

CS3354 Software Engineering
Final Project Deliverable 2

Calendar Software Project

Robert Berghian, Henry Jones, Josh Pahman, Shelby Pham,
Illiana Rodriguez Huerta, Isaiah Shadd, Nathaniel Tan

1. Delegation of Tasks

1. Robert Berghian: Tasked with delivering sequence diagrams for each use case in Deliverable 1 and estimating the costs associated with hardware, software, and personnel for Deliverable 2. Additionally, he was responsible for the submission of all documentation through eLearning. For the proposal draft, he participated in the team's decision-making process regarding the project's objectives and motivation.
2. Henry Jones: Tasked with listing the functional software requirements for Deliverable 1 and estimating the project schedule and cost calculation method for Deliverable 2. His role involved providing detailed insights into scheduling considerations, including the inclusion of weekends and daily working hours. In the proposal draft, he collaborated with the team to define the project's objectives and motivations.
3. Josh Pahman: Tasked with identifying and justifying the software process model for Deliverable 1 and developing a comprehensive test plan with actual coding and testing using JUnit for Deliverable 2. He also integrated the unit tests into the team's GitHub repository. Like the rest, he collaborated on the proposal draft and contributed to the project's objectives and motivations.
4. Shelby Pham: Tasked with delivering a use case diagram and applying the Model-View-Controller (MVC) architectural pattern for Deliverable 1. In deliverable 2, she thoroughly compared the project with similar designs, citing references. For the proposal draft, she created the team's GitHub repository and collaborated in the decision-making regarding the project's objectives and motivations.
5. Illiana Rodriguez Huerta: Tasked with listing the non-functional software requirements for Deliverable 1 and evaluating the team's work in the conclusion for Deliverable 2, including any changes that had to be made and their justifications. She also maintained the overall organization of the project's documentation. In the proposal draft, she worked with the team to define the project's objectives and motivations.
6. Isaiah Shadd: Tasked with providing a unique class diagram for Deliverable 1, including all classes with cardinalities and relationship types. For Deliverable 2, he worked on the estimates for the project scheduling and project timeline. He also collaborated with the team on the proposal draft and helped define the project's objectives and motivations.
7. Nathaniel Tan: Tasked with addressing the feedback of the Final Project Proposal and discussing our plan moving forward for Deliverable 1. For Deliverable 2, he was in charge of creating the team's presentation slides with all our project's information. In the proposal draft, he also collaborated with the team to define the project's objectives and motivations.

2. Project Deliverable 1 Content

What We Are Doing

A versatile calendar software designed to offer a range of essential features for effective time management and organization. It will include various viewing options, each displaying events and schedules. Users will be able to add, edit, delete, and categorize events, with built-in conflict checking and event alerts. The software will also support the creation of recurring events and color-coding for event categories. Moreover, it will provide color highlights for holidays and weekends, and will support zooming and scrolling for enhanced usability.

Motivation

Our main motivation is to provide an efficient and user-friendly tool to help individuals better manage their schedules and events. We chose this project because we have experienced first hand the struggle that comes with personal time management, work scheduling, and coordinating with others. Thus, we will address the common challenges that people face in hopes to provide a calendar software that will enhance productivity and user experience.

Delegated Tasks

- Henry Jones: Will serve as the scrum master, responsible for guiding the project's overall development process.
- Josh Pahman: Responsible for guiding the backend development of the project.
- Robert Berghian: Responsible for helping with the backend components of the project. Will also be in charge of turning in the documents.
- Illiana Rodriguez Huerta: Responsible for guiding the frontend development.
- Shelby Pham: Responsible for helping with the frontend components of the project.
- Isaiah Shadd: Responsible for guiding the UX development of the project.
- Nathaniel Tan: Responsible for helping with the UX components of the project.

Addressing Feedback

The feedback we were provided on the Final Project Proposal advised us to compare our calendar application with other applications that have similar functionalities and to differentiate our application from them.

One of the most popular calendar applications, Google Calendar, already has many capabilities, but a few functions can still be optimized. For example, there are many more options to customize an event that a user adds to their calendar than a task. These options range from specifying when a reminder notification will be sent out before the event, adding guests, and finding a time to place the event within the user's existing schedule.

Building upon the functionalities for tasks and bringing them closer to that of events is one of the primary ways we are aiming to differentiate our app from other popular existing apps such as Google Calendar, but it doesn't end there. There is the possibility for us to add more unique features later on and focus on other ways of evolving the design of our calendar app.

Team Project Github Repository

<https://github.com/ncp9988/3354-TimeKeepers>

Delegation of Tasks

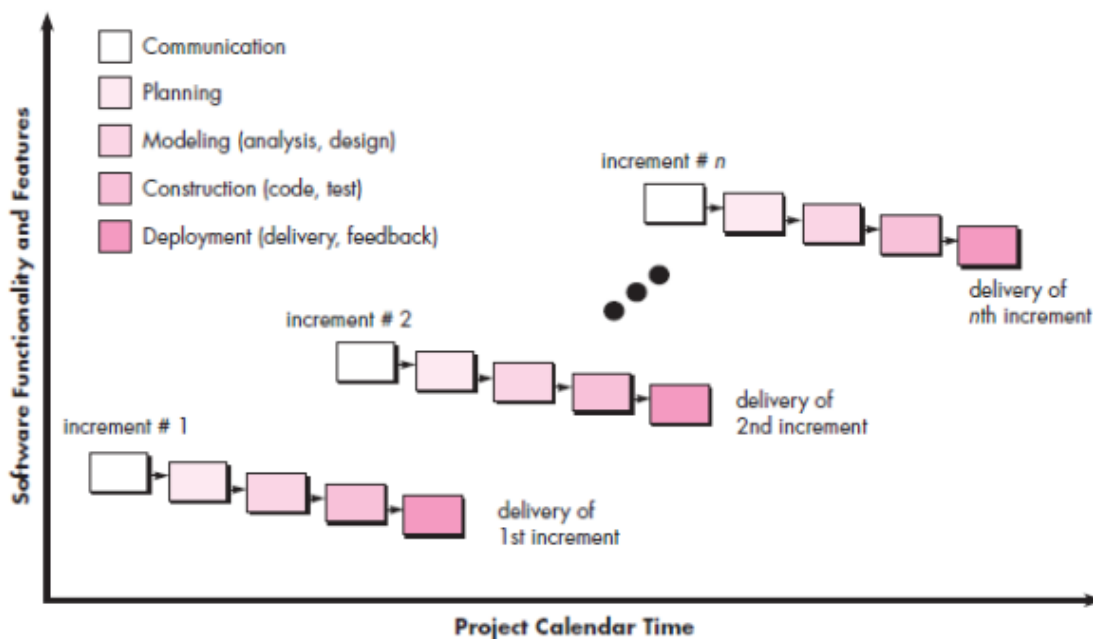
- a. Robert Berghian: Sequence Diagram
- b. Henry Jones: Functional requirements
- c. Josh Pahman: Choosing software process model
- d. Shelby Pham: Case diagram, Architectural Design
- e. Illiana Rodriguez Huerta: Non-functional requirements
- f. Isaiah Shadd: Class Diagram
- g. Nathaniel Tan: Addressing proposal feedback

Software Process Model

For our project we will be using an **incremental process model**. The reasoning for this is that the features of calendar software can be easily broken into incremental feature additions, such as the following:

1. **Increment 1**: Add daily, weekly, and monthly views alongside basic event support, providing basic calendar functionality.
2. **Increment 2**: Add an agenda view as well as checking for time conflicts, periodical events, editing and deleting events, and event alerts.
3. **Increment 3**: Implement adding/deleting event categories, color marking for different categories of events, sending events to other calendar users, holidays and weekends in special colors, and zoom-in/zoom-out scroll support.

Each of these increments consists of a miniature version of the waterfall model, with each increment including communication, planning, modeling, construction, and deployment phases as depicted in the diagram below:



Software Requirements

a. Functional Requirements:

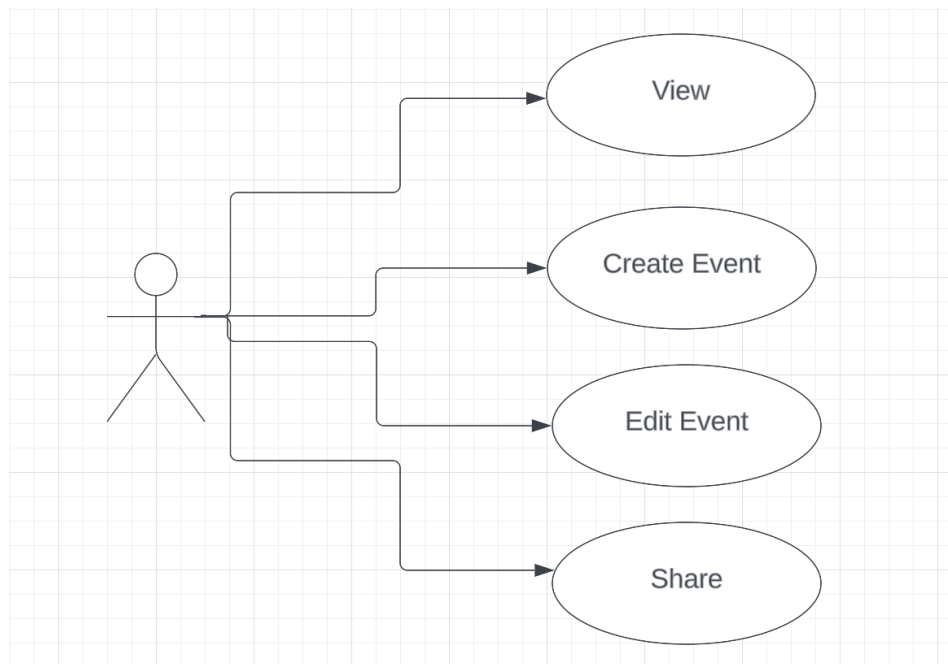
1. **Event Scheduling:** The software's most basic feature is scheduling events to the calendar. The user should be able to specify times and dates, locations, durations, repetitions, and a description of the event of their choice. The software should know if an event will conflict with an existing event and alert the user if it's the case.
2. **Multiple Calendar Views:** Monthly view: show all days in a month, and event snippet for each day, Weekly view: show all days in a week, and event snippet for each day, Daily view: show all events in a day, sorted by their starting time, Agenda view (optional): show all events in future as a list.
3. **Alerts:** The software should be able to notify the user within a set time period of the event scheduled.
4. **Search:** The software should have a search function that lets the user use basic keyword searching to find past events using the name of the event and/or the description of the event.
5. **Sharing:** The software should allow events to be shared between other users of the software. This would be facilitated by sharable internet links.
6. **Data Import/Export:** To be able to keep the software compatible with other calendar softwares, the software should have the ability to export calendar data in a common readable format like CSV or Apple's iCal.

b. Non-functional Requirements:

- **Product Requirements:** The calendar software should be accessible to everyone 24 hours 7 days a week, allowing users to manage their schedules at any time.
 - **Usability Requirements:** The software should feature an intuitive and user-friendly interface with clear instructions, tooltips, and user testing.
 - **Efficiency Requirements:** Event creation, editing, and deletion should be processed efficiently, even when managing a large number of events.
 - **Performance Requirements:** The software should have a maximum response time of 2 seconds for user interactions, load calendar views within approximately 3 seconds, and support up to 500 simultaneous users.
 - **Space Requirements:** The software should optimize data storage to efficiently manage user data and events.
 - **Dependability Requirements:** The software should maintain high availability, with a minimum uptime of 99.9%, and ensure data integrity to prevent loss of user event data.
 - **Security Requirements:** The software should prioritize user authentication and authorization, employ industry-standard encryption for data transmission, and implement protections against cross-site scripting (XSS) and SQL injection.
- **Organizational Requirements:** Users should identify themselves using their identity credentials.

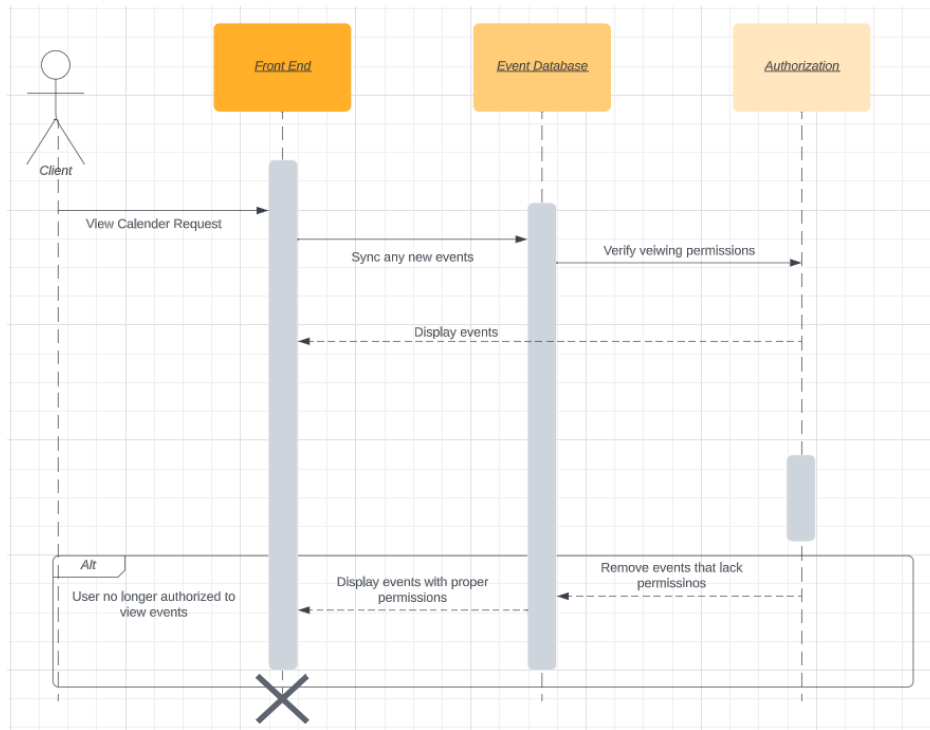
- **Environmental Requirements:** The software should prioritize energy efficiency and reduced environmental impact, adhering to the Software Energy Efficiency Act (SEEA-2023). *Assumption*
- **Operational Requirements:** The software should seamlessly integrate with the organization's existing infrastructure and systems while maintaining routine backup and recovery procedures.
- **Development Requirements:** Developers should adhere to organizational coding and development standards mentioned in the Code Review and Quality Assurance Procedures (CRQAP-2023). *Assumption*
- **External Requirements:** The software should adhere to data protection regulations, ethical standards, and legislative requirements.
 - **Regulatory Requirements:** The software should comply with the Data Privacy Act (DPA-2023). *Assumption*
 - **Ethical Requirements:** The software should adhere to ethical standards, particularly in the handling of personal and sensitive user data
 - **Legislative Requirements:** The software should comply with the User Consent and Data Handling Regulation (UCDHR-2023). *Assumption*
 - **Accounting Requirements:** The software should ensure accurate accounting and reporting of user subscriptions.
 - **Safety/Security Requirements:** The software should provide secure and reliable access to the user's calendar data, with safeguards against data loss or corruption.

Case Diagram

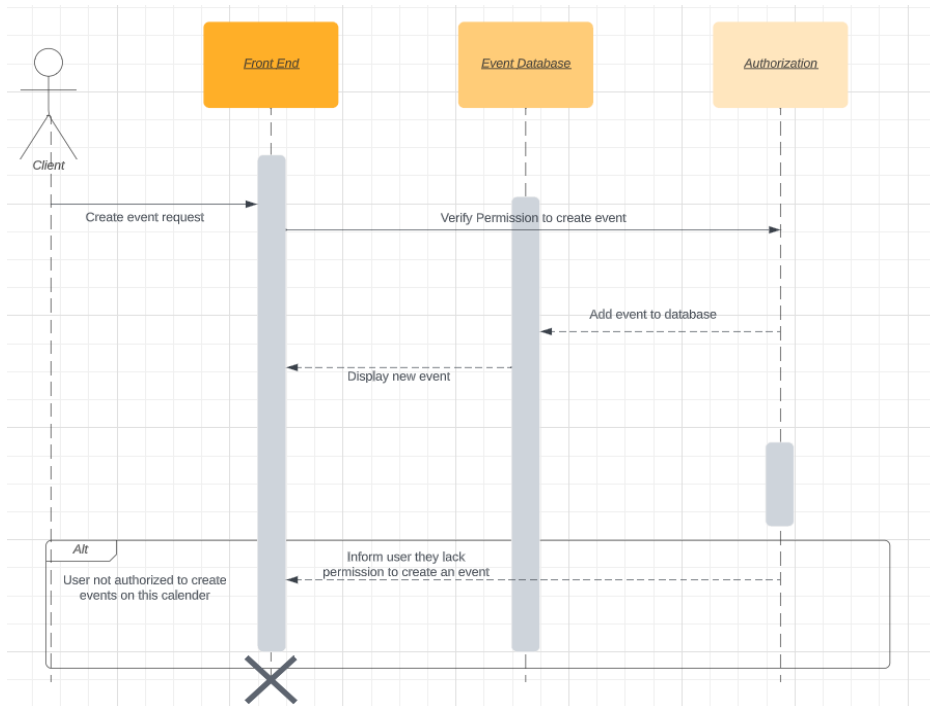


Sequence Diagrams

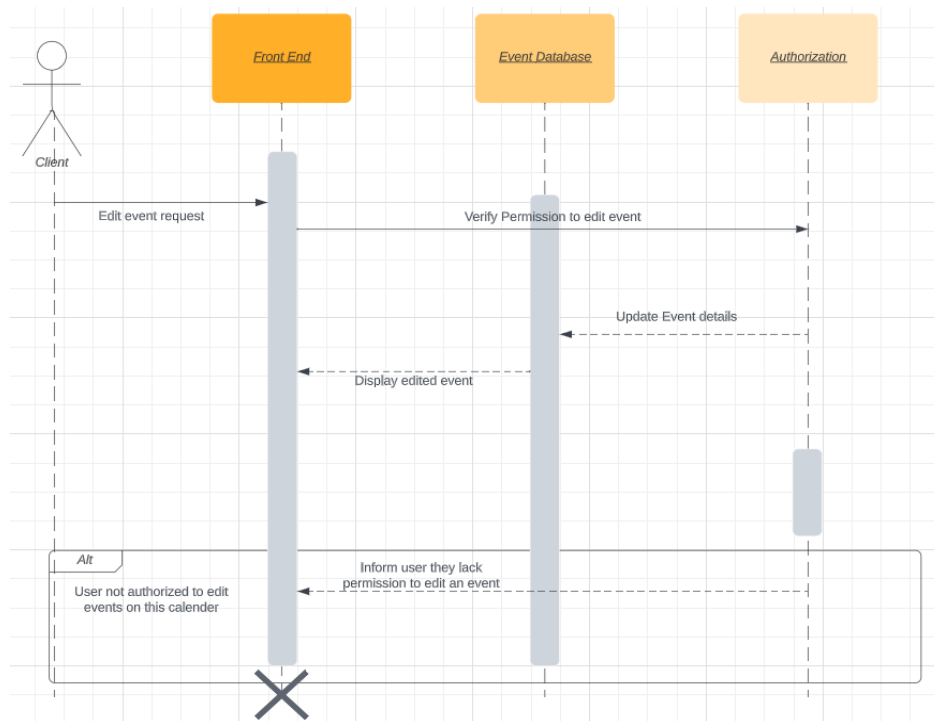
View:



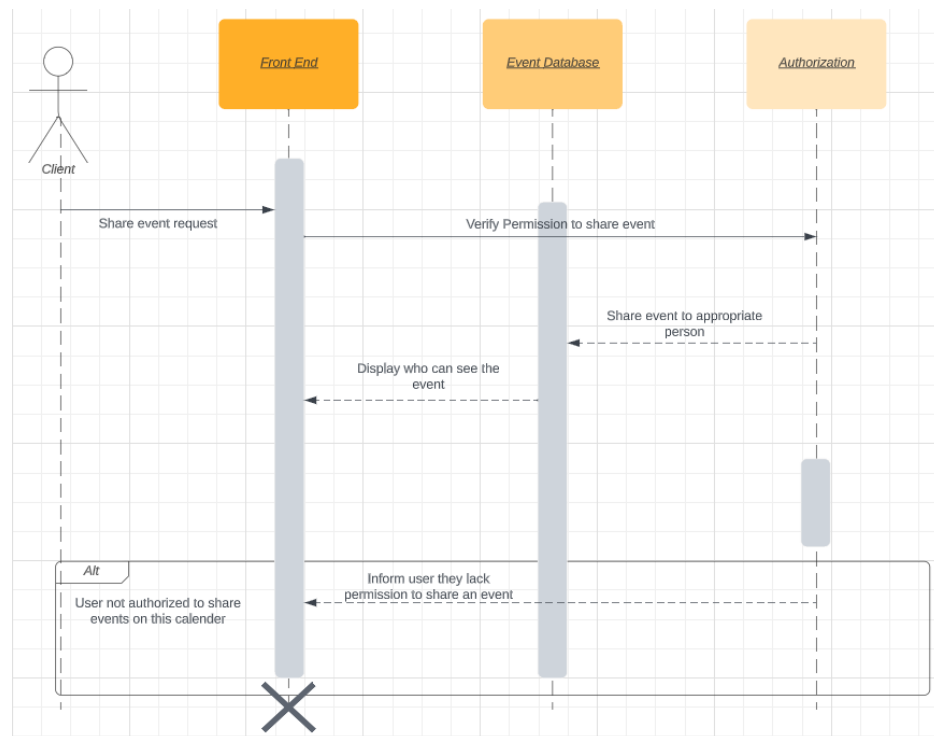
Create Event:



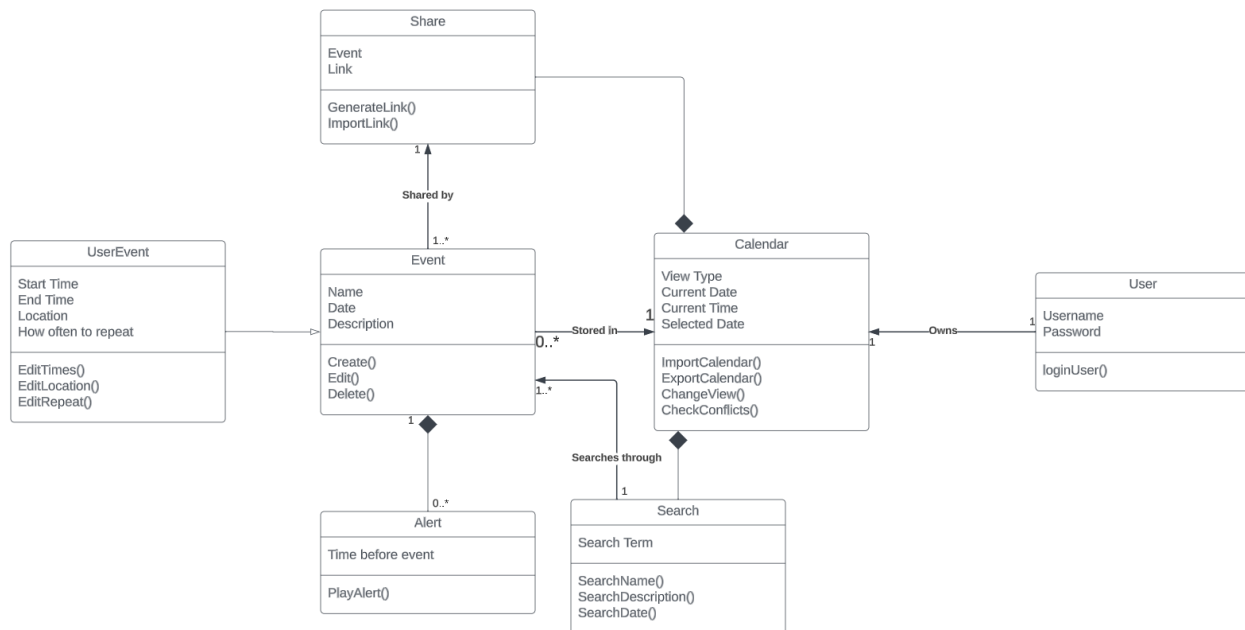
Edit Event:



Share:



Class Diagram



Architectural Design

Model-View-Controller (MVC) pattern:

- Model (M)**: The Model component of Calendar app is responsible for managing the application's data and business logic.
 - Event Data**: Create a structured representation for events. Include title, date, time, location, and any other event-specific information.
 - Calendar Data**: Manage data related to the calendar, such as date ranges, holidays, and special days. This data is used to render the calendar grid and handle exceptional events.
- View (V)**: The View component handles the presentation of data to the user and deals with the user interface (UI).
 - Calendar Interface**: Design the user interface elements to display the calendar grid, navigation controls, and event details.
 - Event Details View**: Create the user interface elements that will display the calendar grid, navigation controls, and event information.
 - Event Creation and Editing Forms**: Create forms or dialog boxes that allow users to add, edit, or delete events. These forms should guide users through the event creation and update processes.
 - Notifications**: Create a system that displays reminders and notifications for impending events or special days, keeping users informed of crucial dates and appointments.
- Controller (C)**: The Controller acts as an intermediary between the Model and the View. It handles user interactions, processes user input, and updates the Model and View accordingly.

- Event Handling: To react to user activities like clicking on a date or creating, amending, or removing events, implement event listeners. The Model and View are updated in response to these actions.
- Data Validation: Ensure that user inputs, such as event details or date selections, are valid and within the expected range. This step helps maintain data integrity.
- Data Flow: In order to ensure that any changes made in one component are correctly reflected in the other, manage the data flow between the Model and View. The calendar user interface must be kept up to date with the underlying data.
- Navigation: Organize user navigation inside the software, including changing between month, week, and day views and advancing and regressing in time. For users to interact with the calendar easily, navigation must be effective.
- User Authentication: Implement user authentication and access control to protect sensitive data and limit certain actions to authorized users if the Calendar app needs user-specific functionality or event management.

3. Project Scheduling, Cost, Effort and Pricing Estimation

1. Project Scheduling

We have estimated 3 weeks for this project. Based on our effort calculations, the prototype for our project takes approximately 0.64706 person-months to complete. If we have a small development team of 3-5 developers, then it will take them around 1 standard work week, consisting of five, eight-hour work days, not including weekends. From there, the team has 2 additional weeks to test, debug, and finalize the project.

Start Date: 11/27/2023

End Date: 12/15/2023

2. Cost, Effort and Pricing Estimation

Our team has decided on using the Application Composition Model to represent our project as well as reusing 50% of objects from a GUI library.

Step 1:

- Number of Screens = 4 (Calendar, Create Event, Edit Event, Share Event)
- Number of Reports = 2 (User Information, Event Information)

Step 2:

Screens:

1. Calendar:

- Number of Views = 4 (Monthly Summary, Weekly Summary, Daily Summary, Daily Agenda)
- Number of Data Tables = 1 (Event Database)
- Number of Servers = 1 (Event Information Database)
- Number of Clients = 1 (Calendar Application)
- Complexity Level: Simple

2. Create Event:
 - Number of Views = 1 (Create Event)
 - Number of Data Tables = 1 (Event Database)
 - Number of Servers = 1 (Event Information Database)
 - Number of Clients = 1 (Calendar Application)
 - Complexity Level: Simple
3. Edit Event:
 - Number of Views = 1 (Edit Event)
 - Number of Data Tables = 1 (Event Database)
 - Number of Servers = 1 (Database Server)
 - Number of Clients = 1 (Calendar Application)
 - Complexity Level: Simple
4. Share Event:
 - Number of Views = 1 (Share Event)
 - Number of Data Tables = 2 (Event Information Database, User Information Database)
 - Number of Servers = 1 (Database Server)
 - Number of Clients = 1 (Calendar Application)
 - Complexity Level: Simple

Total:

- Number of Views = 7 (Monthly Summary, Weekly Summary, Daily Summary, Daily Agenda, Create Event, Edit Event, Share Event)
- Number of Data Tables = 2 (Event Information Database, User Information Database)
- Number of Servers = 1 (Database Server)
- Number of Clients = 1 (Calendar Application)
- Complexity Level: Simple

Reports:

1. User Information:
 - Number of Sections = 3 (Email, Password, Account Identifier)
 - Number of Data Tables = 1 (User Information)
 - Number of Servers = 1 (Database Server)
 - Number of Clients = 1 (Calendar Application)
 - Complexity Level: Simple
2. Event Information:
 - Number of Sections = 6 (Time, Date, Location, Duration, Repetitions, Description)
 - Number of Data Tables = 1 (Event Information)
 - Number of Servers = 1 (Database Server)
 - Number of Clients = 1 (Calendar Application)
 - Complexity Level: Medium

Total:

- Number of Sections = 9 (Email, Password, Account Identifier, Time, Date, Location, Duration, Repetitions, Description)
- Number of Data Tables = 2 (User Information, Event Information)
- Number of Servers = 1 (Database Server)
- Number of Clients = 1 (Calendar Application)
- Complexity Level: Medium

Step 3:

Screens:

1. Calendar: Object points: 1
2. Create Event: Object points: 1
3. Edit Event: Object points: 1
4. Share Event: Object points: 1

Total Screen Object Points: 4

Reports:

1. User Information: Object points: 2
2. Event Information: Object points: 5

Total Report Object Points: 7

Step 4:

Total Object Points: $4 + 7 = 11$

Step 5:

Reusing an estimated 50% of objects using a GUI library

$$\text{NOP} = (\text{OP} * (100 - \% \text{Reuse})) / 100$$

$$\text{NOP} = (11 * (100 - 50)) / 100 = 5.5$$

Step 6:

The development environment is average (nominal: 13), but the team does not have much experience in application development (very low: 4)

$$\text{PROD} = (4 + 13) / 2 = 8.5$$

Step 7:

$$\text{Effort} = \text{NOP} / \text{PROD}$$

$$\text{Effort} = 5.5 / 8.5 \cong 0.64706 \text{ person-month}$$

This project will cost approximately **0.64706 person-month** to complete.

3. Estimated Cost of Hardware Products

Utilizing AWS for storage, it costs \$0.023 per Gb of storage used for the first 50 TB of data each month, assuming 1 million active users per month, with an average of 180 events per month (6 events / reminders per day), that is around 80 GB of storage used across all million users. This is not including the fact that sharing events does not create a

separate instance, so actual storage usage would be lower. This would come out to \$18.40 a month in storage costs for our event database.

User Database: The user database is very small, for one million users it would only take up 0.02 GB of storage, which would cost \$0.0005175 a month.

There will also be around \$5000 in computers purchased for the developers.

4. Estimated Cost of Software Products

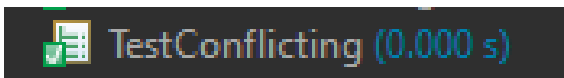
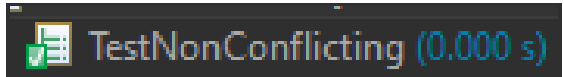
We decided to use AWS for our database and API calls / communication. This is going to be the bulk of the cost of operation. Assuming the 6 events / reminders a day. Assuming absolute worst case of 6 million HTTPS API calls, 6 million REST API calls, and 6 million Websocket messages a day. The cost of operation would be around \$1003.75 monthly. This also assumes that API calls are made very efficiently, with every action requiring its own API call. If we take a more conservative estimate of 3 million, 2 million, and 1 million requests respectively, this cost falls down to a more manageable \$334 a month. And if we assume that we sync every few minutes, and combine changes / events into one larger sized API call (cut the number of calls in half), this makes our cost only \$167 a month.

5. Estimated Cost of Personnel

Assuming a 5 junior developer team getting paid freelance of \$50 an hour, for the 3 weeks estimated for the project completion. This adds up to be \$30,000 for developer salary. Assuming we can fall back to only 2 active developers for maintenance after completion, this would add up to \$13,500 a month in developer salary. As well as \$7,500 a month for a manager.

4. Test Plan for Software

The test plan for our software involves using JUnit to unit test the various units of our calendar software. For an example of what these tests would look like, test cases and results for the Event.EventsConflict() method are given below. This method takes an ArrayList of Event objects and returns whether any of the given events have conflicting time slots. The code for these tests is attached.

| Test Case for Event.EventsConflict() | Result |
|---|--|
| TestConflicting(): <pre>assertEquals("Here is the test for conflicting events", true, Event.EventsConflict(eventList));</pre> |  |
| TestNonConflicting(): <pre>assertEquals("Here is the test for</pre> |  |

| | |
|--|--|
| <pre>non-conflicting events", false, Event.EventsConflict(eventList));</pre> | |
|--|--|

Additionally, it would also be beneficial for us to take advantage of things like integration and regression testing to ensure the software functions as expected, but these are not able to be fully or appropriately considered given that we are not producing an implementation.

5. Comparing Our Work with Similar Designs

- Google Calendar: [1]
 - **Features:** Google Calendar is a widely used calendar application with robust features, including event scheduling, multiple calendar views, event alerts, and sharing options.
 - **Differentiation:** Our project aims to differentiate itself by improving the functionality for tasks, making it more similar to events, and offering additional unique features in the future.
- Microsoft Outlook Calendar: [2]
 - **Features:** Microsoft Outlook Calendar offers features such as event scheduling, multiple calendar views, event alerts, and integration with other Microsoft Office apps.
 - **Differentiation:** we may stand out by offering a more intuitive and user-friendly interface or by focusing on cross-platform compatibility.
- Apple Calendar: [3]
 - **Features:** Apple Calendar provides event scheduling, multiple views, and integration with other Apple devices and apps.
 - **Differentiation:** Our project may differentiate itself by focusing on features that enhance productivity, such as efficient event creation and data export options.

6. Conclusion

Our team has made significant progress in developing a versatile calendar software, outlining detailed functional and non-functional requirements, adopting the MVC pattern, and selecting an incremental process model. The delegation of tasks and responsibilities among our members was clear, with each of us equally contributing to specific aspects. Our team addressed feedback received during the proposal stage, particularly focusing on differentiating our calendar application from popular competitors like Google Calendar. While a detailed project plan, including a timeline and cost estimation, has been established, we've acknowledged the potential for changes and adaptations as the project unfolds in the future. Moreover, the GitHub repository has been actively maintained, and we utilized JUnit for comprehensive unit testing. Overall, our project's emphasis on user-friendly interfaces, security measures, and adherence to various regulations demonstrates a comprehensive approach to software development. We believe that continuous improvement and adaptability are key principles for our team moving forward.

7. References

- [1] “Google Calendar,” Wikipedia, Mar. 06, 2022. https://en.wikipedia.org/wiki/Google_Calendar
- [2] “Welcome to your Outlook calendar - Microsoft Support,” support.microsoft.com. <https://support.microsoft.com/en-us/office/welcome-to-your-outlook-calendar-6fb9225d-9f9d-456d-8c81-8437bfcd3ebf> (accessed Apr. 24, 2023).
- [3] “Apple Calendar Guide: Everything You Need to Know About iCal,” Calendar. <https://www.calendar.com/apple-calendar/>