CS3354 Software Engineering
Final Project Deliverable 2

CalTool

Yousuf Jazzar
Christopher Oliva
Kyle Custodio
Kent Templin
Albert Jean
Kason Alstatt
Piyush Mewada

Delegation of tasks from beginning of the project until the end:

Kent Templin:

1. Created and organized GitHub repository
2. Drew Use Case Diagram, and Class Diagram for Project Deliverable 1.
3. Wrote software process model, functional requirements, and non-functional requirements for Project Deliverable 1 report.
4. Wrote 50 % of Project Deliverable 2 document, like project scheduling, cost, effort, pricing estimation, and duration.

Kyle Custodio:

1. Drew 3 sequence diagrams for Project Deliverable 1.
2. Created a slide for Project Deliverable 2.

Christopher Oliva:

1. Drew 3 sequence diagrams for Project Deliverable 1.
2. Drew architectural model for Project Deliverable 1.
3. Created a slide for Project Deliverable 2.
4. Presentation Script

Yousuf Jazzar:

1. Drew 2 sequence diagrams for Project Deliverable 1.
2. Created CalTool program with Java for Project Deliverable 2.
3. Helped with deliverable files
4. Added JUnit testing to CalTool code
5. Added JUnit testing to presentation

Albert Jean:

1. Drew 2 sequence diagrams for Project Deliverable 1.
2. Created CalTool program with Java for Project Deliverable 2.
3. Added JUnit testing to CalTool code
4. Added JUnit testing to presentation

Kason Alstatt:

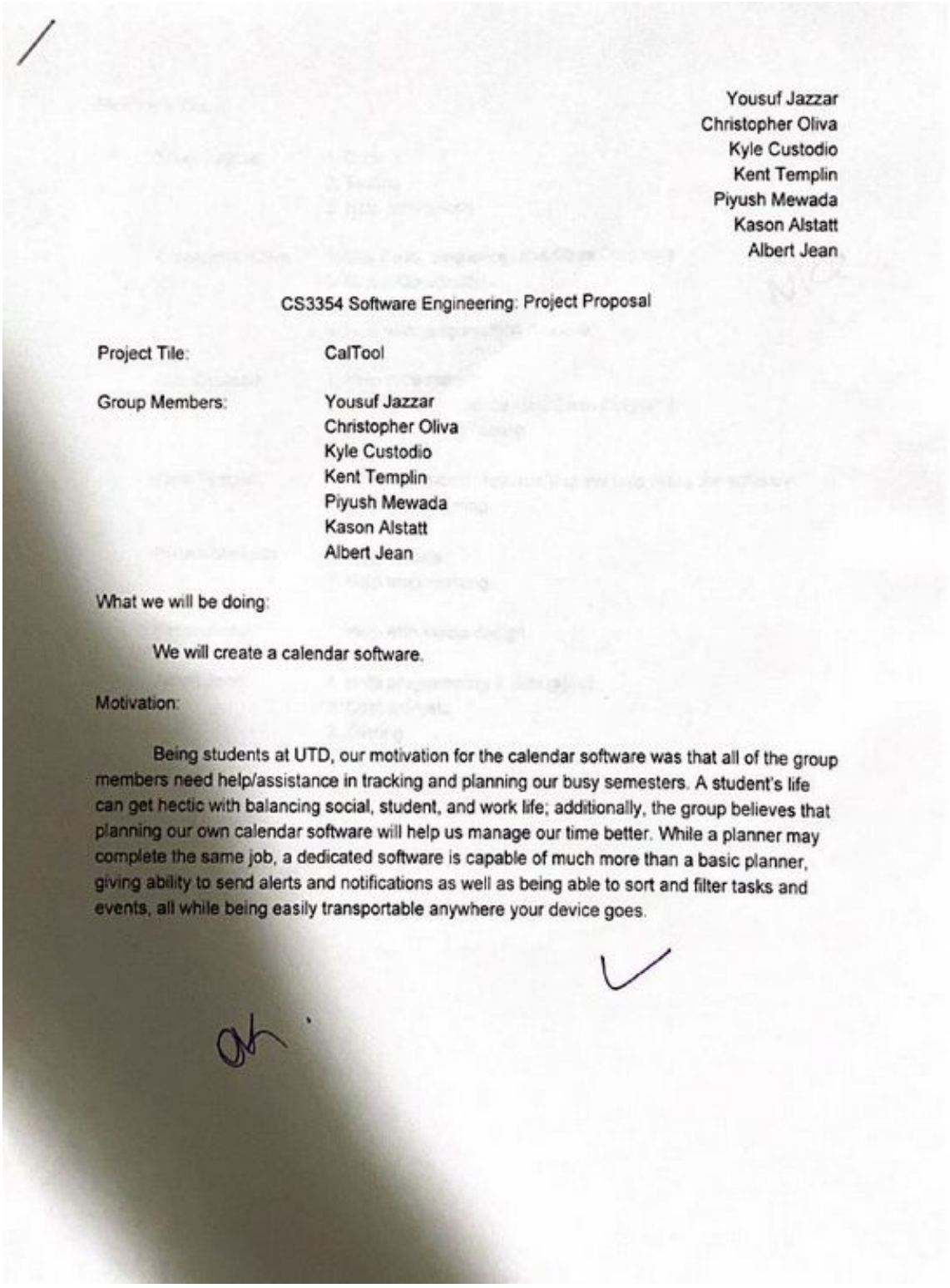
1. Drew 2 sequence diagrams for Project Deliverable 1.
2. Researched and wrote similar design of CalTool for Project Deliverable 2 report.

Piyush Mewada:

1. Drew first use case diagram for Project Deliverable 1.
2. Presentation of Project Deliverable 2.

Addressing the Feedback:

- No feedback from the instructor. (Only “OK” and “nice”.)



Yousuf Jazzar
Christopher Oliva
Kyle Custodio
Kent Templin
Piyush Mewada
Kason Alstatt
Albert Jean

CS3354 Software Engineering: Project Proposal

Project Tile: CalTool

Group Members: Yousuf Jazzar
Christopher Oliva
Kyle Custodio
Kent Templin
Piyush Mewada
Kason Alstatt
Albert Jean

What we will be doing:

We will create a calendar software.

Motivation:

Being students at UTD, our motivation for the calendar software was that all of the group members need help/assistance in tracking and planning our busy semesters. A student's life can get hectic with balancing social, student, and work life; additionally, the group believes that planning our own calendar software will help us manage our time better. While a planner may complete the same job, a dedicated software is capable of much more than a basic planner, giving ability to send alerts and notifications as well as being able to sort and filter tasks and events, all while being easily transportable anywhere your device goes.

Member's Tasks:

Yousuf Jazzar:	1. Coding 2. Testing 3. Help with reports
Christopher Oliva:	1. Use Case, Sequence, and Class Diagrams 2. Group Coordinator 3. Help with Visuals 4. Help with programming if needed
Kyle Custodio:	1. Help write reports 2. Use Case, Sequence, and Class Diagrams 3. Implementation/Testing
Kent Templin:	1. Find information / tutorials that will help make the software 2. Help programming
Piyush Mewada:	1. Help Visuals 2. Help programming
Kason Alstatt:	1. Help with media design
Albert Jean:	1. Help programming & debugging 2. Cost analysis 3. Testing

Nice -

Member's Tasks for Deliverable 1:

Yousuf Jazzar:
1. Coding
2. Testing
3. Help with reports

Christopher Oliva:
1. Use Case, Sequence, and Class Diagrams
2. Group Coordinator
3. Help with Visuals
4. Help with programming if needed

Kyle Custodio:
1. Help write reports
2. Use Case, Sequence, and Class Diagrams
3. Implementation/Testing

Kent Templin:
1. Organize GitHub
2. Use Case and Class Diagrams
3. Find information / tutorials that will help make the software
4. Help programming

Piyush Mewada:
1. Help Visuals
2. Help programming

Kason Alstatt:
1. Help with media design

Albert Jean:
1. Help programming & debugging
2. Cost analysis
3. Testing

Software Process Model:

- Incremental process model

Reasons:

- This model saves money and time for software engineering.
- Software Engineers can add features while they program.
- This process model is used in software engineering often, so it is practical to use it.

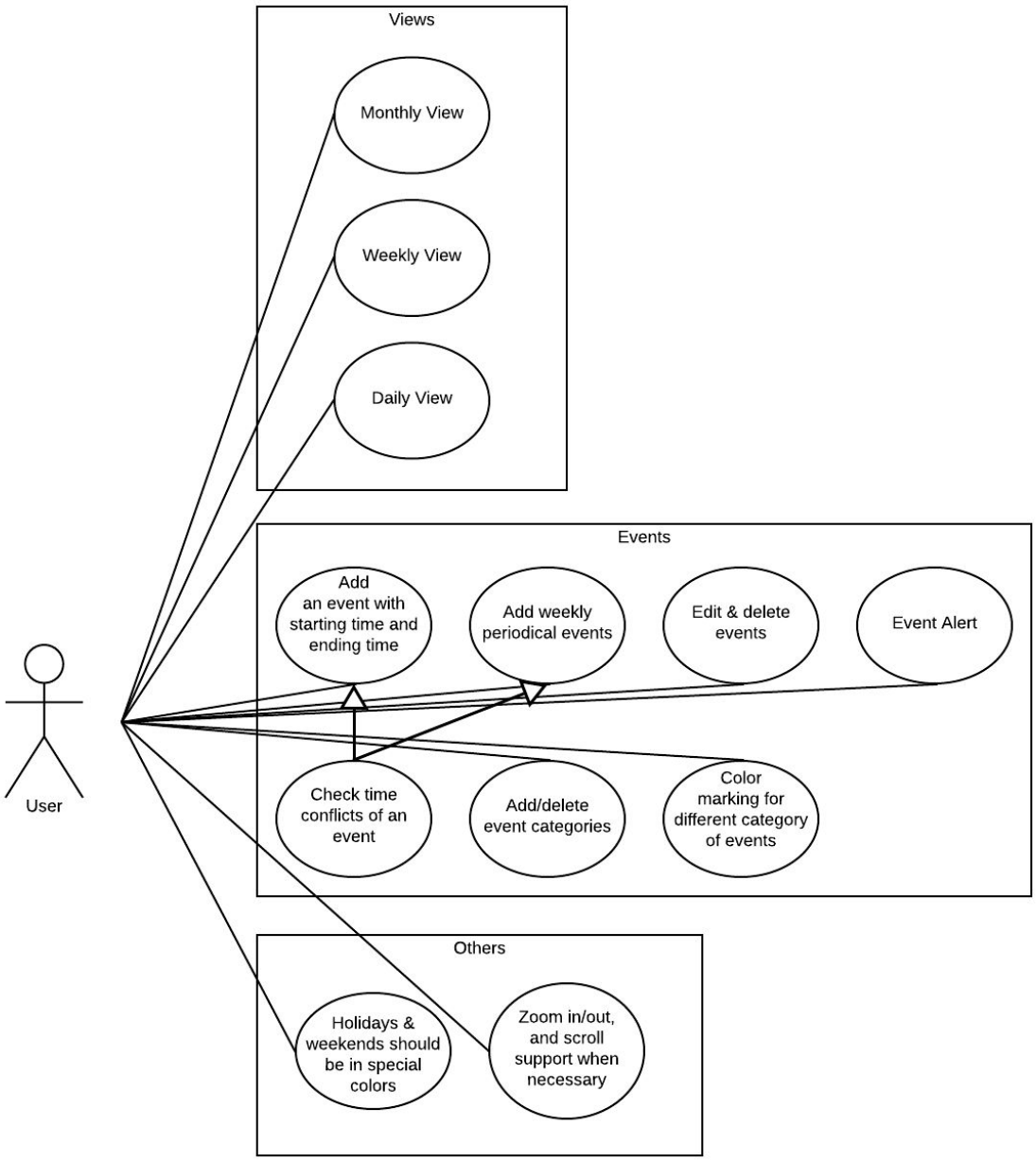
Functional requirements:

- The program needs to be able to display a calendar with dates, a current date, and events that the user has provided.
- The program needs to be able to display a calendar in either Monthly View (shows the current month calendar), Weekly View (shows the current week calendar), or Daily View (shows the current day calendar).
- The program needs to be able to read multiple event data from the user.
- The program needs to be able to store and save multiple event data.
- The program needs to be able to send a notification to the user when an event is on the current date.

Non-functional requirements:

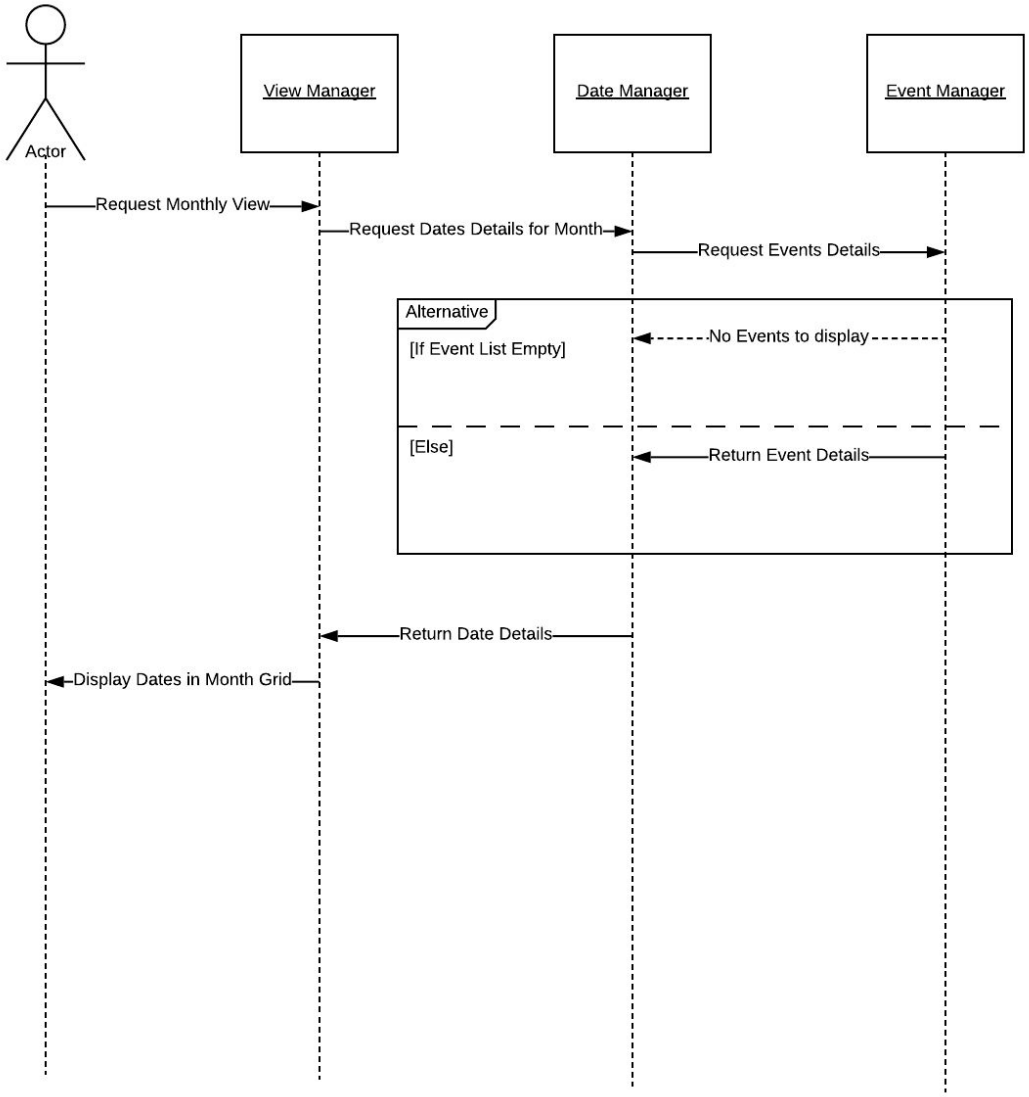
- The program will run in JAVA / C / C++ / Python.
- The total program size will be less than 200 MB.
- The program needs to be able to open in less than 1 second.
- The program needs to be able to read the users input in less than 1 second.
- The program will be able to store 100 years of dates.
- The program needs to be able to store and save at least 100 events from the user's input. Meaning about 100 string data. Meaning at least about 800 bytes. (100 strings * 8 bytes = 800 bytes)

Use case diagram:

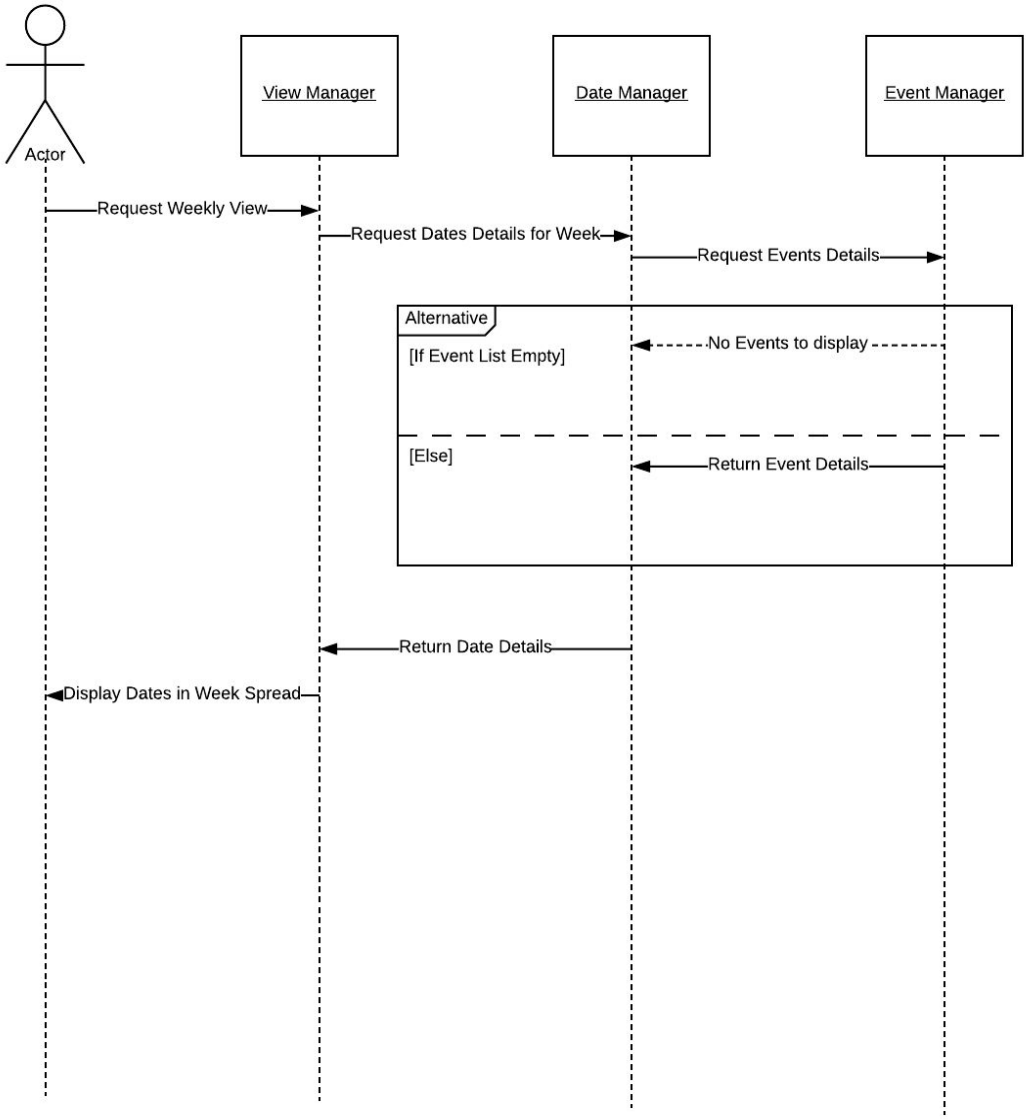


Sequence diagram:

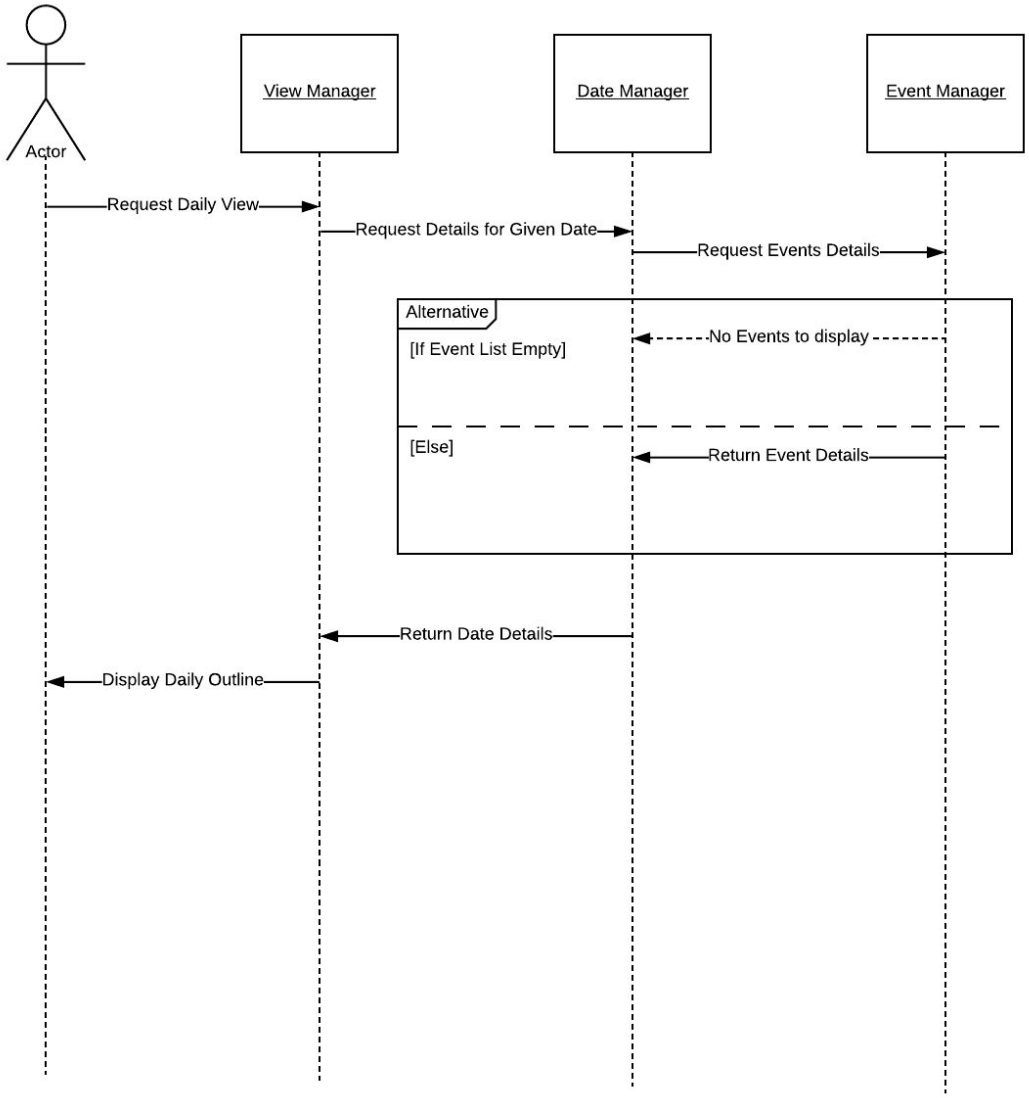
- 1. Monthly View



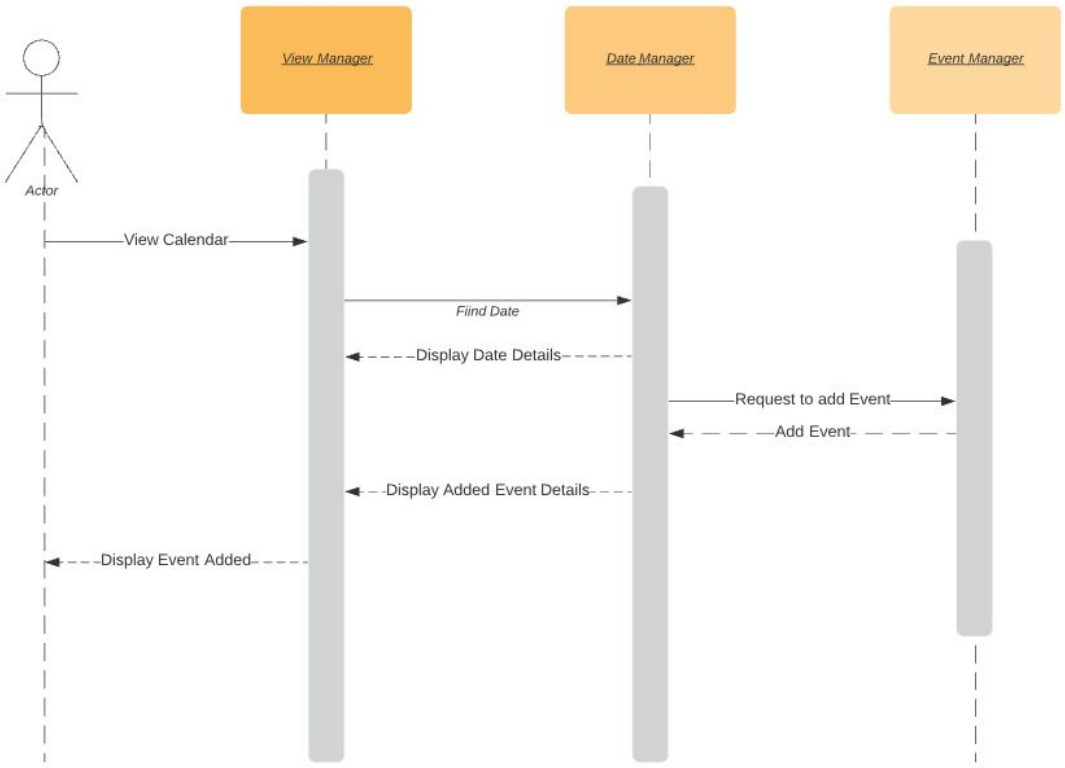
- 2. Weekly View



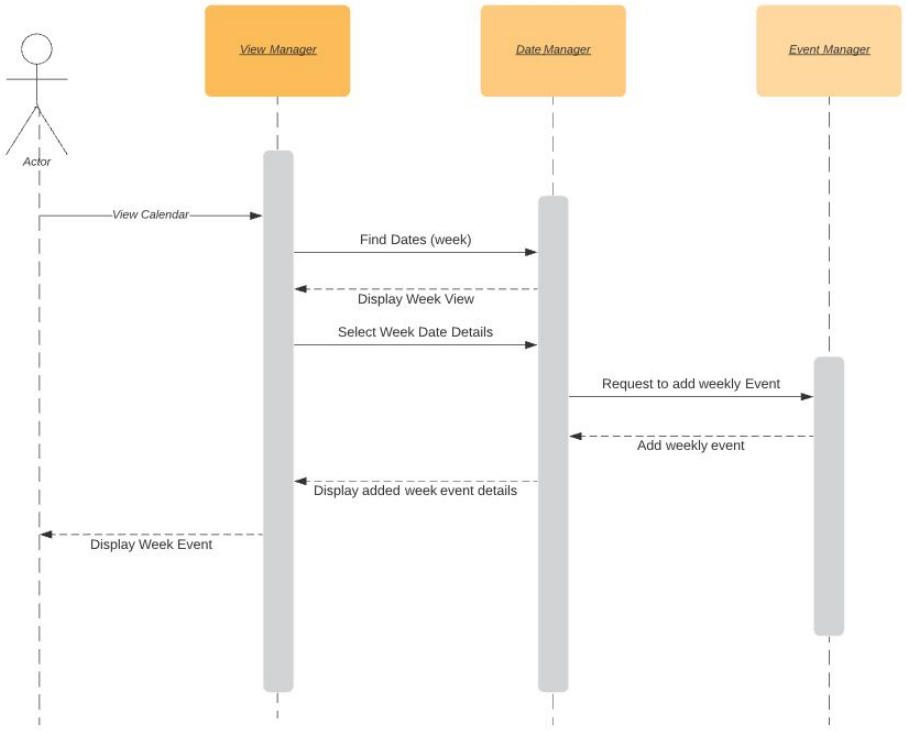
- 3. Daily View


Actor
View Manager
Date Manager
Event Manager
Request Daily View
Request Details for Given Date
Request Events Details
Alternative
[If Event List Empty]
No Events to display
[Else]
Return Event Details
Return Date Details
Display Daily Outline

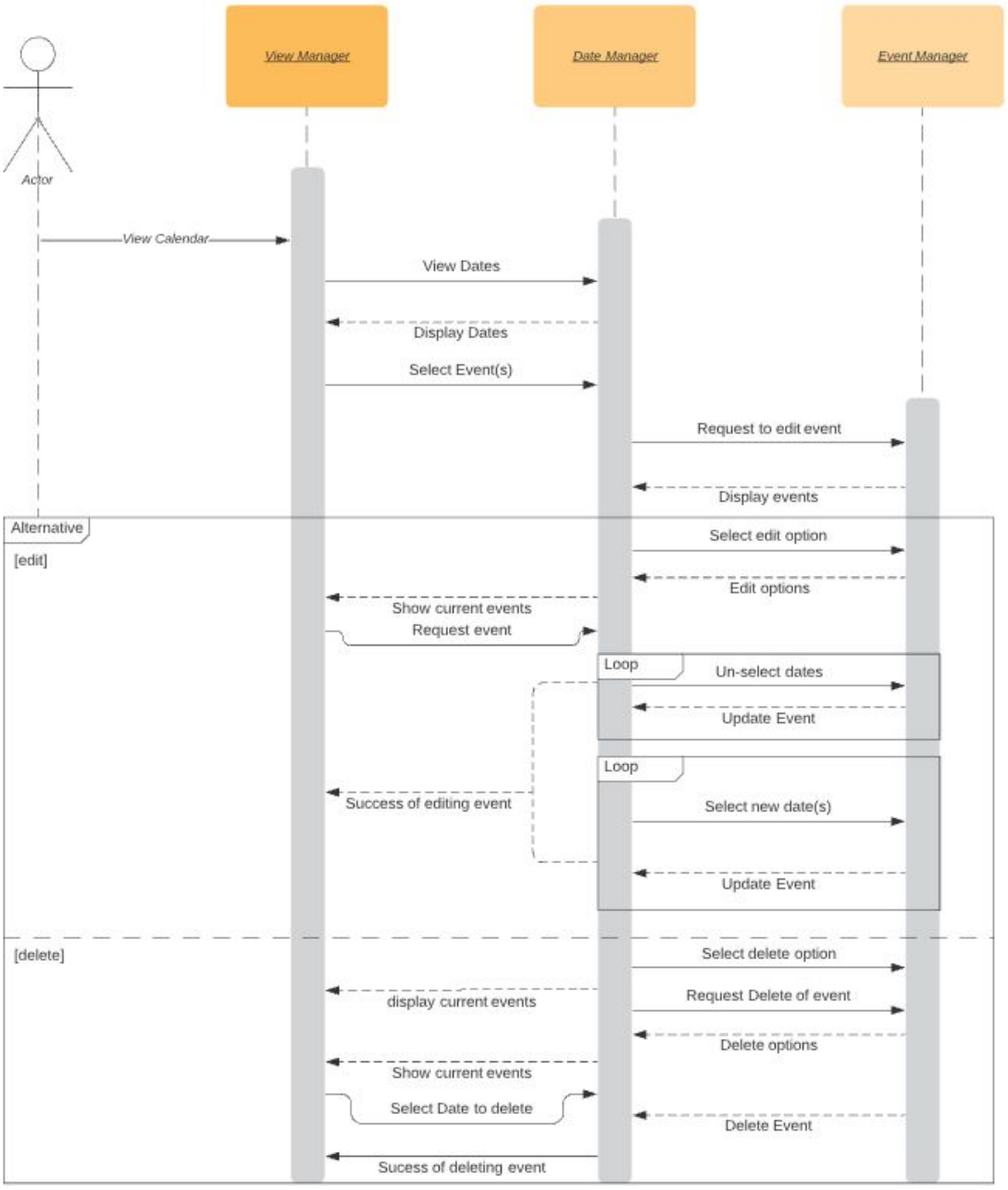
- 4. Add Event


View Manager
Date Manager
Event Manager
Actor
View Calendar
Fiind Date
Display Date Details
Request to add Event
Add Event
Display Added Event Details
Display Event Added

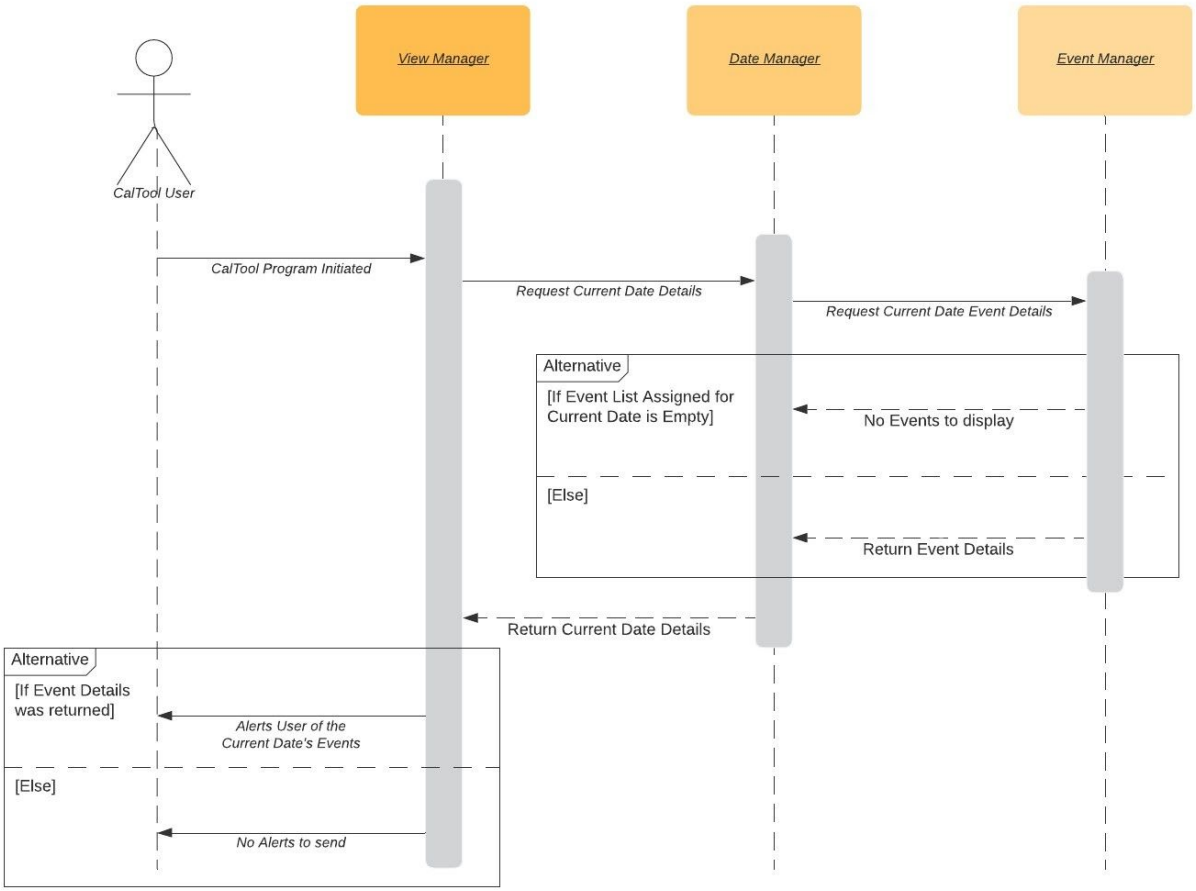
- 5. Add Weekly Events



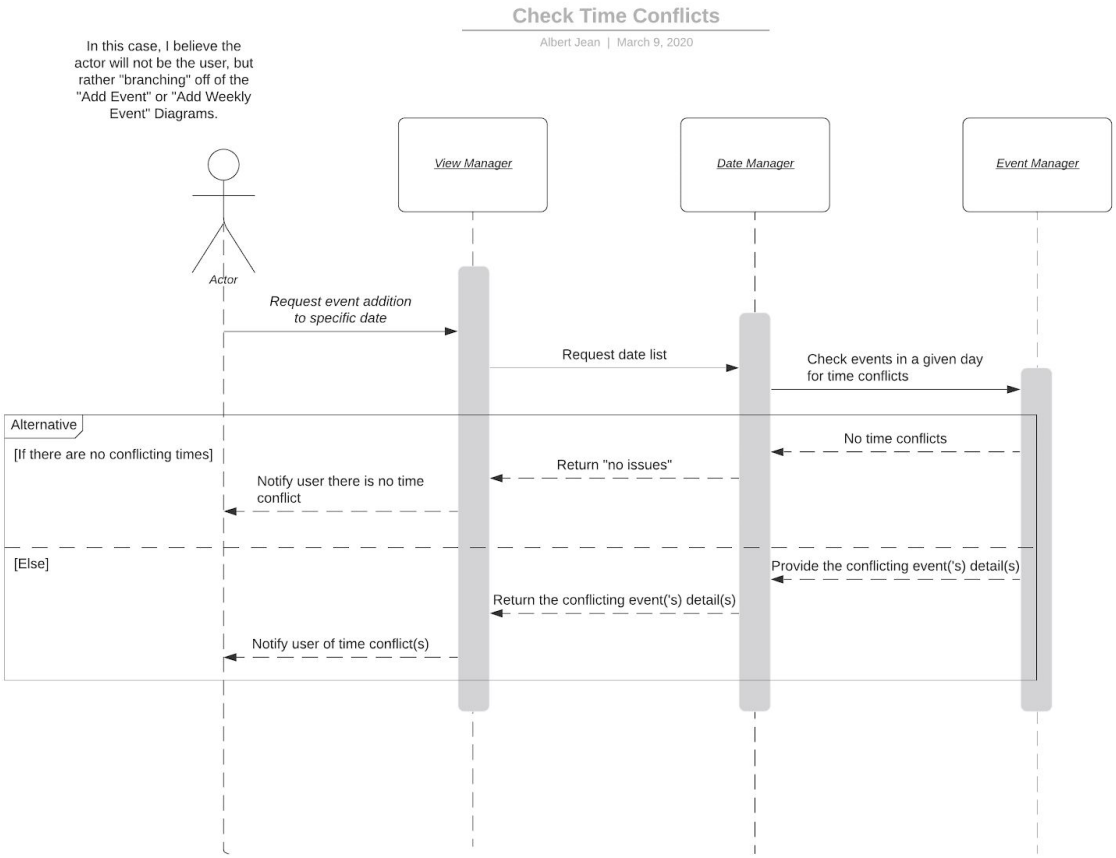
- 6. Edit & Delete Events



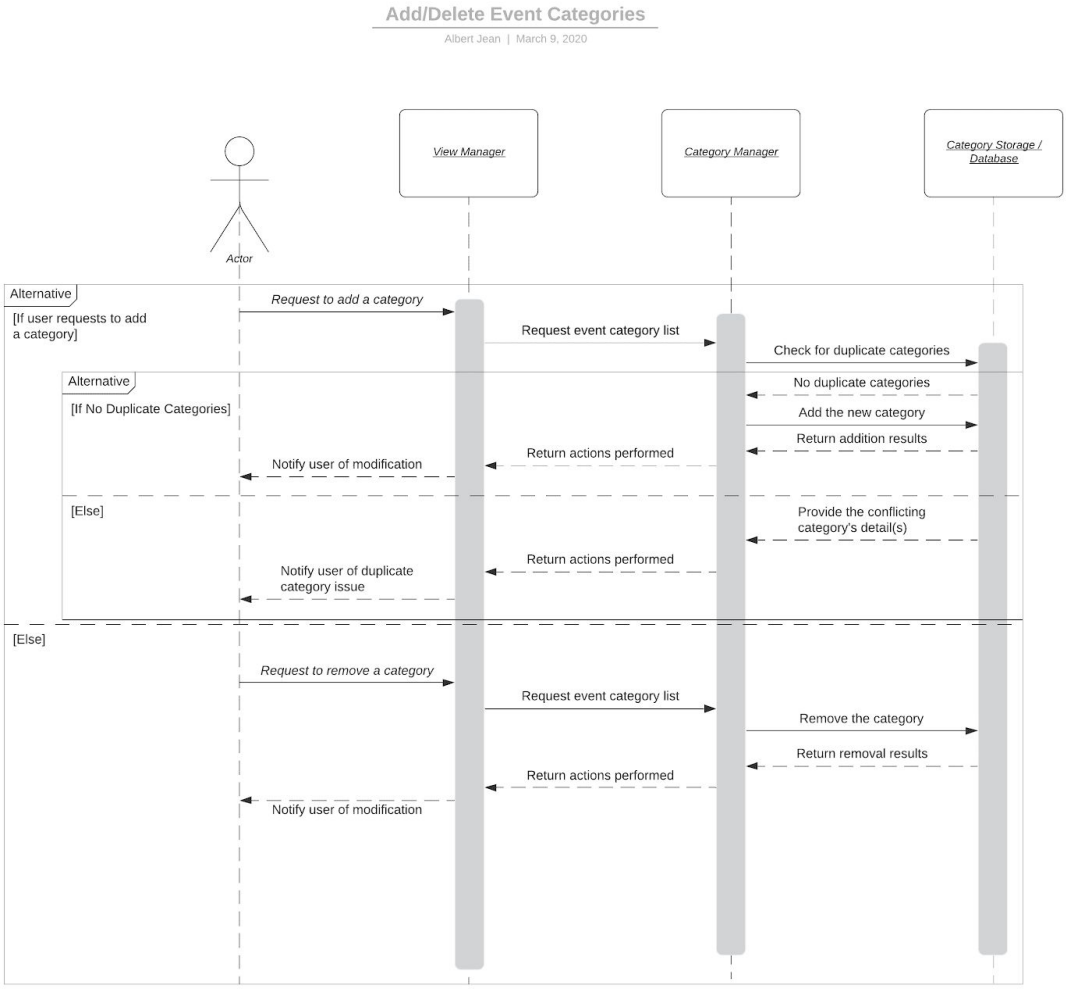
- 7. Event Alert



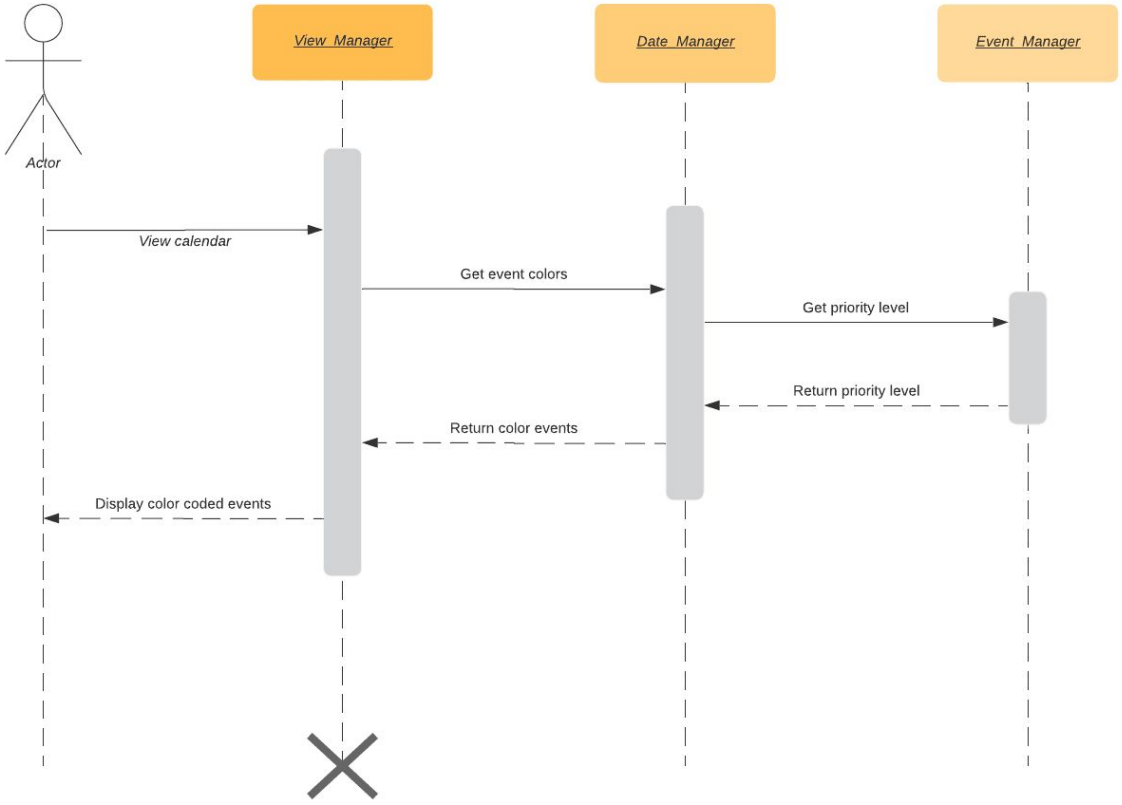
- 8. Check Time Conflicts



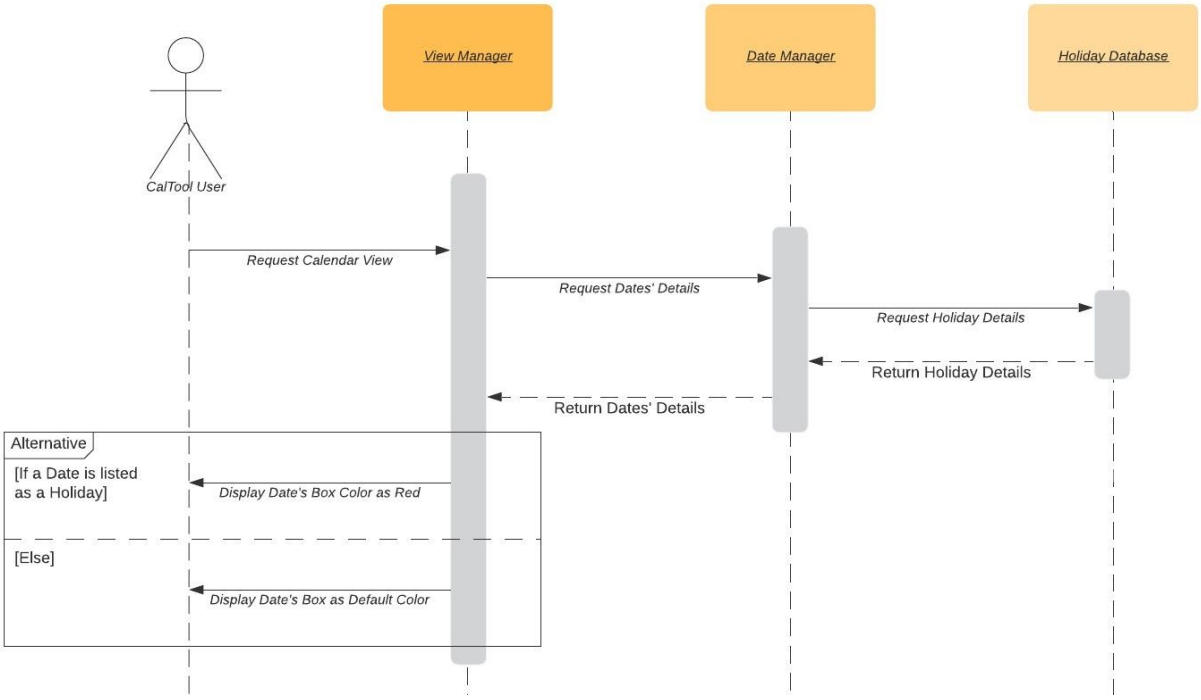
- 9. Add/Delete Event Categories



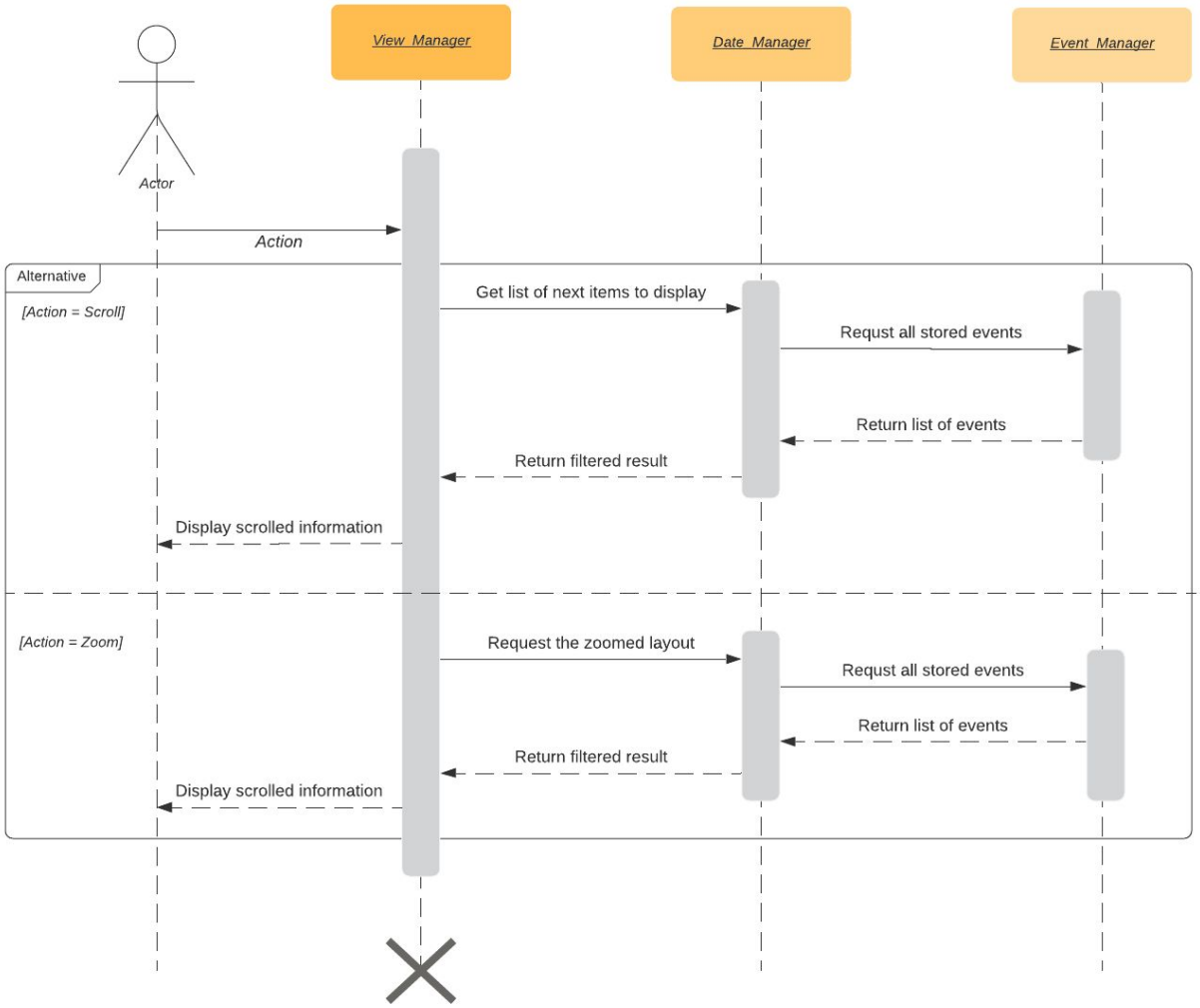
- 10. Color Mark Events



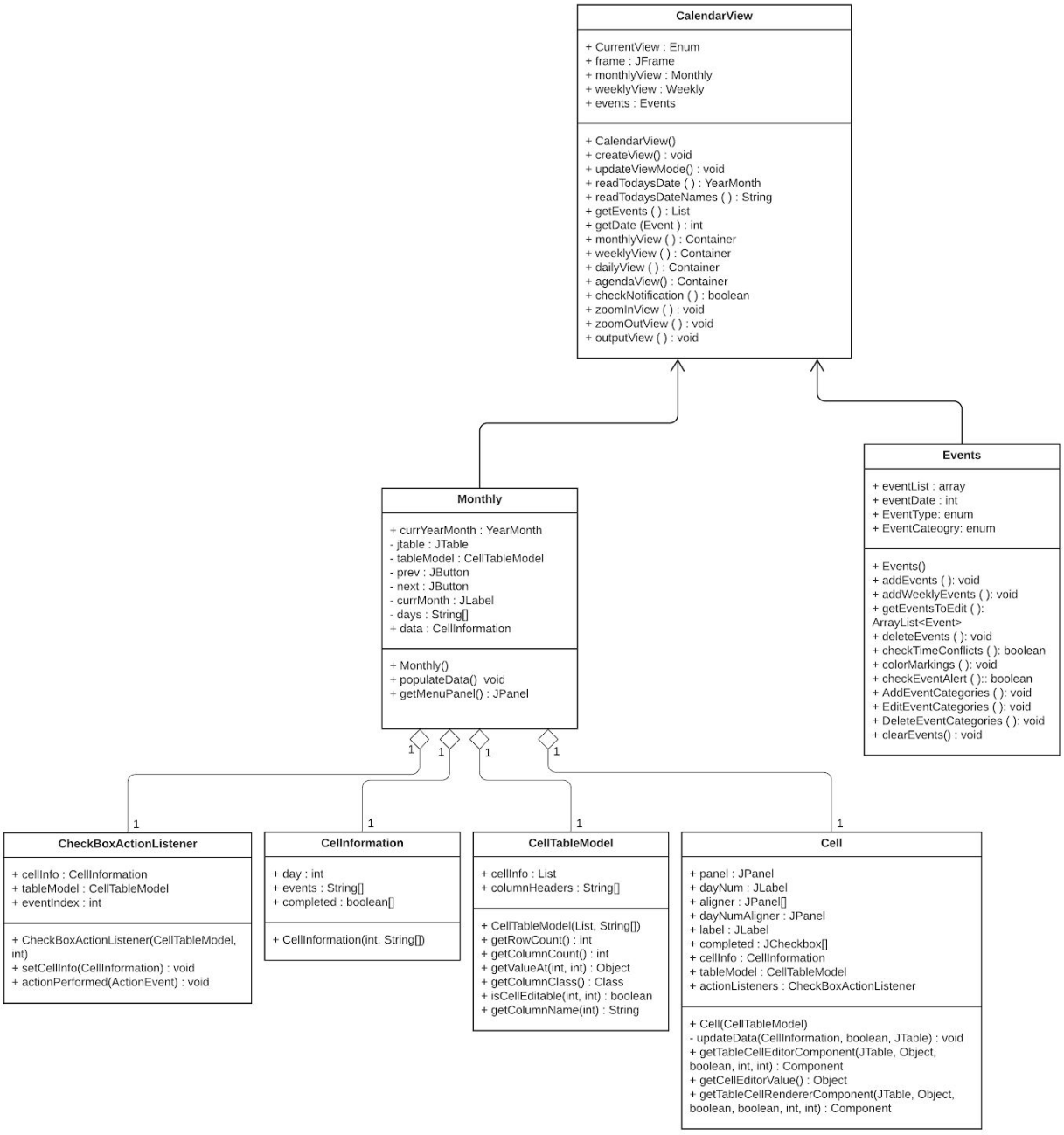
- 11. Special Colors for Holidays



- 12. Zoom and Scroll Support

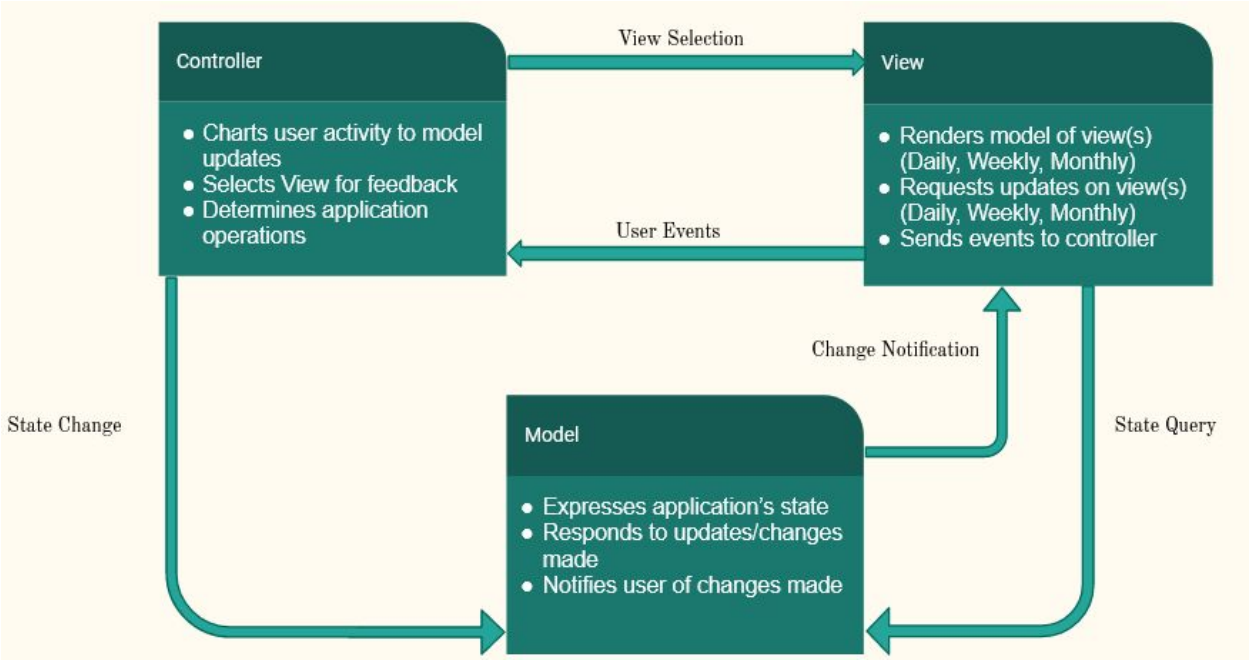


Class diagram:



Architectural design:

Model-View-Controller (MVC) pattern will fit our Calendar-Tool project. This is because we have multiple ways to view data like yearly view, monthly view, and daily view, and MVC supports this feature more than any other architectural design.



Member Tasks for Deliverable 2:

1. Deliverable 2 Report:
    - Christopher O
    - Albert Jean
    - Yousuf Jazzar
    - Kason Alstatt
    - Kent Templin

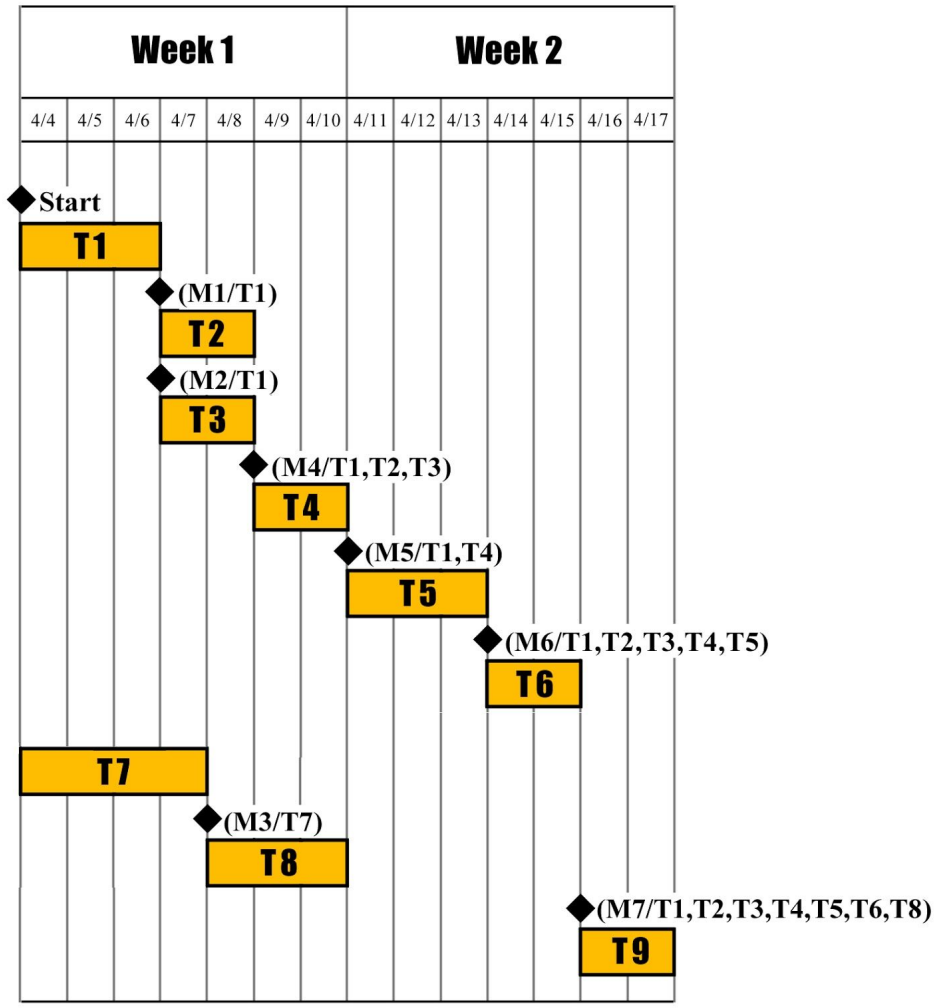
2. Presentation Slide:
    - Kyle Custodio
    - Christopher O
    - Kent Templin

3. Make CalTool:
    - Albert Jean
    - Yousuf Jazzar
    - Kason Alstatt

4. Record Presentation:
    - Piyush Mewada

Project Scheduling:

Task	Effort (person-days)	Duration (days)	Dependencies
T1 Build Monthly.java for CalTool	1	3	
T2 Build Weekly.java for CalTool	1	2	T1 (M1)
T3 Build Days.java for CalTool	1	2	T1 (M2)
T4 Build Main.java for CalTool	1	2	T1, T2, T3 (M4)
T5 Build GUI (View.java)	1	3	T1, T4 (M5)
T6 Build JUnit Class	1	2	T1, T2, T3, T4, T5 (M6)
T7 Write Deliverable 2 Document	3	4	
T8 Write Slide	2	3	T7 (M3)
T9 Record Presentation	1	2	T1, T2, T3, T4, T5, T6, T8 (M7)



This is our hypothetical timeline. It will take about two weeks to finish the project. Longest task is T7, writing a Deliverable 2 document because we thought it requires a lot of time to complete.

Cost, Effort and Pricing Estimation:

- Cost modeling
    + Function Point Algorithmic Estimation Method
        We will use this model to estimate cost, duration, and effort.

- Cost & Effort

    We will use Function Point Algorithmic Estimation Method to estimate effort, time and cost.

    1. Gross Function Point (GFP)

	Function Category	Count	Simple	Average	Complex	Count x Complexity
1	Number of user input (entering events on calendar)	15	3	4	6	45
2	Number of user output (month has 30 days to print)	30	4	5	7	120
3	Number of user queries (month needs 31 queries)	31	3	4	6	93
4	Number of data files and rational tables (a year has 12	12	7	10	15	84
5	Number of external interfaces	4	5	7	10	20
					GFP	362

    Therefore, **GFP = 362**

    2. Processing Complexity (PC)

We will answer 14 questions to determine processing complexity.

(1) Does the system require reliable backup and recovery?
4

(2) Are data communications required?
5

(3) Are there distributed processing functions?
3

(4) Is performance critical?
4

(5) Will the system run in an existing, heavily utilized operational environment?
3

(6) Does the system require online data entry?
4

(7) Does the online data entry require the input transaction to be built over multiple screens or operations?
2

(8) Are the master files updated online?
1

(9) Are the inputs, outputs, files, or inquiries complex?
2

(10) Is the internal processing complex?
1

(11) Is the code designed to be reusable?
1

(12) Are conversion and installation included in the design?
2

(13) Is the system designed for multiple installations in different organizations?
2

(14) Is the application designed to facilitate change and ease of use by the user?

(4+5+3+4+3+4+2+1+2+1+1+2+2+3) / 14 = 2.643
Average PC = 2.643.

3. Processing Complexity Adjustment (PCA)

PCA = 0.65 + 0.01(Total PC)
= 0.65 + 0.01(4+5+3+4+3+4+2+1+2+1+1+2+2+3)
= 0.65 + 0.01(37)
= 0.65 + 0.37
= **1.02**

4. Function Point

FP = GFP * PCA
= 362 * 1.02
= **369.24 FP**

5. Estimated Effort

E = FP / productivity
(assume productivity as 40 function points per person-week)
= 369.24 / 40
= **9.231 person-weeks**

6. Duration

D = E / teamsize
= 9.231 / 7
= 1.8462 weeks
= **2 weeks**

7. Cost

C = D * (software engineer average salary) * (team members)
= (2 weeks) * ($1770 per week) * (7)
= **$24,780**

Therefore,

Effort = 9.231 person-weeks
Duration = 2 weeks
Cost = $24,780

- Pricing Estimation

Most calendar apps in the market cost about $0.00. It is probably a good idea for us to sell our CalTool product for $0.00 too so that the consumer will be able to download it easily and instantly. To gain revenue, we will put pop-up ads in the app. Pop-up ads will appear at the very bottom, so that it is not distracting. If 1,000,000 people downloaded our app and if we would get $0.01 from each popup-ad, then $10,000 will be our revenue. If they view 3 ads, then our revenue will be $30,000.

Therefore,
Price = $0.00

- Estimated cost of hardware products

+ Our company (team) may need 3 computers to code CalTool. If one desktop computer costs $600, then 3 x $600 = $1,800.
+ Server is not needed for this product because time and data will run in the customer's smartphone / computer.

Therefore,
hardware cost = **$1,800**

- Estimated cost of software products

We will only use Java for this project. Java has a GNU GPL (General Public License), so it is free to use and distribute.

Therefore
software cost = **$0.00**

- Estimated cost of personnel

+ 7 people needed.
+ Training will cost $0.00; no need for training because our members are very familiar with Java.
+ If a company is going to pay each employee about the average of a software developer, then the cost will be the same as what we explained before, $24,780.

(time) * (software engineer average salary) * (team members) + (training cost)
= (2 weeks) * ($1770 per week) * (7) + 0
= **$24,780 for 2 weeks**

Test Plan for Our Software:

- Unit Testing Case:

  We used the Java programming language, so we will be using JUnit for our unit testing.

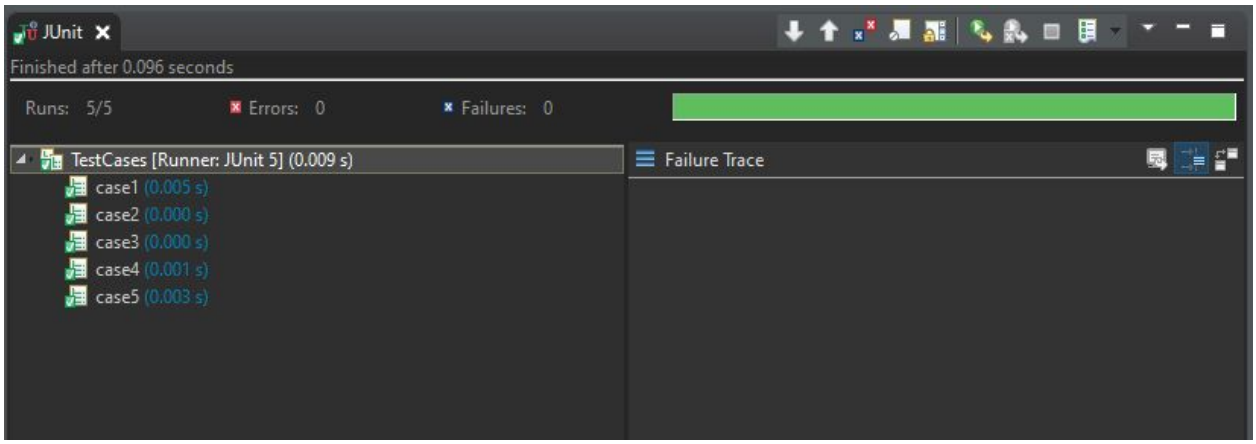
  We chose to test the Events class within our project, which is responsible for handling the adding of events, removing of events, editing events, checking time conflicts, and more.

  To test the project, these test cases were created:
  1. Testing if events add correctly (instantiation)
  2. Testing if only valid events are added correctly (and an exception is thrown otherwise).
  3. Testing if event attributes are added correctly
  4. Checking if time conflicts are checked correctly
  5. Testing to see if events can be cleared without any errors.

+ Code for one unit testing with Junit.

```java
public class TestCases {

  // This case will test to see if events can be added successfully under correct conditions
  @Test
  public void case1() throws ClassExceptionError {
    Events events = new Events();

    events.addEvent(new Event("Event Name", 10, Color.BLACK));

    assertEquals("Checking if event added successfully", 1, events.getEventsToEdit().size());
  }


  // This case will test to see if only valid Event instances are added to the list of stored events
  @Test (expected = ClassExceptionError.class)
  public void case2 () throws ClassExceptionError {
    Events events = new Events();

    events.addEvent(new ArrayList<Integer>());

    assertEquals("Checking if event added successfully", 0, events.getEventsToEdit().size());
  }


  // This function will test if values remain the same when they are added to the list of stored events
  @Test
  public void case3() throws ClassExceptionError {
    Events events = new Events();

    events.addEvent(new Event("Expected Event Name", 100, Color.CYAN));

    assertEquals("Checking if eventName added correctly.", "Expected Event Name", events.getEventsToEdit().get(0).name);
    assertEquals("Checking if eventDate added correctly.", 100, events.getEventsToEdit().get(0).eventDate);
    assertEquals("Checking if eventColor added correctly.", Color.CYAN, events.getEventsToEdit().get(0).color);
  }


  // This case will test for time conflict
  @Test
  public void case4() throws ClassExceptionError {
    Events events = new Events();

    events.addEvent(new Event("Conflict #1", 100, Color.RED));
    events.addEvent(new Event("Conflict #2", 100, Color.GREEN));

    assertEquals("Checking for correct time conflict return value (true).", true, events.checkTimeConflicts());
  }


  // This case will test to see if the eventList can be cleared correctly using the Events.clearEvents() function
  @Test
  public void case5() throws ClassExceptionError {
    Events events = new Events();

    // Add all events
    for (int i = 1; i <= 100; i ++) {
      events.addEvent(new Event("Event #" + i, i%30, Color.RED));
    }

    assertEquals("If all events were added correctly", 100, events.getEventsToEdit().size());

    // Clear the stored events
    events.clearEvents();

    assertEquals("If all events were cleared correctly", 0, events.getEventsToEdit().size());
  }


}
```

+ Result of one unit testing with Junit.



Test case results:

1. Testing if events add correctly (instantiation)
    a. Expected Result: 1
    b. Result: 1
2. Testing if only valid events are added correctly (and an exception is thrown otherwise).
    a. Expected Result: 0
    b. Result: 0
3. Testing if event attributes are added correctly
    a. Expected Result: "Expected Event Name"
    b. Result: "Expected Event Name"
    c. Expected Result: 100
    d. Result: 100
    e. Expected Result: Color.CYAN
    f. Result: Color.CYAN
4. Checking if time conflicts are checked correctly
    a. Expected Result: 1
    b. Result: 1
5. Testing to see if events can be cleared without any errors.
    a. Expected Result: 100
    b. Result: 100
    c. Expected Result: 0
    d. Result: 0

Comparison of our work with similar designs:

Considering the nature of the software on which we have been working, there exist numerous amounts of various softwares that are similar to ours. Since there are a gargantuan number of calendar softwares that we could use in our comparison, choosing a single software would be better for comparing the similar designs. That software that was chosen was Microsoft's Outlook Calendar, which shares similarities but also differences between the two works.

Starting off, the Outlook Calendar has many similarities with CalTool, as they are both calendar softwares. The two designs both have the option to create, color, and delete events [1]. They both can switch between monthly, weekly, and daily views, as well as showing the user that a certain day is a holiday [2].

Conversely, the Outlook Calendar has several differences with CalTool. For starters, CalTool was mainly designed for the software to focus on helping the user themselves. Outlook Calendar, on the other hand, has functions that give the user the ability to manage and organize meetings, create appointments, and view different group schedules [1]. The software gives users a way to plan activities with other users.

Since CalTool is a calendar software, it has several similar designs that could be used for comparison. After comparing CalTool to a well known calendar software, Outlook Calendar, it is obvious that our software is fairly new. Overall, although CalTool is a well developed calendar software, it lacks some functionalities compared to Outlook Calendar, which was developed by a high end company, Microsoft.

Conclusion:

As a team we were able to get together and plan out and build a software that is aligned with software engineering principles along with our practice of teamwork. With the help of the software engineering design process, we came out with a product that is effective in solving our problem and is reliable to use.

Class diagram deviated from what we have originally planned. Our original class diagram only had three classes (calanderView, dates, and events); it was simple. However we realized that CalTool would be much more complicated than we thought because GUI and Junit were needed; they would make CalTool more stable, presentable and user friendly. Therefore, at the end, CalTool got more elaborate and ended up having seven classes (calanderView, monthly, events, CheckBox, CellInfo, CellTableModel, and Cell).

Again, our motivation for CalTool was we needed assistance in planning/scheduling for our busy semester. A student's daily activities can be overburdening so a calendar software would help us manage our time and our future user's time. CalTool's single user ease of use will be helpful to our fellow student's plans for their semesters.

References:

Outlook

[1] "Introduction to the Outlook Calendar," *Outlook*. [Online]. Available: https://support.office.com/en-us/article/introduction-to-the-outlook-calendar-d94c5203-77c7-48ec-90a5-2e2bc10bd6f8. [Accessed: 10-Apr-2020].

[2] "Change how you view your Outlook calendar," *Outlook*. [Online]. Available: https://support.office.com/en-us/article/change-how-you-view-your-outlook-calendar-a4e0dfd2-89a1-4770-9197-a3e786f4cd8f. [Accessed: 10-Apr-2020].