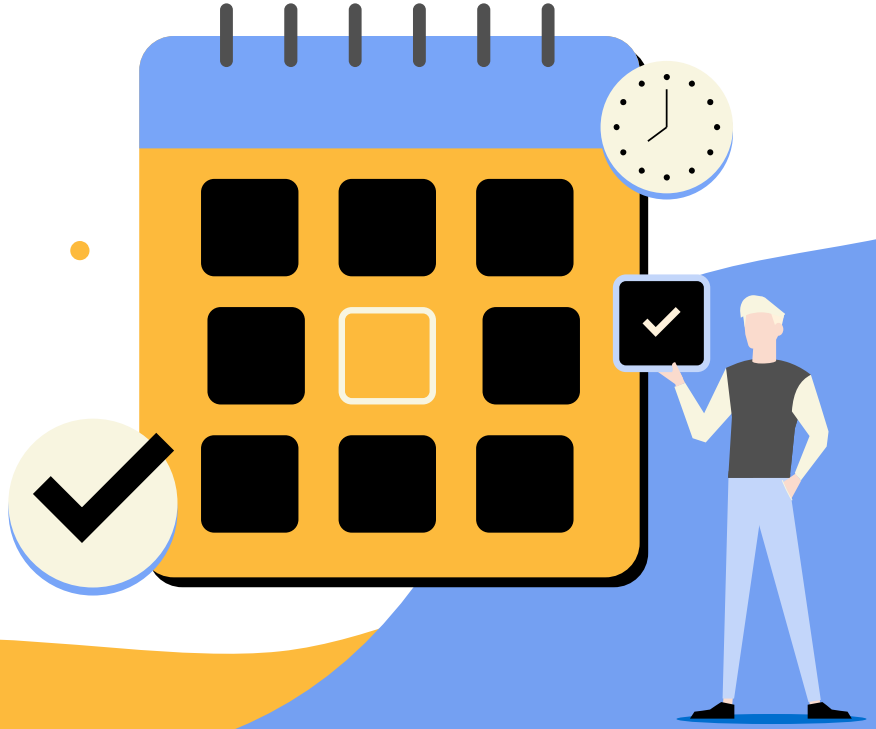
DailyDose

Zina Abou Assali, Ugonna Anyalemechi, Jeremiah Boban, Montserrat Milke Mosconi, Amrita Parbhakar, Erin Turgut, Shriya Vetapalem



Project Objective

- **Objective:**
  - Create a motivating calendar app that helps users manage daily, weekly, and monthly events while staying organized
- **Why:**
  - As college students, we rely on calendars to keep up with classes, exams, meetings, and personal goals. DailyDose adds motivational quotes and goals to help students stay on track
  - Useful for students and anyone who needs an easy way to manage their schedule

Cost Estimation

Function Category	Count	Complexity (Simple)	Complexity (Average)	Complexity (Complex)	Count x Complexity
1	4	3	4	6	12
2	1	4	5	7	4
3	2	3	4	6	6
4	2	7	10	15	14
5	0	5	7	10	0

- 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) :

Function Point Analysis (FPA) Calculation:

- 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

- Productivity Rate: 5 hours per FP
- Total Effort: 169.2 person-hours
- Labor Rate: $30/hr
- Estimated Cost: $5,076
- +15% Profit Margin: $761.40
- Total Price: $5,837.40

Project Timeline

Activity	Estimated Time of Completion (Days)
Gather system reqs. by interviewing students and other stakeholders	14
Identify and design system architecture, database, interfaces	10
Implement daily, weekly, and monthly views to display calendar with test cases	7
Implement event addition feature with test cases	3
Implement event deletion feature with test cases	3
Implement event editing feature with test cases	3
Implement app authentication with test cases	7
Perform system and acceptance testing, debugging and making additional changes as needed	14
v1.0 release	

Functional Requirements

- 1.1 View events (Daily/Weekly/Monthly)
- 1.2 Add an event
- 1.3 Delete an event
- 1.4 Edit an event
- 1.5 Sign up
- 1.6 Login

Non-Functional Requirements

Product:

- Usability: Interface supports actions in 3 clicks or less
- Performance: Save events immediately after clicking “Save”
- Space: Auto-delete cached drafts after 48 hours
- Dependability: 99% uptime
- Security: Encrypt all user data

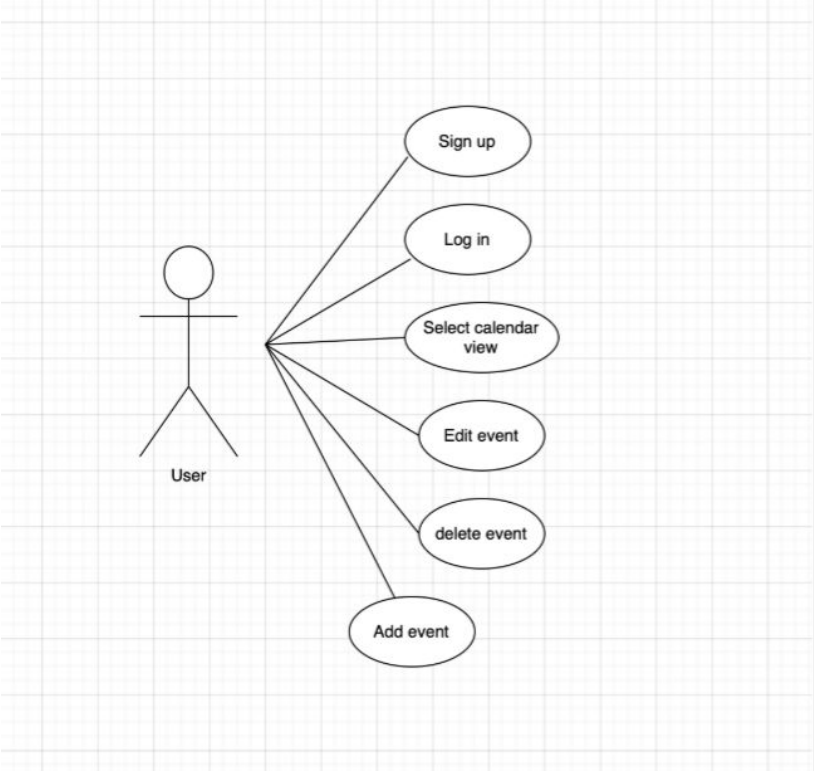
Organizational:

- Coding standards
- Latest stable tech stack
- Scheduled bug fixes

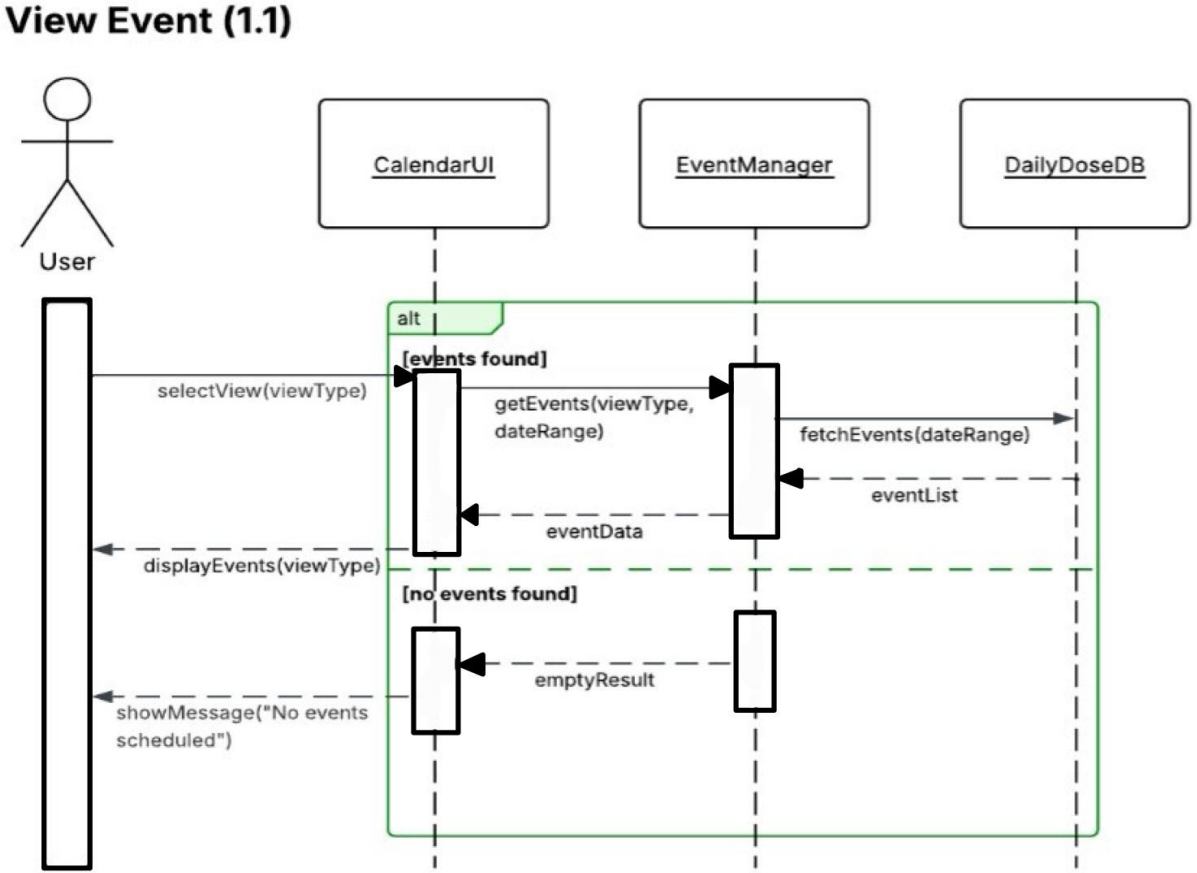
External:

- Regulatory privacy rules
- Users can permanently delete data
- Never store passwords directly
- 72-hour breach notification law

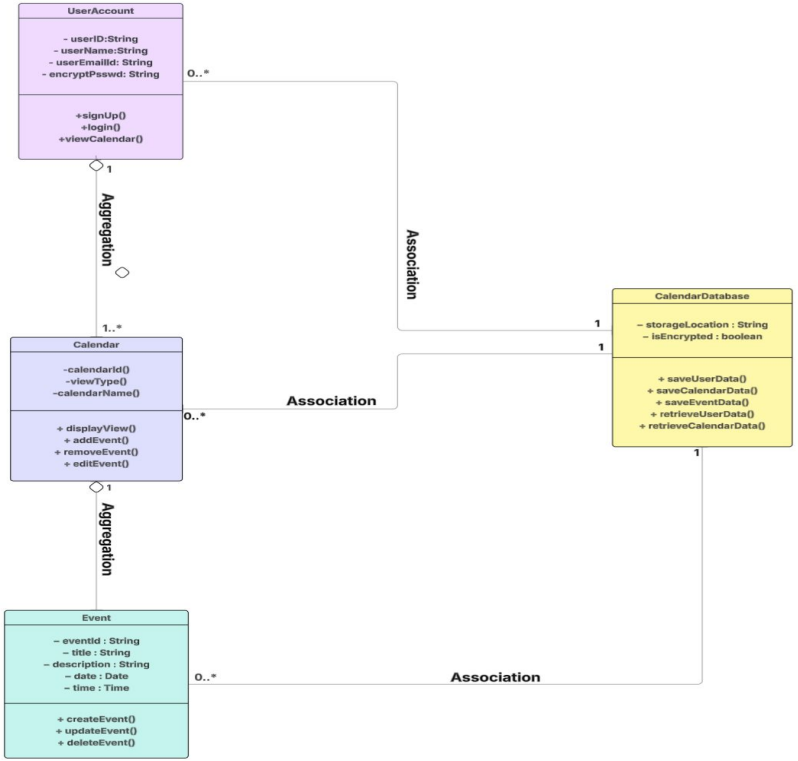
Use Case Diagram



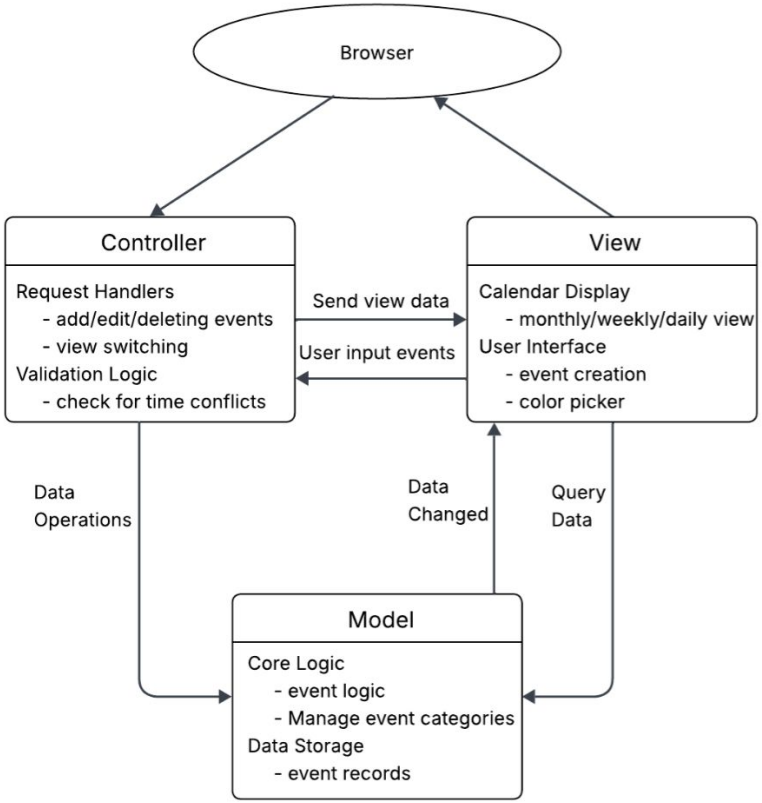
Sequence Diagram



Class Diagram



Architectural Design



DailyDose vs. Existing Calendar Systems

	DailyDose	Google Calendar	Outlook Calendar
Purpose	Simple, personal scheduling	Consumer-grade scheduling within Google ecosystem	Enterprise-grade scheduling within Microsoft 365
Platform	Browser-based	Web, Android, iOS	Web, Android, iOS, Windows
Architecture	MVC model	Distributed Microservices (client server + repository)	Client Server (repository + layered architecture)
Features	Add/edit/delete events, conflict detection, simple log-in/sign-up, clean UI	Smart suggestions, reminders, auto-sync, shared calendars	Meeting invites, team scheduling, email/calendar integration
Strengths	Lightweight, beginner-friendly, minimal to no distractions	Deep integration with Google services (Gmail, Meet, Tasks, Contacts)	Powerful collaboration features more suitable for corporations
Limitations	No sync or collaboration	Heavily reliant on Google ecosystem, can feel cluttered	Complex features, most tools are not necessary for average users

TESTING

We selected one of the most important components to implement a snippet of code for an test: time conflicts.

```
//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);
}
```

TESTING

We tested if there was an already existing event or not before adding our new event. We also tested to make sure our event was actually being added to the list of events after we add it in our main function

```
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));
    }
}
```

In Conclusion

- We spent the most amount of time determining the functional requirements, and deciding on what we want our project to look like.
  - This illustrates real life struggles in different expectations for an application.
- Even a simple implementation of a calendar app goes a long way in demonstrating real life software engineering processes