Team 7 Deliverable # 2 Report

GitHub Repo: https://github.com/SRV1302/CS3354-DailyDose-Team7.git

1. Title of the project: DailyDose

2. Delegation of tasks

Team member 1 - Zina Abou Assali (Frontend):

- Gather functional and non-functional requirements
- Work on sequence diagram for requirement _____ and the use case diagram
- Work on test code and cases
- Testing slide and presentation
- Use case slide and presentation
- Monthly View, motivational quotes to cycle and display in the monthly view

Team member 2 - Ugonna Anyalemechi (Backend):

- Gather functional and non-functional requirements
- Write out the non-functional requirements under "Organizational Requirements"
- Work on sequence diagram for requirement 1.3 and the use case diagram
- Adding, editing, deleting events (times included)
- Project timeline report and presentation slide
- Conclusion report

Team member 3 - Jeremiah Boban (Backend):

- Work on sequence diagram for requirement 1.2 and the use case diagram
- Send event to other calendar users, zoom in/out and scroll support
- Report and presentation slides

Team member 4 - Montserrat Milke Mosconi (Frontend):

- Create sequence diagram for viewing an event
- Create different agenda view options and send event alerts
- Comparison of our work with similar Calendar models, including proper research and citations

Team member 5 - Amrita Prabhakar (Frontend):

- Create Architectural Diagram
- Presentation Slides
- Assemble the recording

Team member 6 - Erin Turgut (Backend / Full Stack):

- Gather functional and non-functional requirements
- Write out the functional requirements
- Write out the non-functional requirements under "External Requirements"
- Worked on sequence diagram for requirements 1.5 and 1.6
- Worked on code to test for test plan
- FP calculations and assumptions breakdown
- Sequence diagram slide and presentation
- Functional and non-functional requirements slide presentation

Team member 7 - Shriya Vetapalem (Frontend):

- Gather functional and non-functional requirements
- Write out the non-functional requirements under "Product requirements" and "External Requirements"

- Work on the class
- Weekly View
- Presentation Slide- Class Diagram
- Estimated cost of hardware products
- Estimated cost of software products
- Estimated cost of personnel

3. Content from Deliverable #1:

----

3. We are making an assumption that it is government law that, if there is a user data security breach for a company, the company must notify all users of this breach within 72 hours.

4. The feedback we received from the project proposal was to include tasks for each team member that cover more parts of the SDLC than just implementation. We have addressed this by adding more tasks for each member in question #2 that include diagrams and engineering requirements.

5. **Chosen Software Process Model:** Waterfall

Why: Clear understanding of what we want from the app, fixed deadlines, simple application

6. **Software Requirements:**

a. Functional Requirements:

1.1 User shall be able to view all events by selecting either daily view, weekly view, or monthly view.

1.2 User shall be able to add an event with event details

1.3 User shall be able to delete an event

1.4 User shall be able to edit an event

1.5 User shall be able to sign up

1.6 User shall be able to login

b. Non-functional Requirements:

<u>**Product Requirements**</u>

- **Usability Requirement** - The DailyDose system shall have a clear and easy-to- use interface that lets the users add, delete or edit events in three clicks or less from the main calendar screen. The DailyDose shall also have a consistent interface across all pages and views.

- **Performance Requirements** - The DailyDose system shall update or save the events after the user clicks the "Save" button.

- **Space Requirements** - The DailyDose shall automatically delete temporary or any unsaved event drafts and cached files after 48 hours.

- **Dependability Requirements** – The DailyDose system shall be available 99% of the time, and users should still be able to view their existing calendar during scheduled maintenance.

- **Security Requirements** – The DailyDose system shall encrypt all the user data stored on the server to protect the user's privacy.

<u>**Organizational Requirements**</u>

- **Development Requirement -** The DailyDose system shall follow an established set of coding standards to ensure consistency among developers.

- **Environmental Requirement –** The DailyDose system shall be developed using the latest stable and supported technology stack.

- **Operational Requirement –** The DailyDose system shall receive scheduled bug fixes to ensure consistent reliability and compliance with operational standards.

External Requirements

- **Regulatory Requirement**- The DailyDose shall store and handle any user data according to the privacy rules, and these rules shall be clearly shown to the user when they sign up.

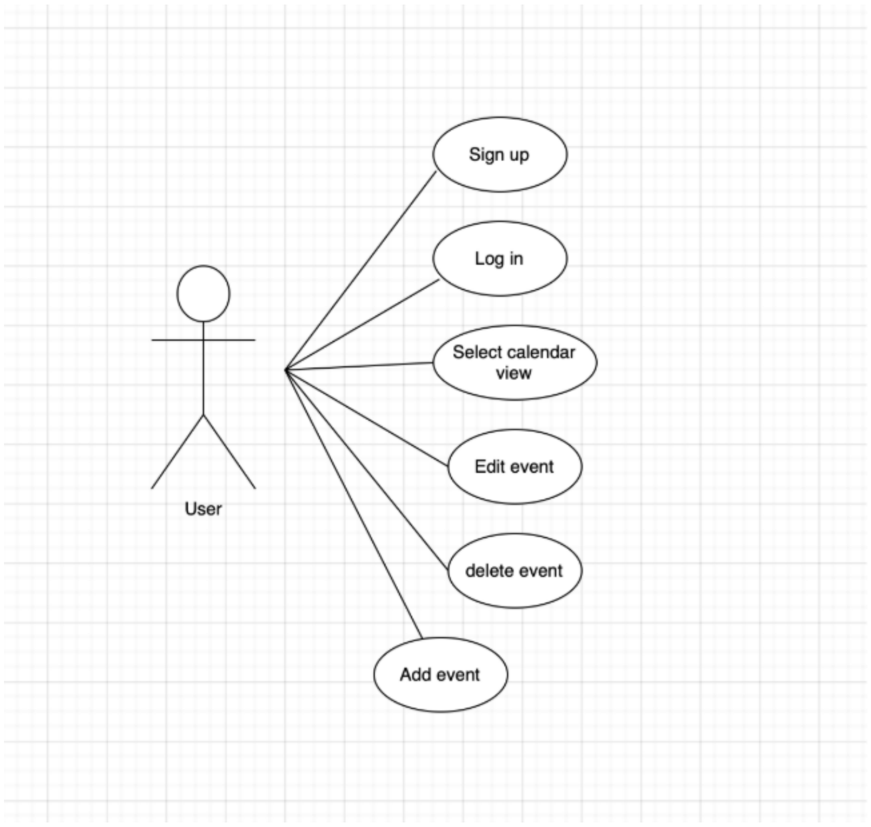
- **Accounting Requirement** – The DailyDose system does not include any payments or financial transactions.

- **Ethical Requirement** – The DailyDose system shall allow users with an option to delete their data permanently if they wish to delete it.

- **Safety/Security Requirement** - The DailyDose system shall ensure user data security by never storing user passwords directly.

- **Legislative Requirement** - The DailyDose system shall notify all users in the case of a data security breach within 72 hours as set out in government law.

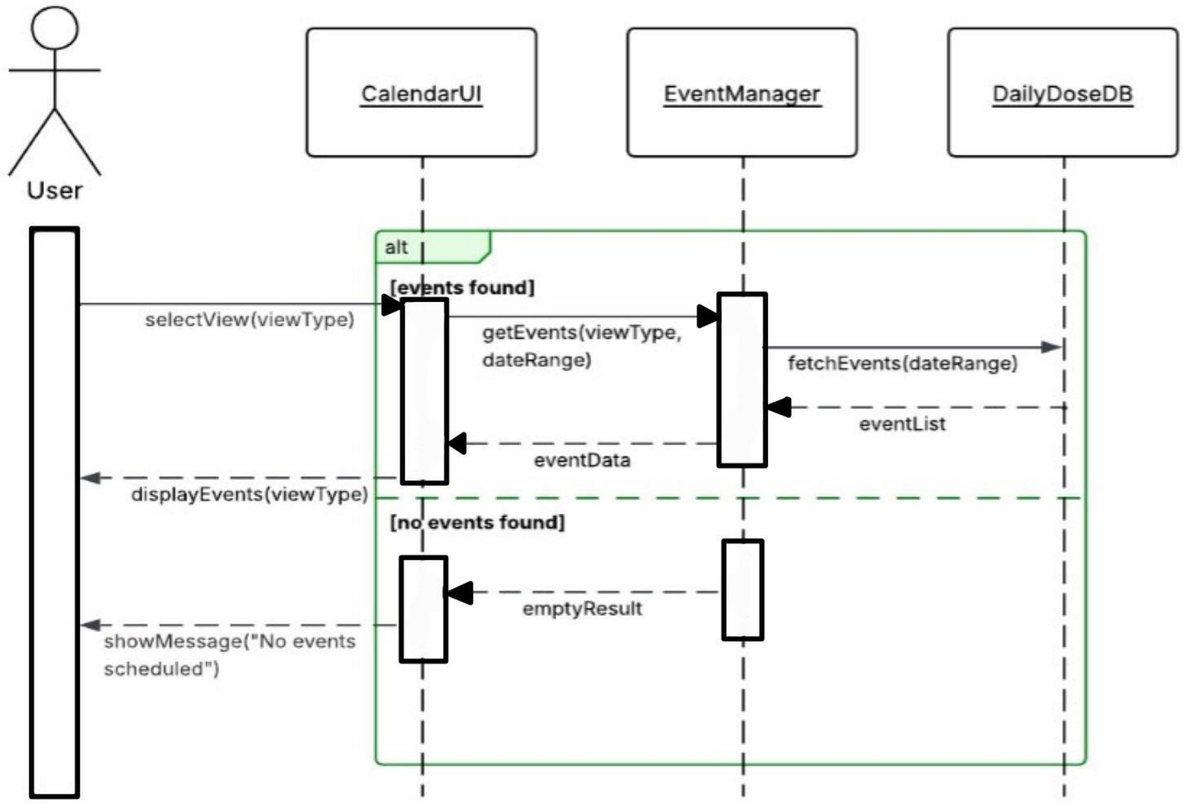
7. Use case diagram:



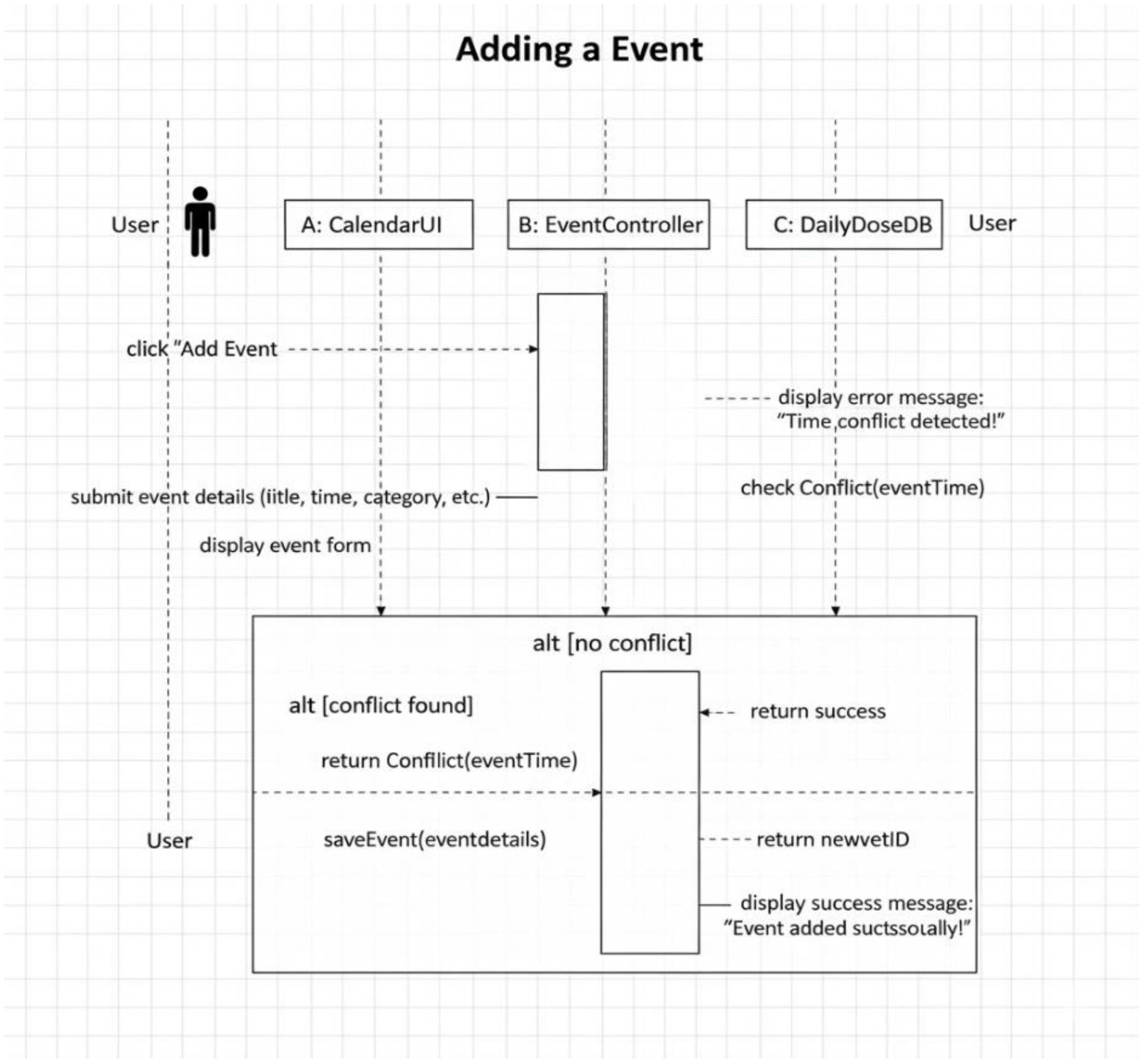
8. Sequence diagrams:

1.1:

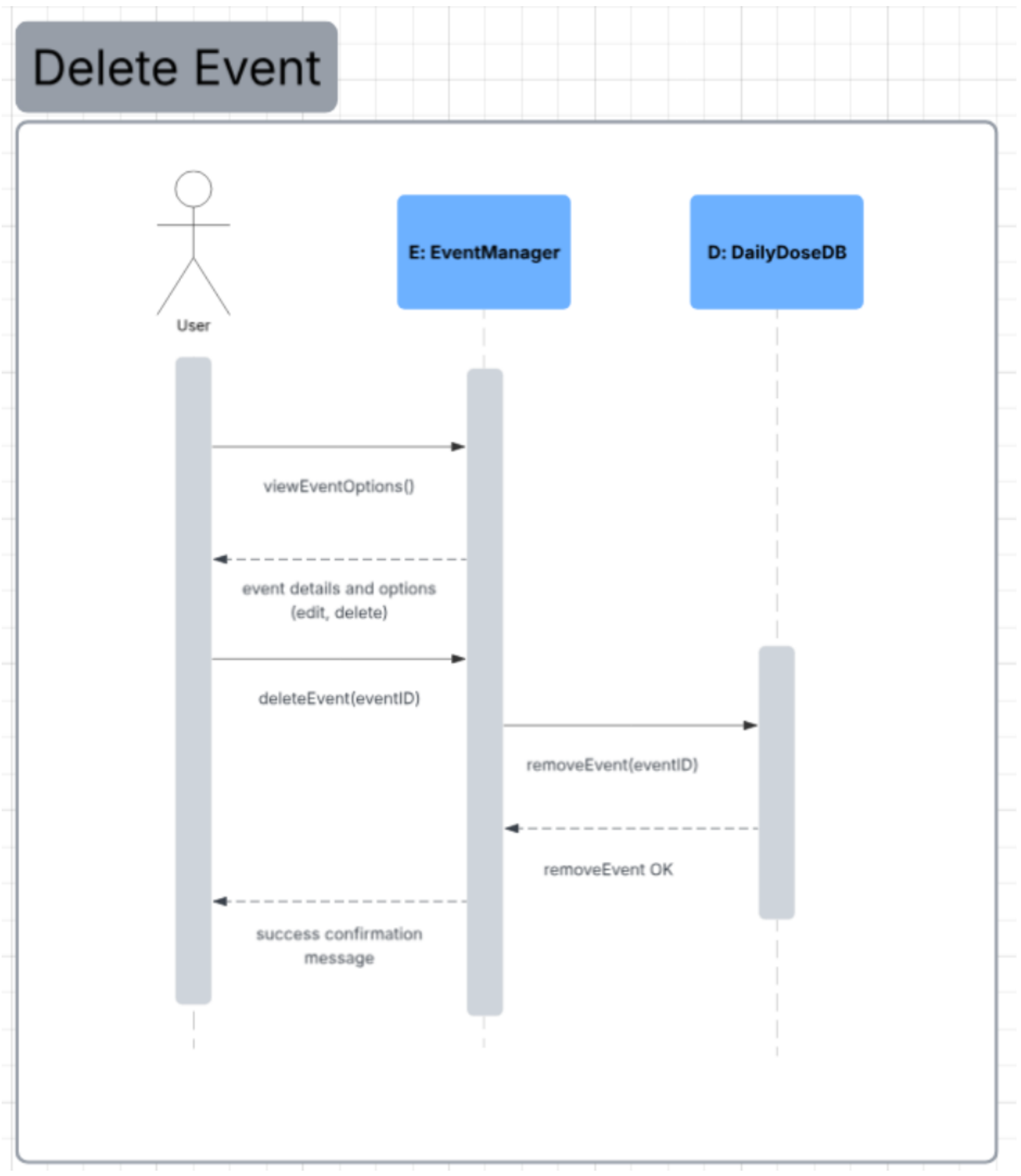
View Event (1.1)



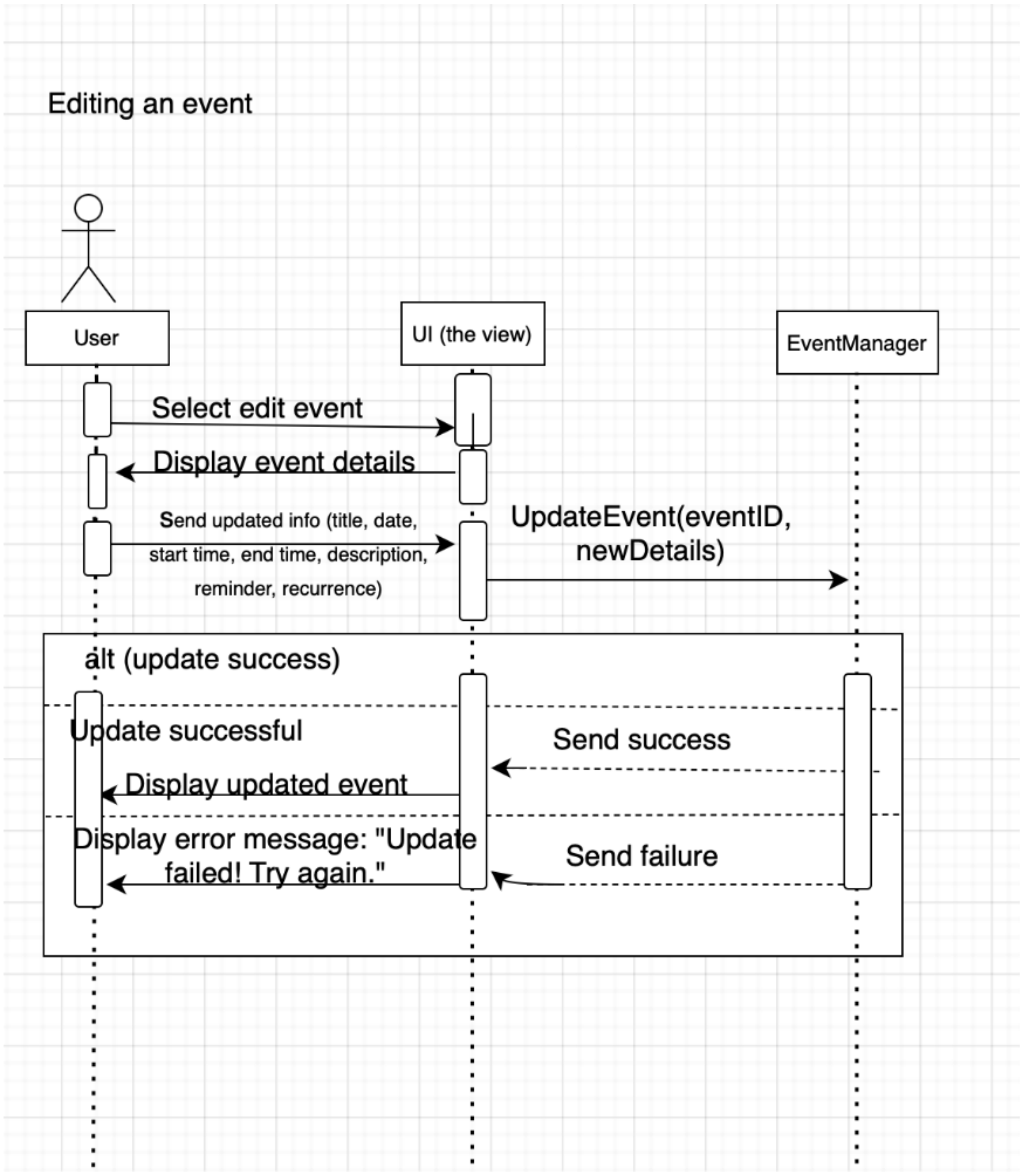
1.2:


Adding a Event
User
A: CalendarUI
B: EventController
C: DailyDoseDB
User
click "Add Event
display error message:
"Time conflict detected!"
submit event details (iitle, time, category, etc.)
check Conflict(eventTime)
display event form
alt [no conflict]
alt [conflict found]
return success
return Confilict(eventTime)
User
saveEvent(eventdetails)
return newvetID
display success message:
"Event added suctssoually!"

1.3:


Delete Event
User
E: EventManager
D: DailyDoseDB
viewEventOptions()
event details and options (edit, delete)
deleteEvent(eventID)
removeEvent(eventID)
removeEvent OK
success confirmation message

1.4:


Editing an event
User
UI (the view)
EventManager
Select edit event
Display event details
Send updated info (title, date, start time, end time, description, reminder, recurrence)
UpdateEvent(eventID, newDetails)
alt (update success)
Update successful
Send success
Display updated event
Display error message: "Update failed! Try again."
Send failure

1.5:

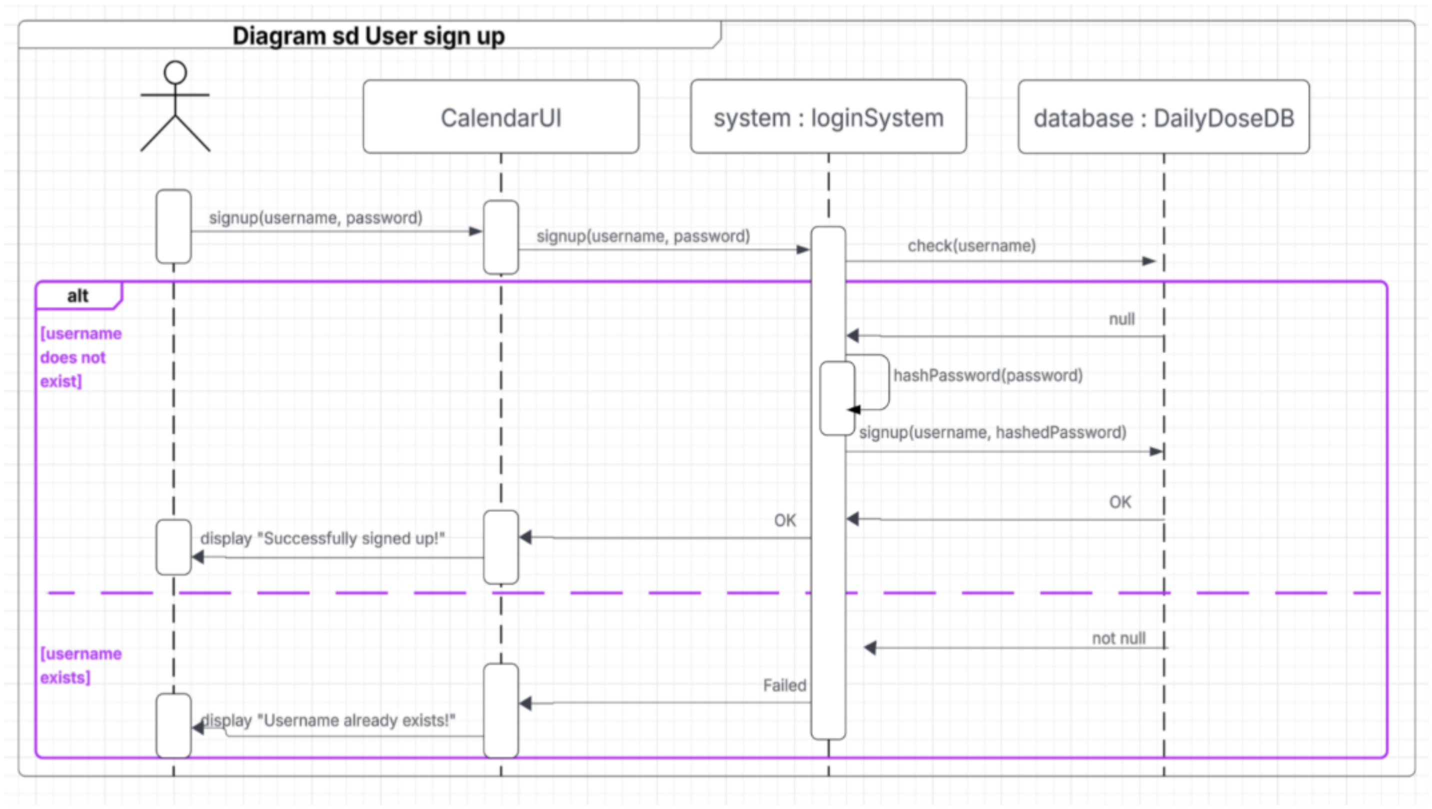
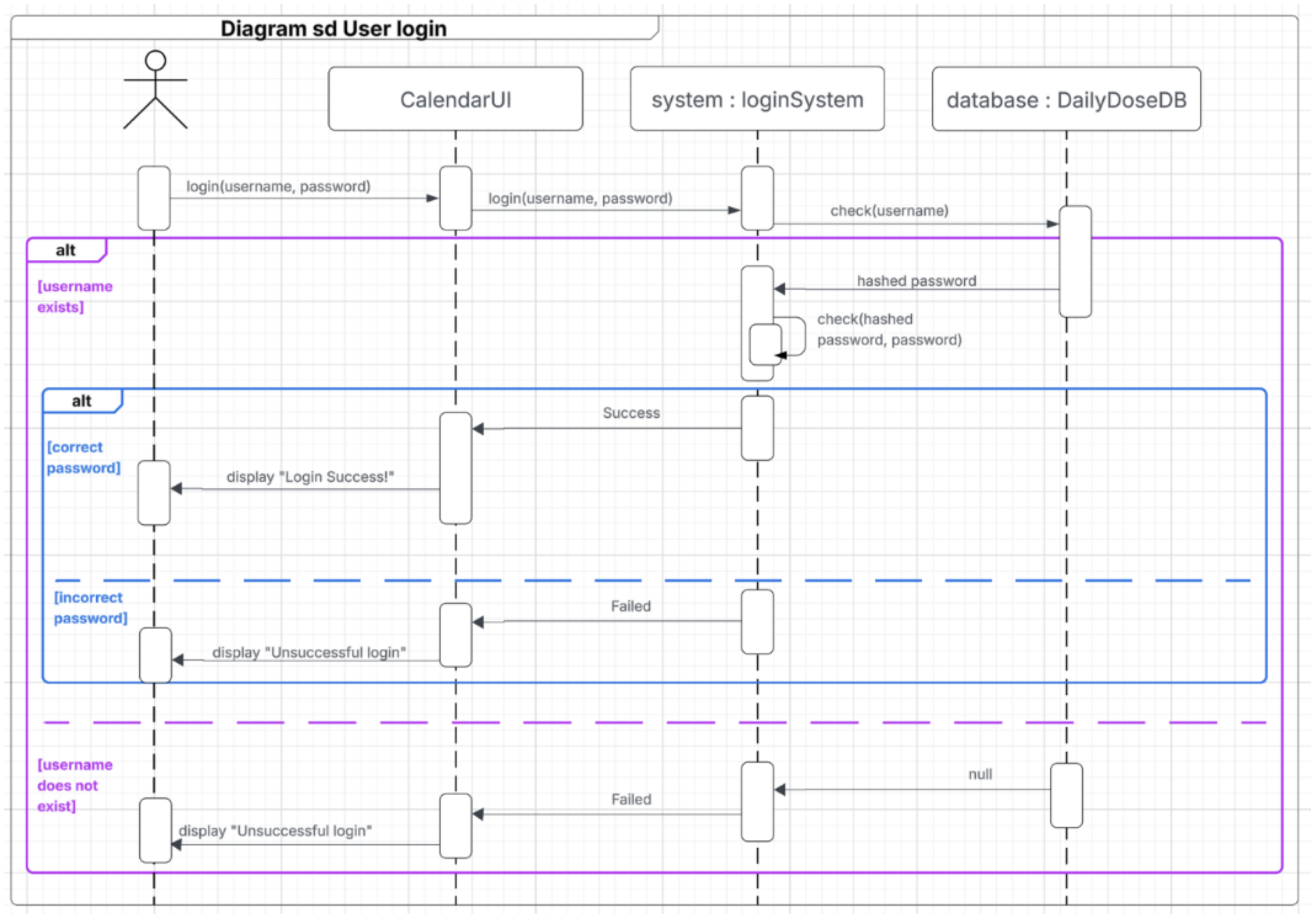
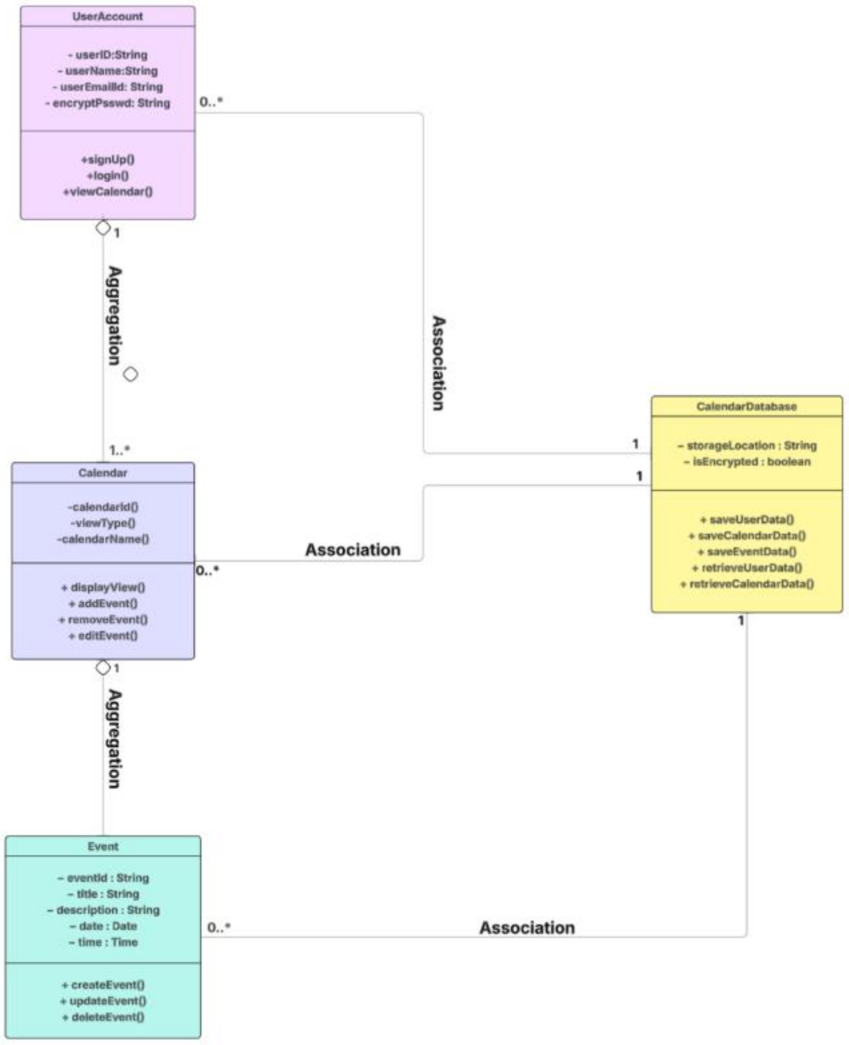


Diagram sd User sign up
CalendarUI
system : loginSystem
database : DailyDoseDB
signup(username, password)
signup(username, password)
check(username)
alt
[username does not exist]
null
hashPassword(password)
signup(username, hashedPassword)
OK
OK
display "Successfully signed up!"
[username exists]
not null
Failed
display "Username already exists!"

1.6:

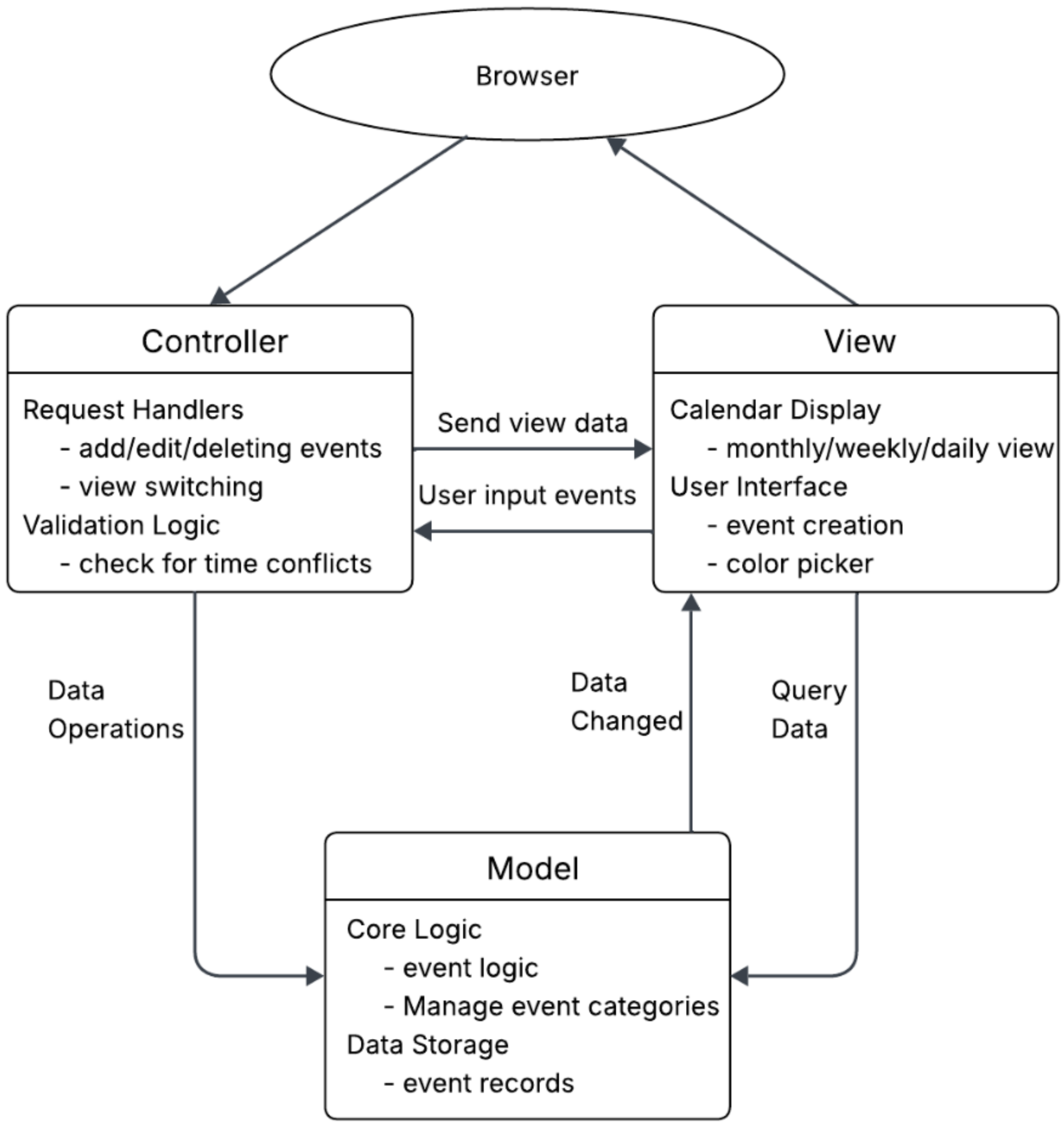


9. Class diagram:



10. Architectural design:

4.



End of content from deliverable #1

4.

a. Project Scheduling:

Start Date: January 1, 2026

End Date: March 6, 2026

Activity	Estimated Time of Competition (Days)
System Requirements Gather system requirements by interviewing students and other potential stakeholders	14
System Design Identify and design system architecture, database, and interfaces; determine necessary component interfaces to reuse	10
Software Feature Addition Implement daily, weekly, and montly views to display calendar with necessary test cases	
Software Feature Addition Implement event addition feature along with necessary test cases	7
Software Feature Addition Implement event deletion feature along with necessary test cases	3
Software Feature Addition Implement event editing feature along with necessary test cases	3
Software Feature Addition Implement application authentication with necessary test cases	3

System Validation/Testing Perform system acceptance testing, debugging and making additional modifications as needed	14
v1.0 release	

Justification: This 8.7-week timeline follows a "Lean Startup" methodology, sufficient for developing a Minimum Viable Product (MVP) with core calendar functionality (CRUD events, views).

Schedule Details:

1. **Weekends:** Weekends **are counted** in the schedule to allow for flexible development sprints typical in small software projects.
2. **Working Hours:** The project assumes **4 hours per day** per developer. This reflects a focused, part-time workload often seen in freelance application development

b. Cost, Effort and Pricing Estimation:

(i) Methodology: We utilized **Function Point Analysis (FPA)** to estimate the size and cost of the project. This method was chosen to quantify the functionality provided to the user based on the requirements.

(ii) Assumptions & Parameters:

- I am assuming this to be a simple calendar app with only the key CRUD operations for a user's personal event management. Therefore, I am assuming the complexity of weighing for all the above function categories to be **simple**.
- I am assuming that no external applications or APIs, such as ones for authentication or notifications, will be needed. This will be coded with the project. Therefore, **Number of external interfaces = 0**.

  1.Number of user input = 4
    - User can add events
    - User can delete events
    - User can edit events
    - User can sign up/login

- 2.Number of user output = 1
    - Reminders/notifications to user about upcoming event
- 3.Number of user queries = 2
    - View daily, weekly, or monthly calendar
    - View event details
- 4.Number of data files and relational tables = 2
    - Data table for all users
    - Data table for events
- 5.Number of external interfaces = 0 (reason stated above)
    - For determining the processing complexity, here is the breakdown I have assumed:
        - 0 (no influence) : 3,10,12,13
        - 1 (incidental) : 9
        - 2 (moderate) : 2,4, 5,8
        - 3 (average) :
        - 4 (significant) : 1,6,7,11,14
        - 5 (essential) :

(iii) Estimations & Calculation:

1. Unadjusted Function Point (UFP) Count: Based on the requirements, we identified the following counts and applied "Simple" complexity weights:

Function Category	**Count**	**Complexity (Simple)**	**Complexity (Average)**	**Complexity (Complex)**	**Count x Complexity**
External Input (EI) *(Add, Delete, Edit, Sign Up/Login)*	4	x 3	4	6	12
External Output (EO) *(Notifications)*	1	x 4	5	7	4

External Inquiry (EQ) *(View Calendar, View Details)*	2	x 3	4	6	6
Internal Logic File (ILF) *(User Table, Event Table)*	2	x 7	10	15	14
External Interface File (EIF)	0	x 5	7	10	0

Gross Function Point = 12 + 4 + 6 + 14 + 0 = 36 FP
PCA = 0.65 + 0.01(4 x 0 + 1 + 4 x 2 + 5 x 4) = 0.94

Function Points = 36 x 0.94 = 33.84 FP

2. Final Function Point & Effort:

- **Net Function Points:** 36 (Gross) * 0.94 (PCA) = **33.84 FP**
- Estimated Effort: Using a productivity rate of 5 hours per FP (standard for simple web projects), the total effort is:

33.84 FP * 5 hours ≈ 169.2 Person-Hours

(iv) Pricing:

- **Labor Cost:** 169.2 hours * $30/hr (Junior Rate) = $5,076
- **Overhead & Profit (15%):** $761.40
- **Final Project Price: $5,837.40**

c. Estimated cost of hardware products:

For this DailyDose app, no physical hardware is required because the application would run entirely on a cloud platform. Based on our study of current AWS pricing for lightweight applications a small server instance and basic storage are sufficient for this type of calendar application.

Assumptions:

- One small virtual machine is enough for hosting the backend.
- Only a small amount of SSD storage is required for user and event data.
- Standard snapshot storage is used for backup.
- The app is expected to handle light traffic as an MVP.

The t4g.small price covers the server, storage, and IP added together, and the snapshot cost is added separately for backups.

Item	Description	Cost per Month
AWS t4g.small instance	Includes compute, storage, and IP	$10.72
Snapshot storage	$0.05 per GB per month 10 GB is required	$ 0.50

[1] [2]

d. Estimated cost of software products:

Most of the software needed to build DailyDose is free because we are using open source tools and frameworks. We are also not using any paid APIs for external services within our own code. The only software-related cost comes from the small cloud database needed to store user accounts and events.

Assumptions:

- Using free and open source development tools such as React, HTML, CSS, Java, GitHub Free Tier
- No paid third-party API required

- A small managed SQL database service is needed to store users and events reliably
- All other tools required during development have no licensing fee

Cost:

Item	Description	Cost per Month
Cloud Database	Small managed SQL database to store event and user data	$10

[3]

Based on the hourly rates listed on the AWS RDS pricing page, we calculated that the smallest database instance would cost roughly $10/month, which is appropriate for our app's size and usage.

<u>e. Estimated cost of personnel:</u>

Since DailyDose is a small application only a minimal part-time is needed to design, build,test and suuport the system. We estimated personnel costs using the current hourly rates for entry level software roles in Texas. These rates come from ZipRecruiter job data and represent typical pay for developers, testers, project managers and IT support staff in this region.

Although the salaries are listed for full time positions the hourly rates remain the same for part timers, so they can be used directly to estimate cost of personnel for this project.

Assumptions:

- A small part time team is enough for full development of this app.
- Only essential software engineering roles are included
- Hourly rate are based on the entry level salary data in Texas.
- Training after installation is minimal and can be covered by IT support.
- No long term staff is needed due to small scope of the application

Cost:

Role	Hourly rate	Source
Entry level Software Developer	~ $45 /hour	ZipRecruiter – Entry level Software developer in Texas
QA Tester	~ $43 / hour	ZipRecruiter – Entry level Software Tester in Texas
Entry level Project Manager	~ $46/ hour	ZipRecruiter – Entry level Project manager in Texas
IT support Specialist	~ $31 /hour	Zip Recruiter – IT Support Specialist

[4][5][6][7]

Personnel Required:

- 1 Project Manager
- 3 Developers
  - 1 Frontend Developer
  - 1 Backend Developer
  - 1 Full Stack Developer
- 1 QA Tester
- 1 Training / Support Person

These rates provide a basic estimate of the staffing cost and the total cost for personnel would depend on the number of part-time hours required.

5. Test Plan

a. One of the most important aspects of using a calendar is to make sure that there are no conflicts between something you are trying to schedule and something you have already scheduled. It would be catastrophic to use a calendar app and having it schedule two events for the same day and time, only to arrive at the day and realized that you have been overbooked by the app because it did not flag that there was already an event booked for the same time and day. To ensure that our calendar app does not succumb to this problem, we will be creating a function of the code that checks for an already created event on the same time and day as the event the user is already trying to create. We will then test that method, ensuring that it returns false when there is already another event booked, or true if there is not, allowing the user to proceed with their creation of a new event.

b. For the implemented code, we created a snippet of a method that would check for any conflicting events when the user wants to add an event. What this method does is take in all of the information the user inputs about their event they are wanting to schedule (such as the year, month, day, hour, and minute) and turn it into an object from one of our classes that we can use and send around. The method then checks for conflicts by looping through each existing event that we already have stored in our array and ensuring that no such event coincides with the one we are wanting to schedule. If there is an existing event at the time we are wanting to schedule it for, then we will return false. If there is not, then the method returns true, after we have already added the specific event to our dynamic ArrayList. Below is the implemented code (this was also published to our GitHub repository):

```
//This class will include user schedule data and CRUD operations.

//For now, the method scheduleEvent checks for scheduling conflicts when a new event is
added by the user. This is the method that is tested in our test plan.


import java.time.LocalDateTime;


public class UserSchedule {


  private static LocalDateTime[] userEvents = {LocalDateTime.of(2025, 12, 8, 15, 0),

      LocalDateTime.of(2025, 11, 28, 12, 30),

      LocalDateTime.of(2025, 11, 27, 9, 45),

      LocalDateTime.of(2025, 11, 26, 16, 0) };


  public static boolean scheduleEvent(int year, int month, int dayOfMonth, int hour, int
minute) {

    LocalDateTime newEvent = LocalDateTime.of(year, month, dayOfMonth, hour,
minute);
```

```
        //check if the arraylist contains an event at the same time that we are wanting to add a
new event
        if(UserEvents.contains(newEvent))
        {
            return false;
        }

        //if not, then add the event and return true
        userEvents.add(newEvent);
        return true;
    }

    //the arraylist that holds the events
    return new ArrayList<>(userEvents);
}
```

c. To test this method, we decided to come up with three different tests that we implemented using JUnit5 testing. Firstly, we needed to test whether there was an already existing event in the ArrayList at the same time that we are trying to create our new event. We have one test that asserts false for no existing event, and one test that assrts true for having an existing event in place. We use these tests by giving the ArrayList example events and testing to see if there's an already existing event or not. For our third test, we needed to make sure that once we added the event, it would actually be in the ArrayList as a new event. In order to test for this, we created a new event and added it to the list using the method we are testing, and then check the ArrayList to see if the method was created in the list. We would then assert true if it was. Below is the following code for the testing that was also published to the github:

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import java.time.LocalDateTime;
```

```
//these tests will ensure the schedule event function correctly checks for an existing event
public class UserScheduleTest {

  //this test returns false since there is no existing events
  @Test
  void testNoExistingEvent() {
    boolean result = UserSchedule.scheduleEvent(2025, 11, 28, 12, 30);
    assertFalse(result);
  }

  //this test run returns true that there is an existing event
  @Test
  void testWithExistingEvent() {
    boolean result = UserSchedule.scheduleEvent(2025, 12, 1, 10, 0);
    assertTrue(result);
  }

  //this test checks if the event was actually added to the arrayList after we add it in our
function
  @Test
  void testEventAdditionValidity() {
    //create a new event
    LocalDateTime newEvent = LocalDateTime.of(2025, 12, 1, 10, 0);

    //add to the list and check if it
    UserSchedule.scheduleEvent(2025, 12, 1, 10, 0);

    //check the list to see if the event was added and return true if it was
    assertTrue(UserSchedule.getUserEvents().contains(newEvent));
  }
}
```

6. Comparison of work with similar designs:

For this analysis, we compared different aspects of our software to other existing calendar systems such as Google and Outlook Calendars. We also analyzed the primary audiences and users who would specifically benefit from each of these systems.

DailyDose was designed to be a very lightweight and simple personal scheduling tool. It utilizes browser-based MVC architecture that separates logic, UI rendering and user interactions, demonstrating its status as an architecture in the small approach as defined by Kruchten due to its simplicity and self-contained application scope [8]. The major advantage of this software heavily revolves around its simplicity due to enabling fast user interactions with minimal overhead, making the system more user-friendly and personal rather than distributed or enterprise-based.

Contrarily, Google Calendar is built using a much more sophisticated approach, which operates on a globally distributed back-end system built upon Google's spanner infrastructure [9]. Its architecture in the large approach provides numerous services that are distributed and synchronous, like having consistent storage across multiple data centers worldwide [8]. Its large-scale deployment of its Calendar services relies on cluster management like Borg and Omega, which handle large computational workloads across servers in complex configurations [10], allowing for real-time synchronization, cross-device access and multi-user collaboration. These features are more suitable for users who have considerable usage of Google services and may benefit from its advanced features.

Outlook's Calendar software is supported by the Microsoft Exchange Server, which uses centrally-managed repository to store mail, calendar events and directory information [11]. Access to data like events and user scheduling details is exposed though interfaces like the Microsoft Graph API [12], which provides developers with the proper resources to organize and synchronize user data across different Microsoft services. Outlook's Calendar uses a layered model in which user authentication, application logic and data storage are each separated into distinct components with central management, therefore allowing for meeting invites, resource booking and other organization-wide collaboration. With this in mind, it is evident that this is a popular choice among corporations or larger enterprises due to its collaboration capabilities.

DailyDose's largest restraint is its limited architectural model, which was simply designed for single-user local scheduling instead of collaborative or enterprise-level functionality. However, this is exactly why it's the most beginner-friendly out of the three explored programs; it is much likelier to experience problems with complexity within Google or Outlook-based calendars due to the extent of their features. Overall, DailyDose prioritizes clarity and simplicity for any personal use while Google and Outlook focus more on synchronization of complex data and collaboration within enterprises.

7. Conclusion:

Based on our thorough research and examination of software engineering principles and methodologies, we have evaluated that our development approach is feasible given the scope and constraints of the application. Although no significant changes were made to what was originally planned, we expect several changes during the actual development of the application as a result of the ever-evolving nature of software in today's tech-centric world. Even with the simple implementation of a calendar application, numerous decisions and processes that must be made to deliver a reliable product to consumers, underscoring the importance of software engineering guidelines when developing real software solutions for individuals, organizations, and communities.

8. References:

[1] "EBS pricing," Amazon Web Services, Inc. https://aws.amazon.com/ebs/pricing/

[2] Kuberns, "Kuberns: One-Click AI Cloud Deployment," Kuberns. https://kuberns.com/pricing

[3] "Amazon RDS for MySQL pricing," Amazon Web Services, Inc. https://aws.amazon.com/rds/mysql/pricing/?pg=pr&loc=2

[4]"Entry Level Software Developer," *ZipRecruiter*, 2024. https://www.ziprecruiter.com/Salaries/Entry-Level-Software-Developer-Salary--in-Texas

[5]"Qa Tester," *ZipRecruiter*, 2025.

https://www.ziprecruiter.com/Salaries/Qa-Tester-Salary

[6]“Entry Level Project Manager,” *ZipRecruiter*, 2025. https://www.ziprecruiter.com/Salaries/Entry-Level-Project-Manager-Salary--in-Texas (accessed Nov. 23, 2025).

[7]”IT Support-Specialist-Safety-Salary-in-Plano,” *ZipRecruiter*, 2025.

https://www.ziprecruiter.com/Salaries/It-Support-Specialist-Salary-in-Plano,TX

[8] G. Kruchten, “Architectural Blueprints—The ‘4+1’ View Model of Software Architecture,” IEEE Software, vol. 12, no. 6, 1995. https://ieeexplore.ieee.org/document/469759

[9] J. C. Corbett et al., “Spanner: Google’s Globally-Distributed Database,” Google Research, 2012. https://research.google/pubs/pub39966/

[10] B. Burns et al., “Borg, Omega, and Kubernetes,” ACM Queue, vol. 14, no. 1, 2016. https://queue.acm.org/detail.cfm?id=2898444

[11] Microsoft, “Exchange Server Protocol Documentation.” https://learn.microsoft.com/en-us/openspecs/exchange_server_protocols/

[12] Microsoft, “Microsoft Graph API Overview.” https://learn.microsoft.com/en-us/graph/overview