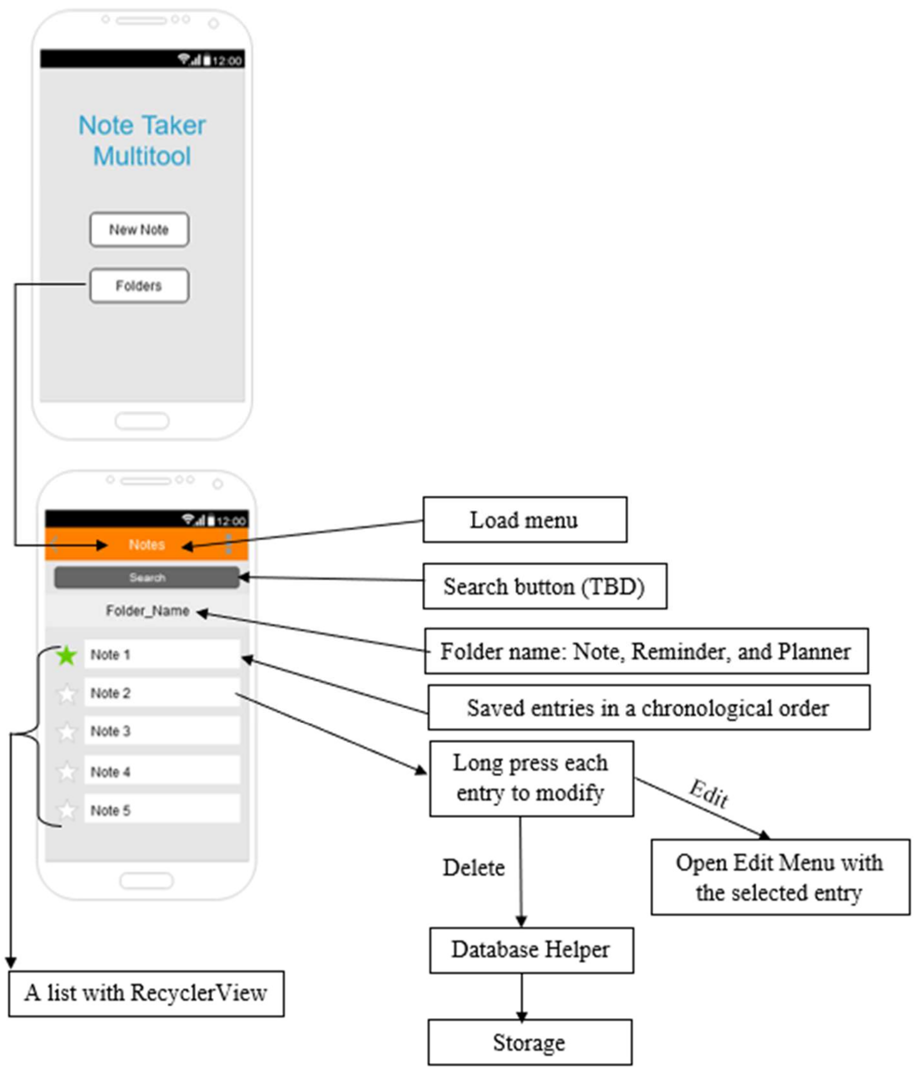
- In the Layering architecture style, components are organized in layers so each component can only communicate with the adjacent ones. It's a great architecture for systems that need to work procedurally like a package delivery system. However, the flow of the data is not so procedural in this application so the Layering architecture is not the most suitable one.

Pipe-and-Filter

- The Pipe-and-Filter architecture style is the most suitable style for this application. This style can accurately depict the transformation of data and show how components communicate with one another.

- Provide a diagram of the system interfaces. Describe each of the interfaces of your system. How will the major components interact with each other?





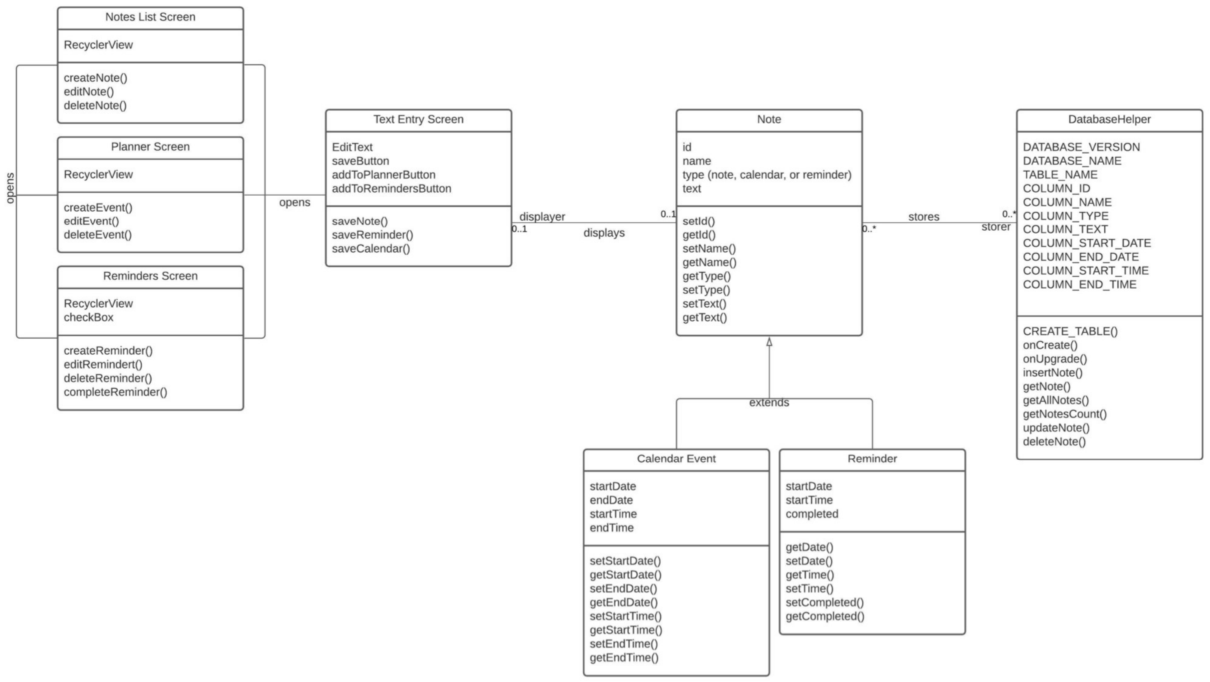
- Discuss your team's evaluation of the major design issues: reliability, reusability, maintainability, testability, performance, portability, security, and safety.
  - Reliability
    - We would need to make sure the application effectively saves the user's notes in the right places and can retrieve that information reliably.
  - Reusability
    - Since our app will be using an SQL database, we will be able to store and pull information on an Android device, but we will not be able to do this for any other device. If we want to expand the application to different mobile operating systems, we need to recode everything for the targeted device.
  - Maintainability
    - N/A We will not be maintaining this application after the product is finished.

- Testability
    - Some issues we have with this is that we will be able to test our devices in an emulator, but we will not have access to do testing on a large number of real Android devices. Instead of local storage in a mobile device being used, we will be using storage from our computers. If problems occur because of this, we will have no way of knowing.
- Performance
    - One issue with performance is that we will be testing our application on an emulator instead of a mobile device. Our computers are more powerful than mobile devices so we may have issues when testing this that are hidden.
- Portability
    - Our application will only be able to be used on Android devices so there is no way to use the application on different devices. However, if those devices support text files then the other phones will be able to view files created in the application.
- Security
    - If there is a security problem with the user's phone causing it to be unlocked or compromised, there are no security measures in place on our side. With this, if something were to happen, the intruder would have access to their notes if they had a way of finding it.
- Safety
    - N/A None of our functions for this application will be able to cause harm to anyone else unless there is confidential content inside their notes. This issue would fall into security.

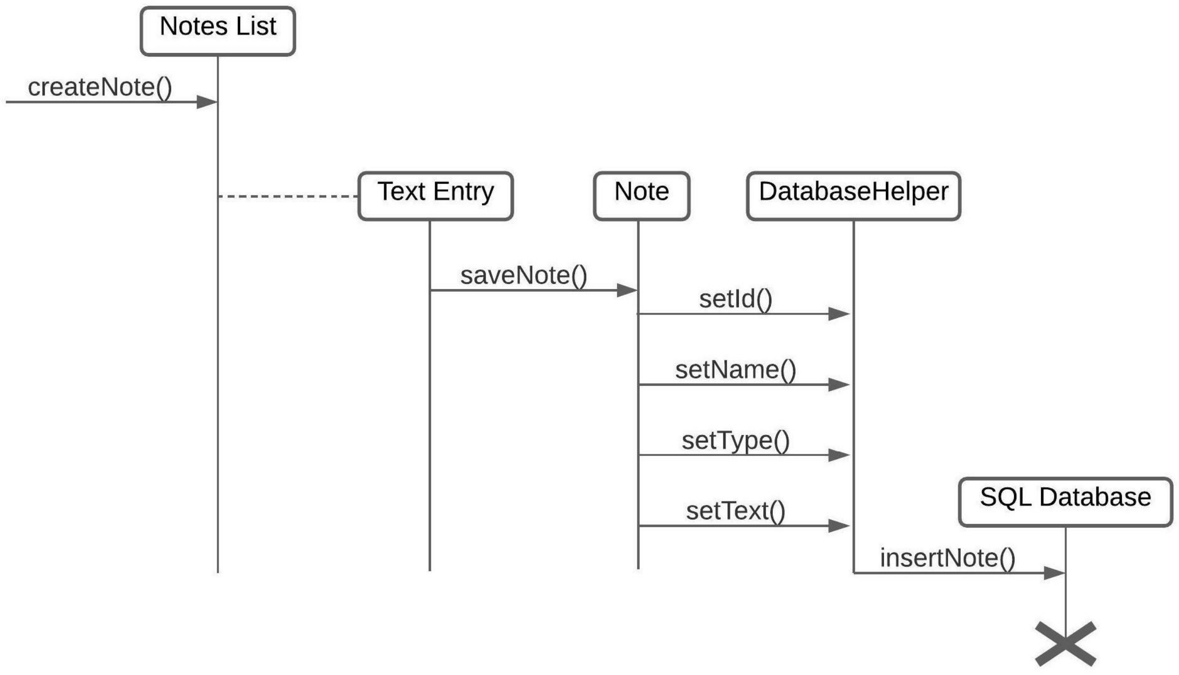
- Which issues are relevant to your project?
    - Reliability, Reusability, Testability, Performance, Portability, and Security
- What prototypes (if any) will you need to do to evaluate alternative design strategies?
    - We won't really need to make prototypes to evaluate alternative design strategies. Inside our IDE, we can just create a different version of a class with a different name, and we will be able to do testing from that screen specifically instead of creating different prototypes. We can also use GitHub to save and test different application versions.
- What technical difficulties do you expect to encounter? How will you solve them?
    - We expect issues to occur if the user duplicates a name to store a file. This will cause the user to overwrite a note that they have made previously unintentionally. To mitigate this, we will place a check in place to notify the user to change their file name or overwrite the previous note they have stored.

Lower-level Architecture:

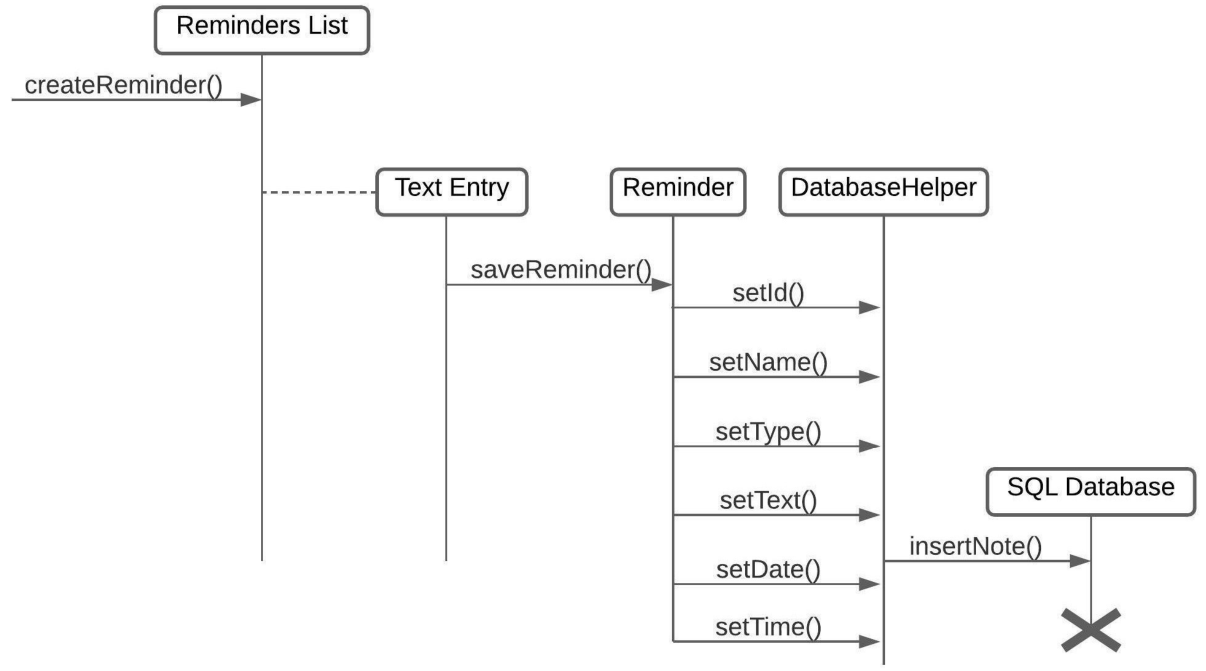
- **UML Class Diagrams**



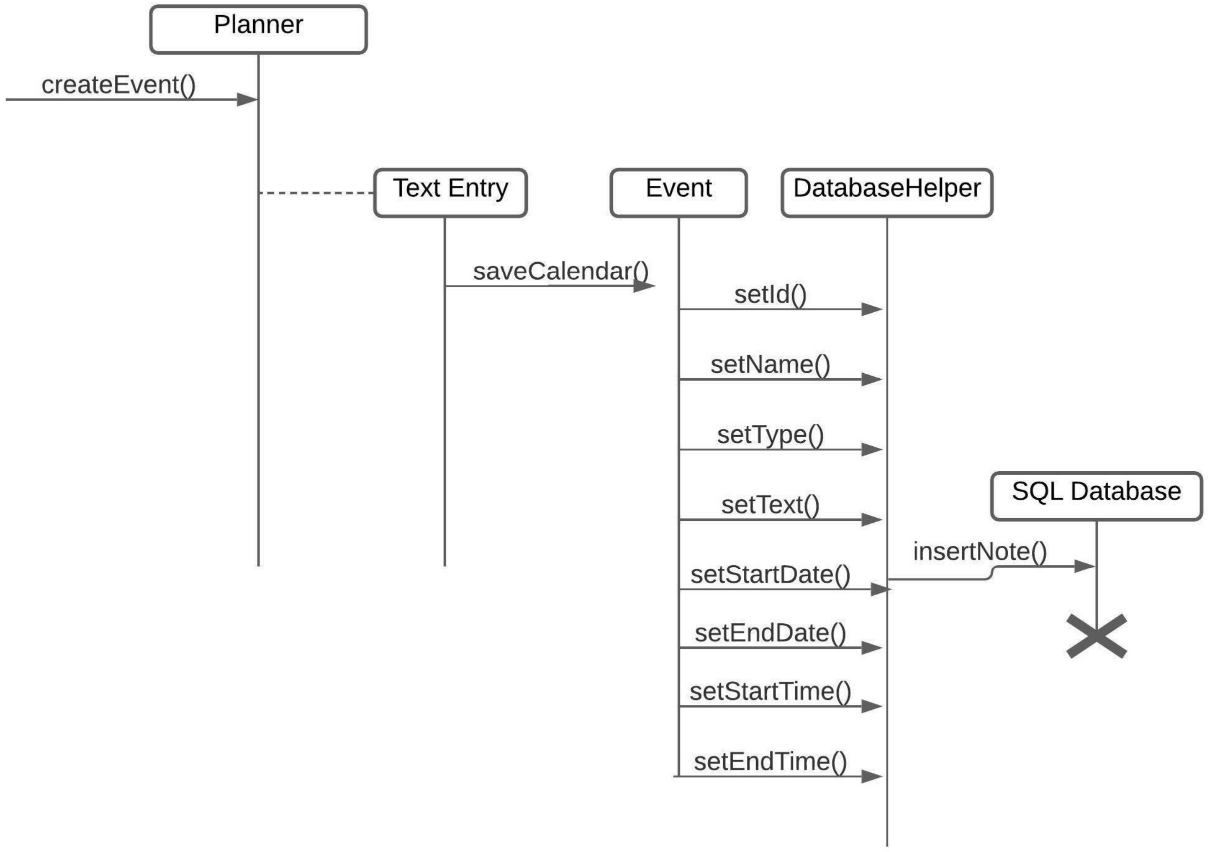
- **UML Message Sequence Charts**
  - Create Note



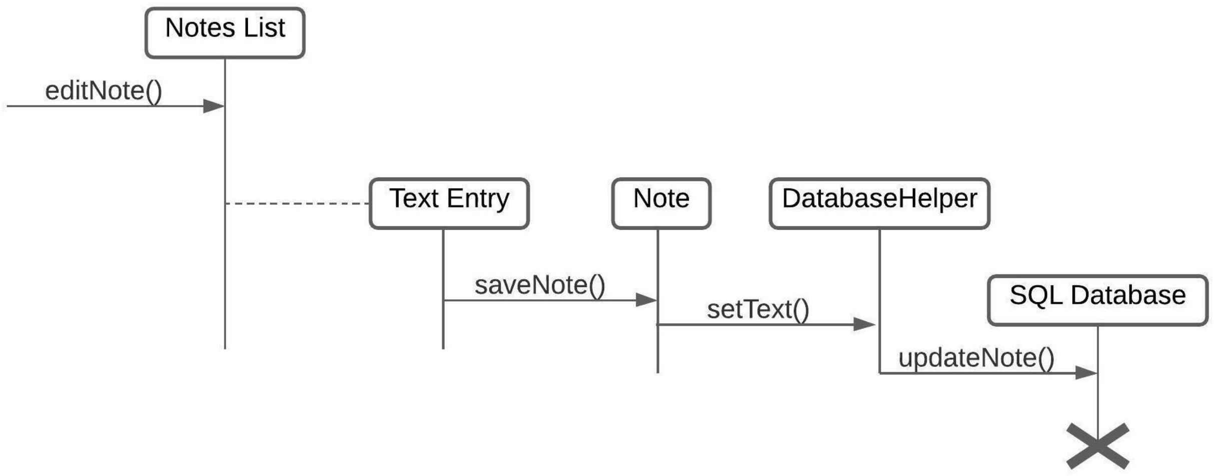
  - Create Reminder



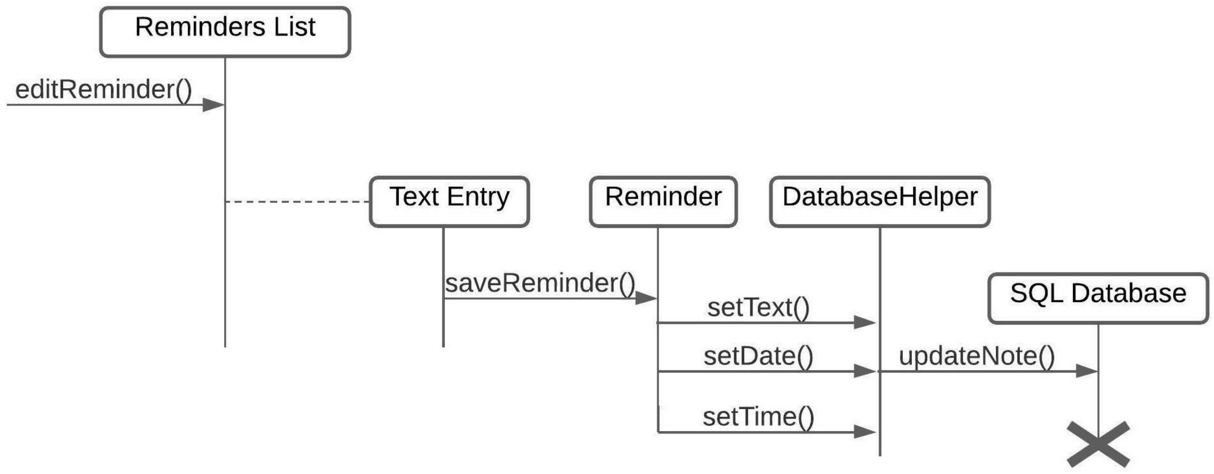
- Create a Planner Event



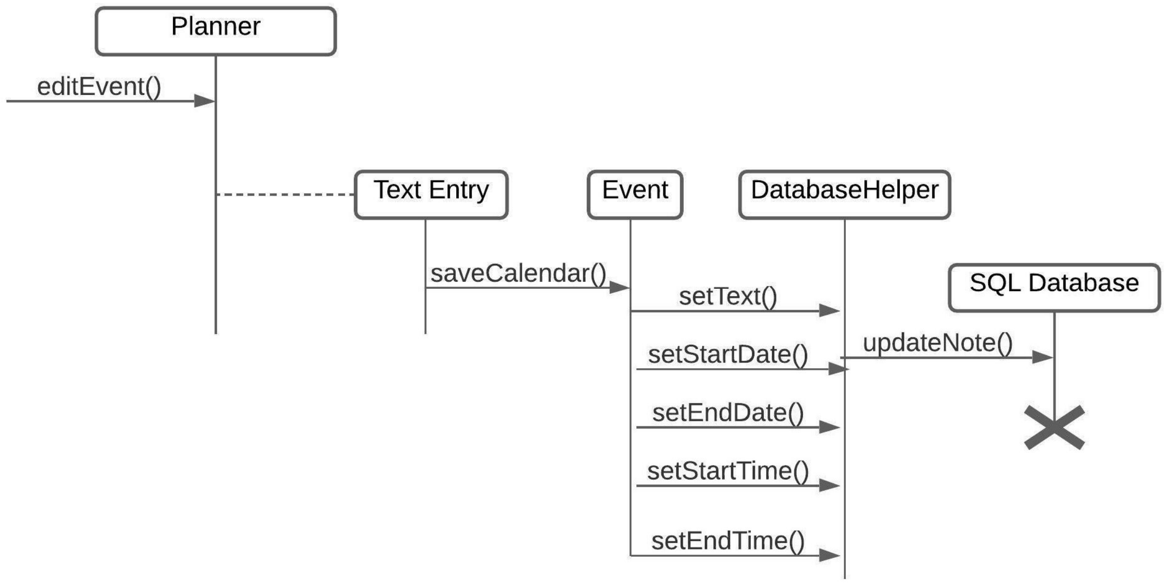
- Edit Note



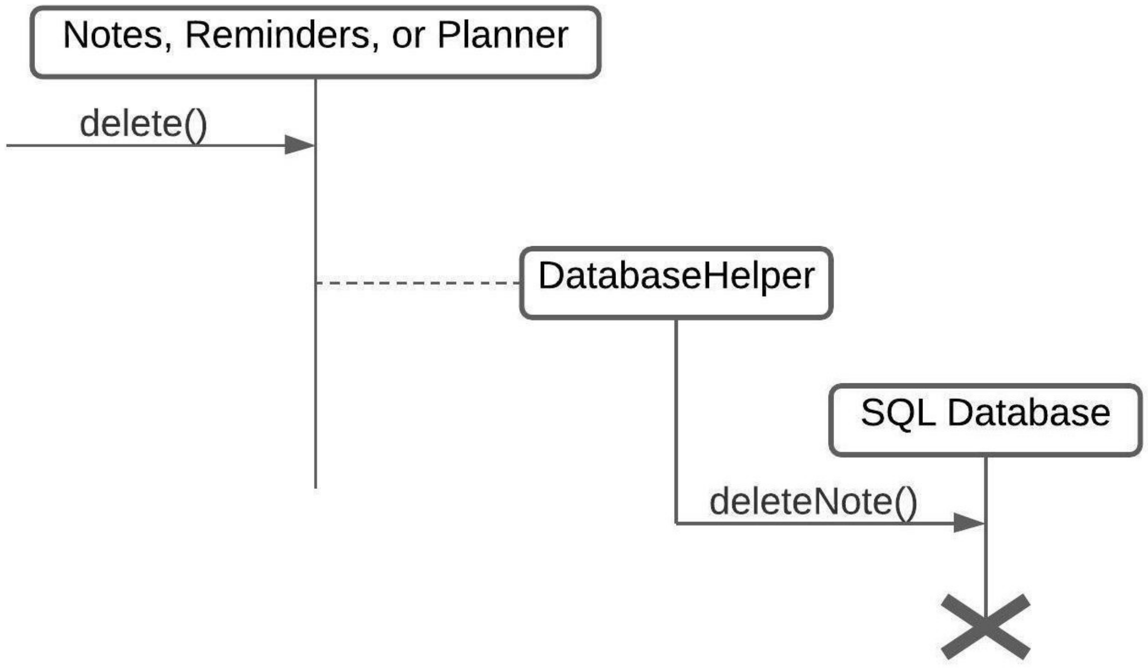
- Edit Reminder



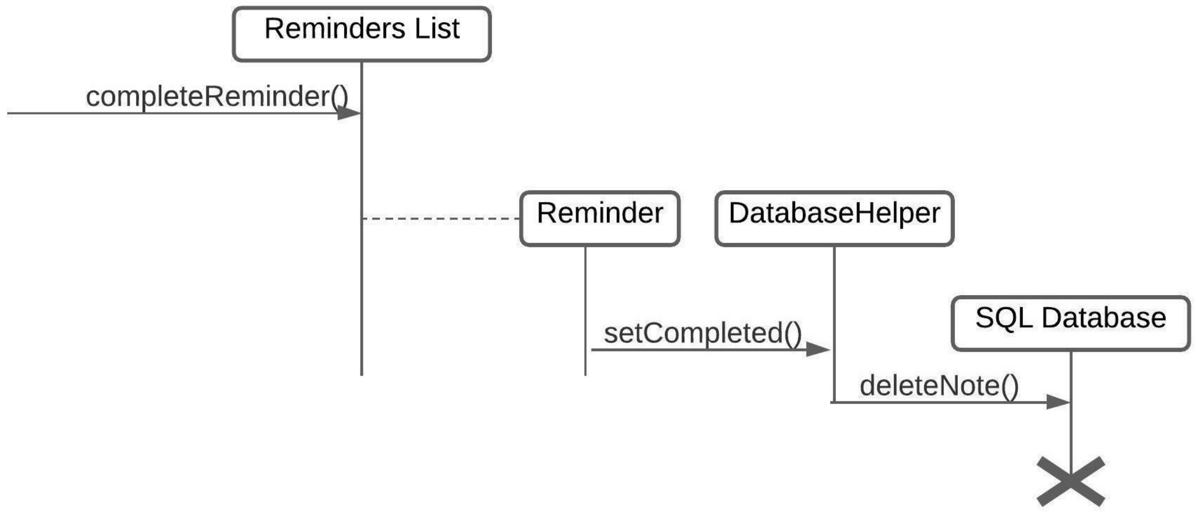
- Edit Planner Event



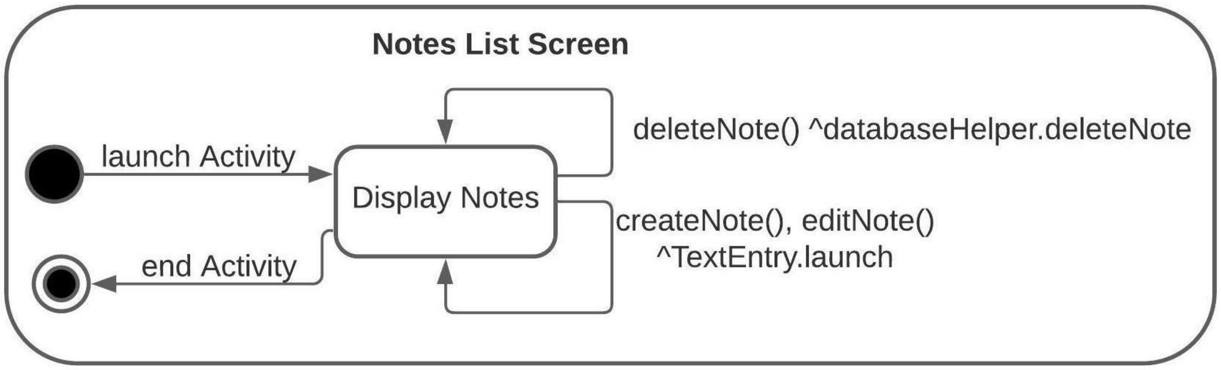
- ○ Delete Note, Reminder, or Planner Event



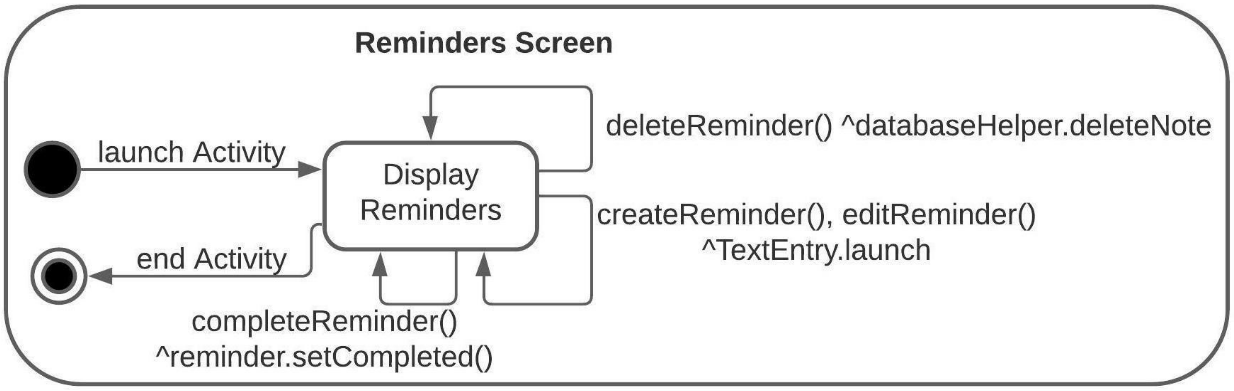
- ○ Complete Reminder



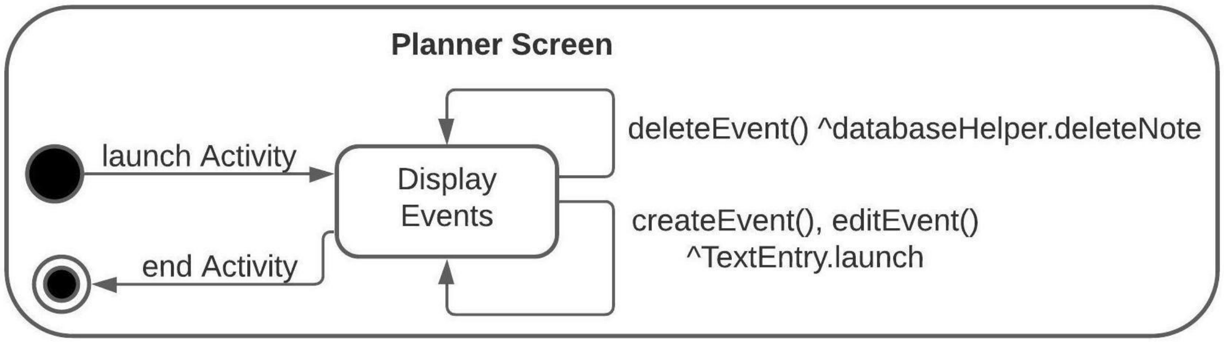
- **UML Statechart Diagrams**
  - Notes List Screen



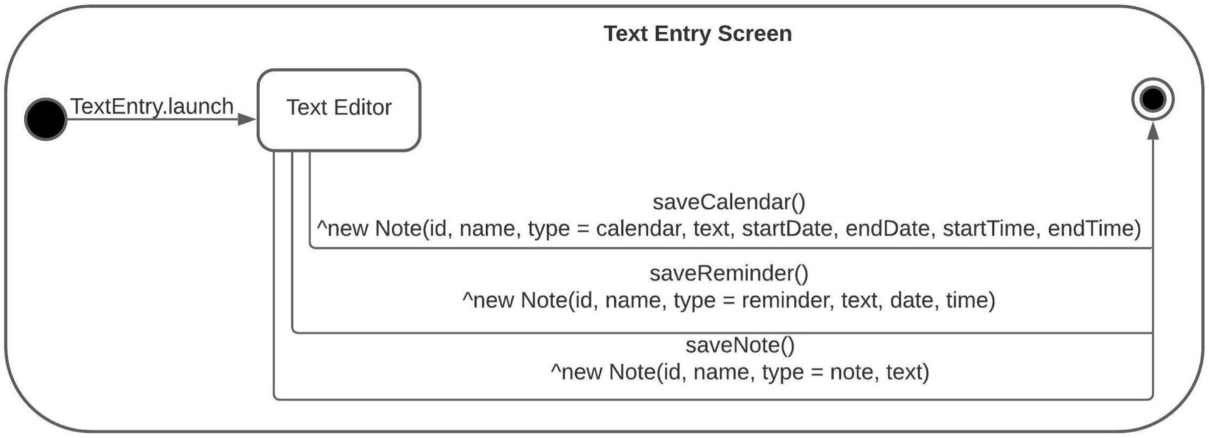
  - Reminders Screen



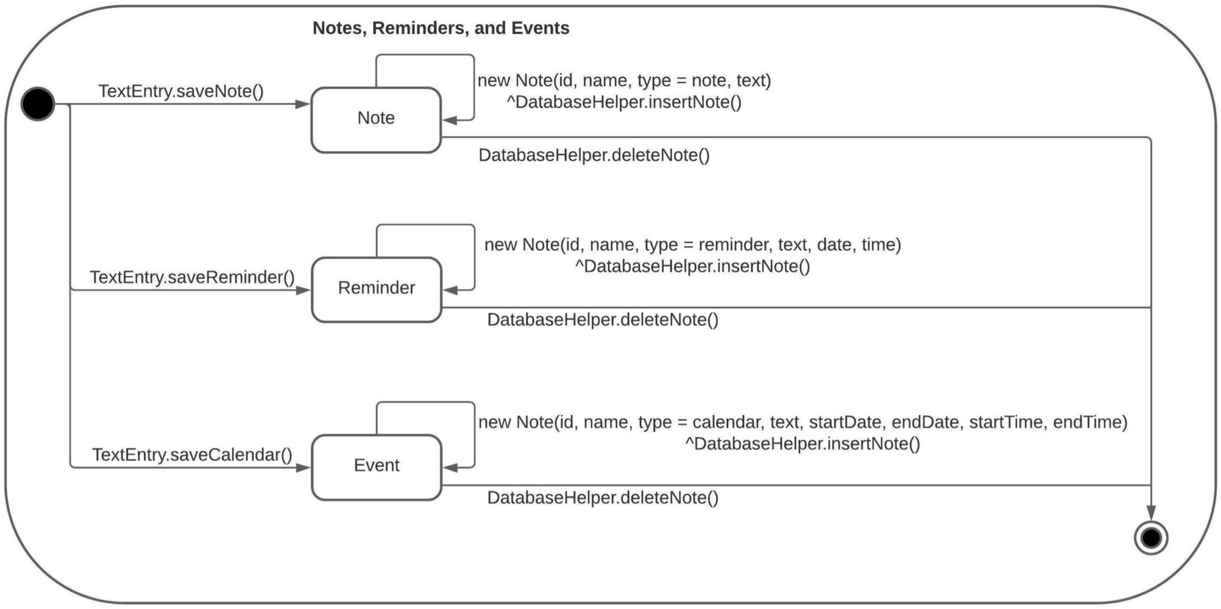
  - Planner Screen


Planner Screen
launch Activity
Display Events
end Activity
deleteEvent() ^databaseHelper.deleteNote
createEvent(), editEvent() ^TextEntry.launch

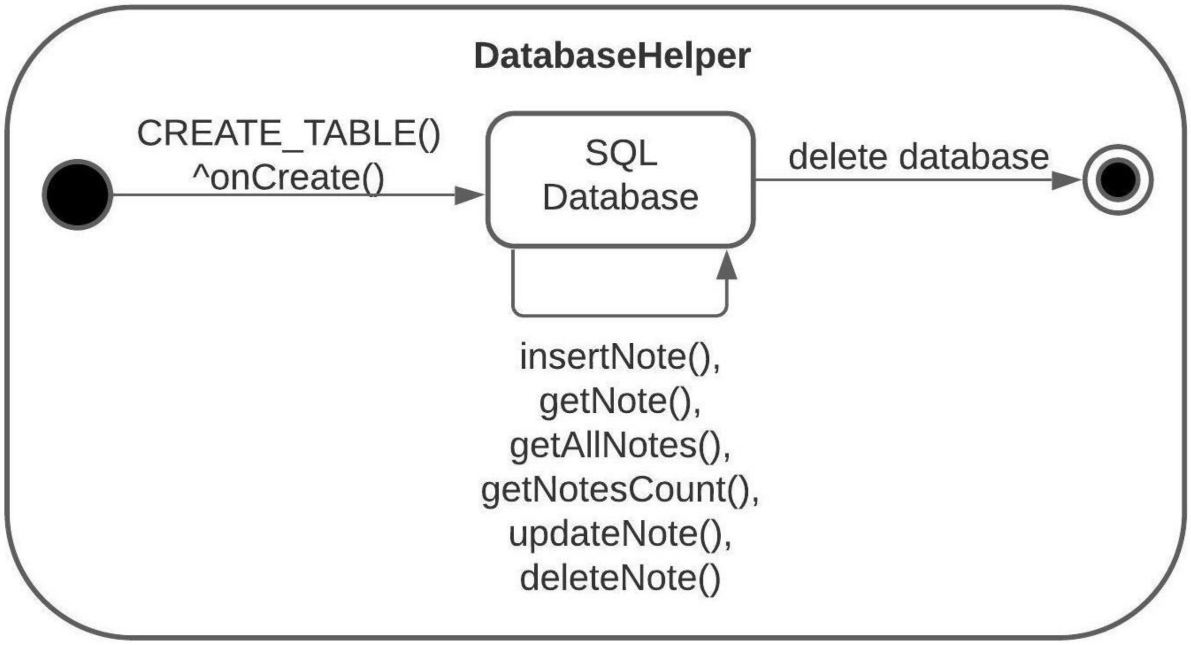
- Text Entry Screen


Text Entry Screen
TextEntry.launch
Text Editor
saveCalendar() ^new Note(id, name, type = calendar, text, startDate, endDate, startTime, endTime)
saveReminder() ^new Note(id, name, type = reminder, text, date, time)
saveNote() ^new Note(id, name, type = note, text)

- Note Class, Reminder, and Event Subclass


Notes, Reminders, and Events
TextEntry.saveNote()
Note
new Note(id, name, type = note, text) ^DatabaseHelper.insertNote()
DatabaseHelper.deleteNote()
TextEntry.saveReminder()
Reminder
new Note(id, name, type = reminder, text, date, time) ^DatabaseHelper.insertNote()
DatabaseHelper.deleteNote()
TextEntry.saveCalendar()
Event
new Note(id, name, type = calendar, text, startDate, endDate, startTime, endTime) ^DatabaseHelper.insertNote()
DatabaseHelper.deleteNote()

- Database Helper


DatabaseHelper
CREATE_TABLE()
^onCreate()
SQL Database
delete database
insertNote(),
getNote(),
getAllNotes(),
getNotesCount(),
updateNote(),
deleteNote()

Lessons Learned

Through this project, the team learned several lessons regarding planning and development.

First, we saw the importance of planning a project, especially its timeline, as demonstrated in the first deliverable. The activity graph and the Gantt chart were helpful guides to our development process to keep us on track throughout the semester. Additionally, specifying the team hierarchy, documentation plan, and technical requirements early on allowed us to avoid conflict later.

Moving on to the requirements specification, we learned the importance of precise, measurable requirements. The clear description of the functions, using use-case diagrams, made development go exponentially smoother. Early planning allowed the developers to focus solely on implementation rather than attempting to determine and implement features simultaneously.

The UML diagrams created in the third deliverable also brought about this ease of implementation. These diagrams gave a clear definition of the functions that the application would implement and their scope. Additionally, determining the system’s high- and low-level architecture was critical to defining how we would implement our application later.

Overall, this project demonstrates the importance of early planning, team management, and precise requirement specification to develop an application smoothly.