CalTool

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

Deliverable 1 Presentation Agenda

1. Team Introduction
2. Initial Project “CalTool” Ideas
3. Software Process Model
4. Requirements
5. Diagrams
    a. Use Case Diagram
    b. Sequence Diagrams
    c. Class Diagrams
6. Architectural Design

Deliverable 2 Presentation Agenda

1. Project Scheduling
    a. Cost, Effort, and Price Estimation
    b. Estimated Hardware Costs
    c. Estimated Software Costs
    d. Estimated Personnel Costs
2. Test Plan
3. (CalTool Presentation (?))
4. Comparison of Our Work with Similar Designs
5. Conclusion
6. References

Deliverable 1

Team

Project name: CalTool

Team Members:

Yousuf Jazzar

Christopher Oliva

Kyle Custodio

Kent Templin

Piyush Mewada

Kason Alstatt

Albert Jean

Software Process Model

Incremental Process Model

- Saves money and time for software engineering
- Engineers can add features while programming
- Used in software engineering often, fits our project needs

Functional Requirements

The program needs to be able to:

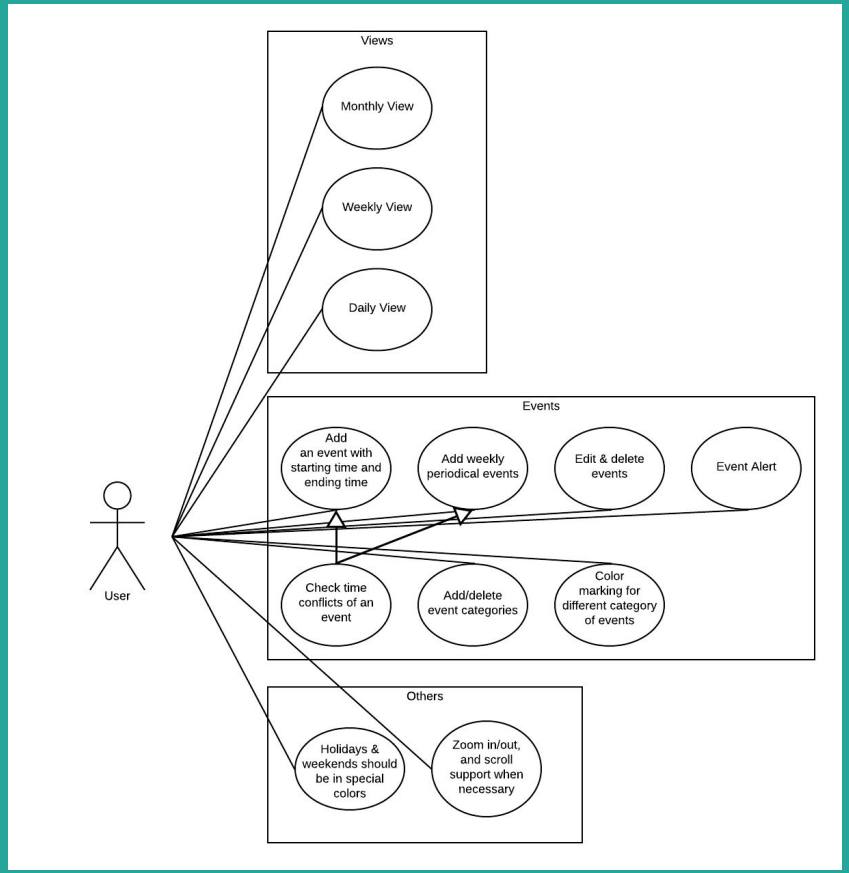
- Display calendar with dates, current dates, and user generated events.
- Display calendar in Monthly View, Weekly View, or Daily View
- Read multiple event data from the user
- Save multiple event data
- Send event notifications

Non-Functional Requirements

- Run in JAVA, C, C++, or Python
- Total size less than 200mb
- Open within 1 second
- Store 100 years worth of dates
- Save at least 100 user generated events

Use Case Diagram

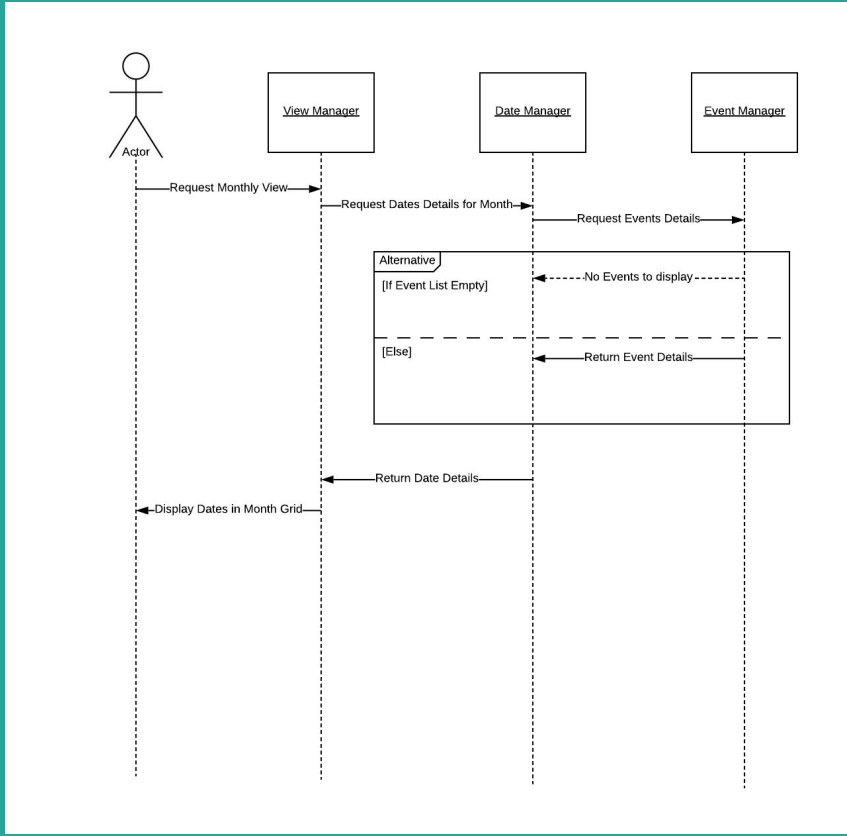
All of the use cases available to our user



Monthly View

User will be able to see the days in a month

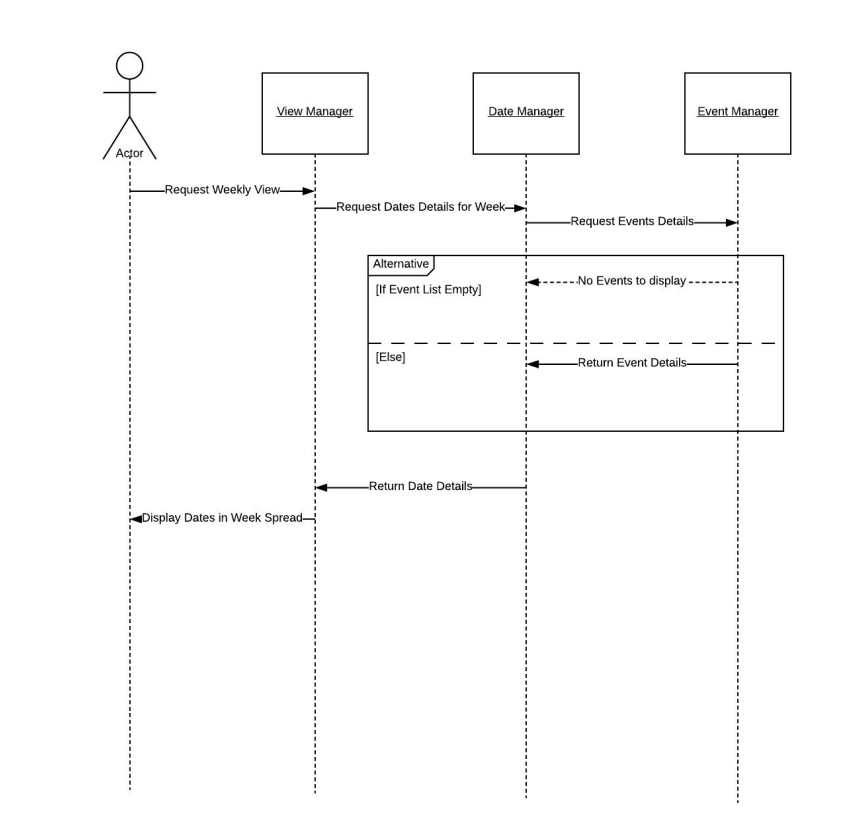
Events on each day listed



Weekly View

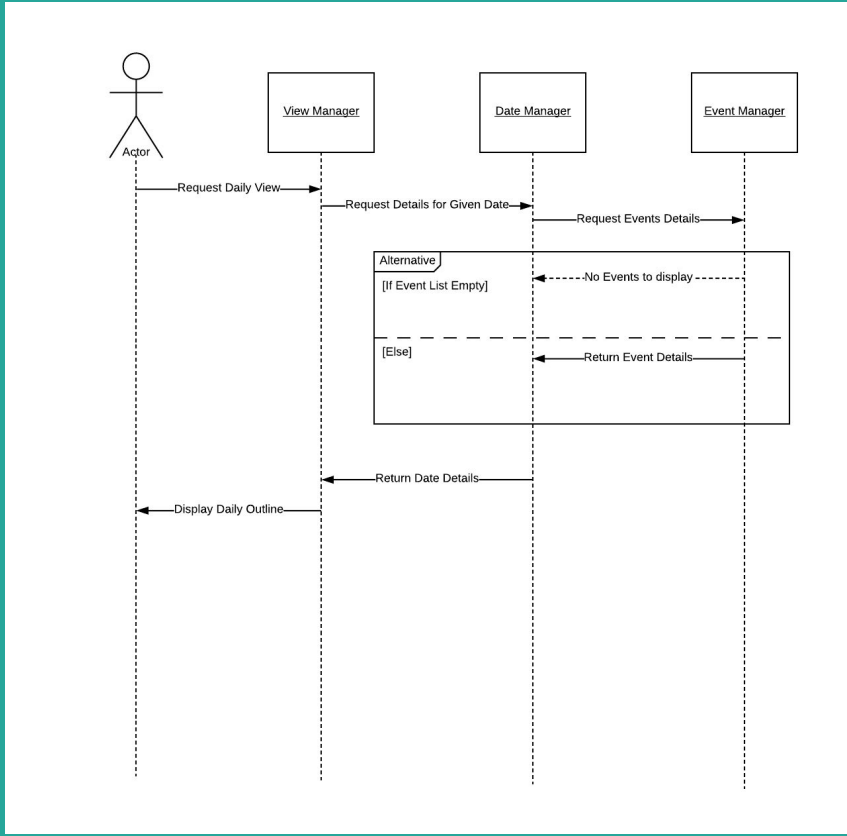
User will be able to see the days of the current week

Events on each day listed



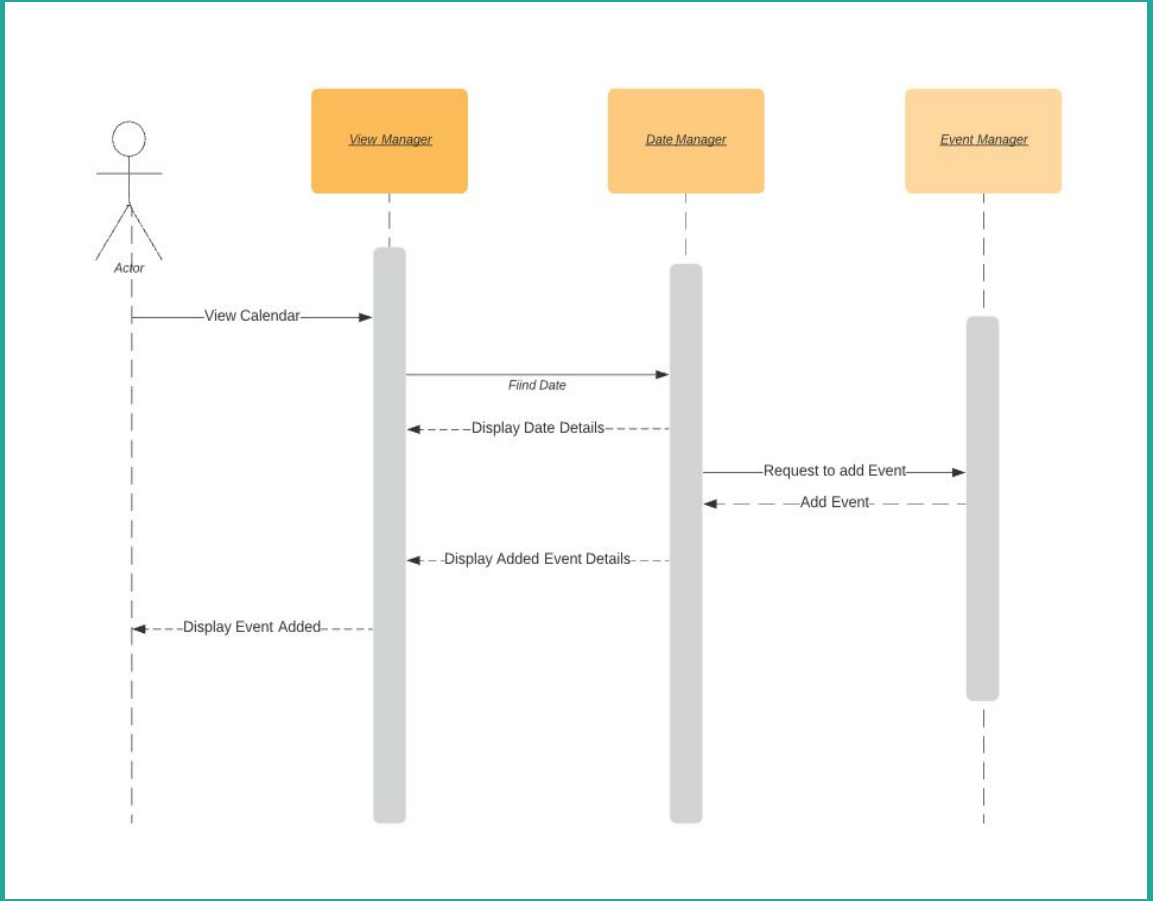
Daily View

User will be able to see the events in a given day



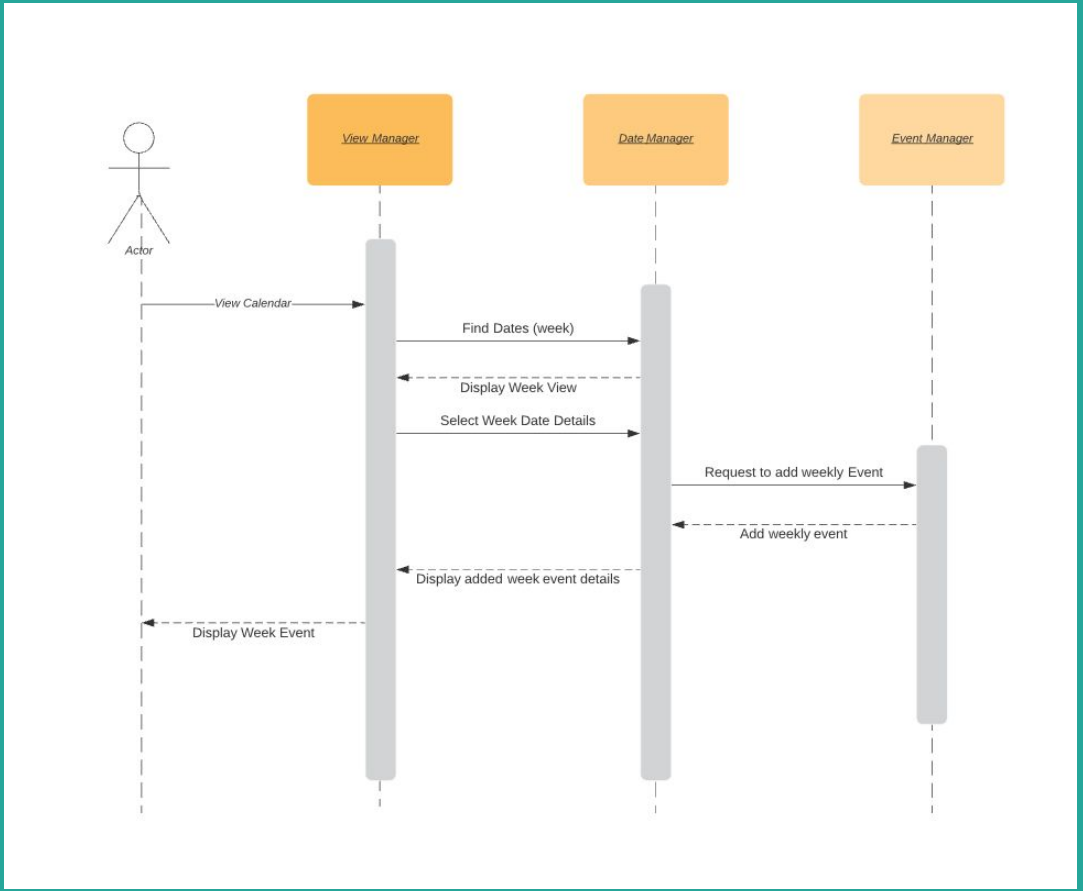
Add Event

User will be able to add an event in CalTool



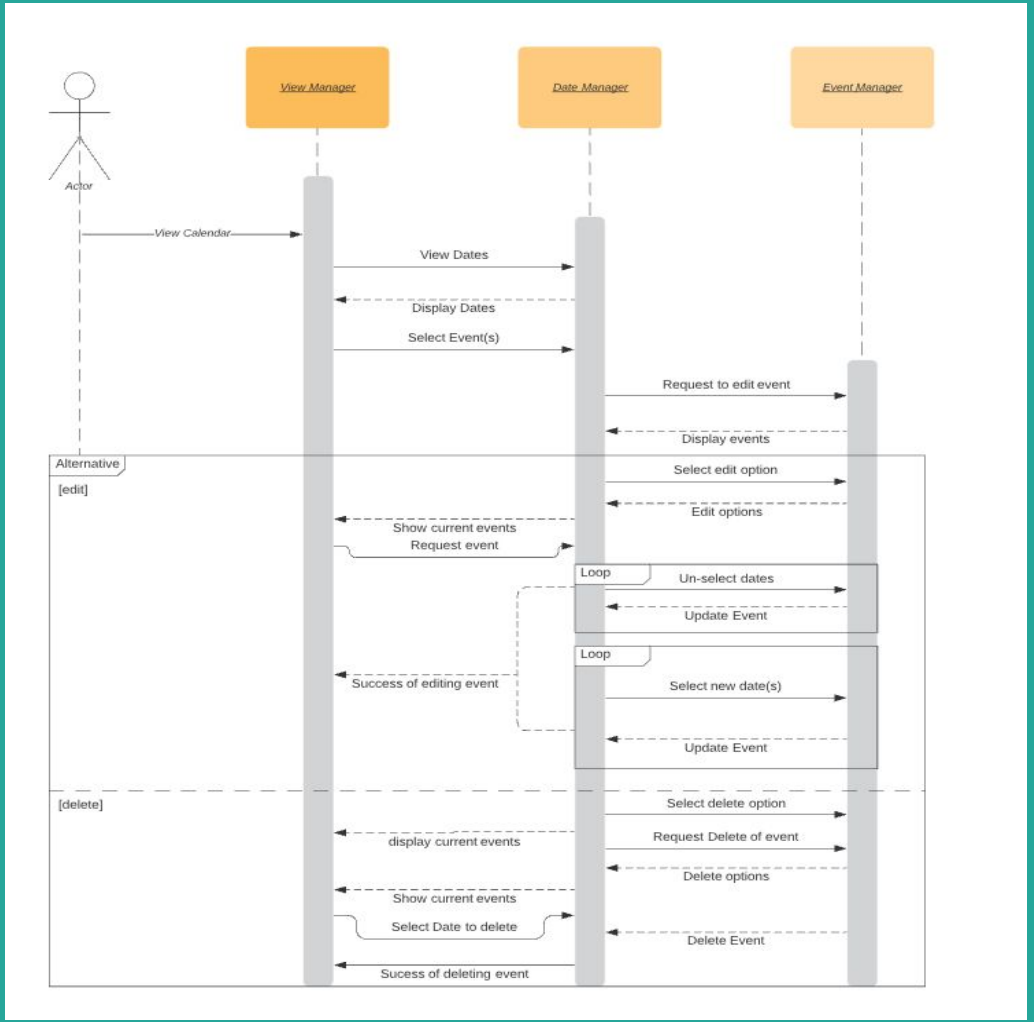
Add Weekly Event

The user will be able to add a weekly event in CalTool



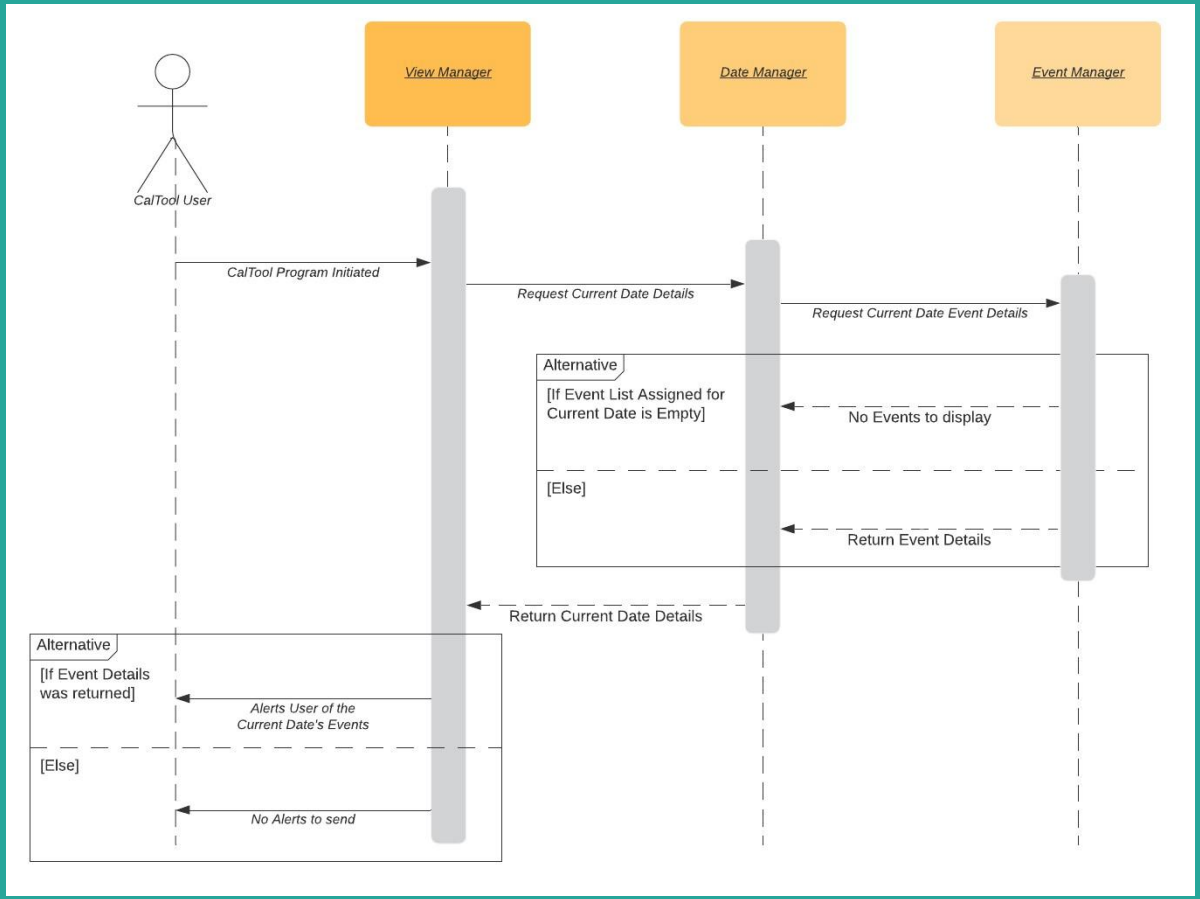
Edit and Delete Event(s)

The user will have the ability to view dates and select certain events. Within this view they will have the option to either delete and event or edit an event. Editing an event includes: altering days, changing length of event, and moving events.



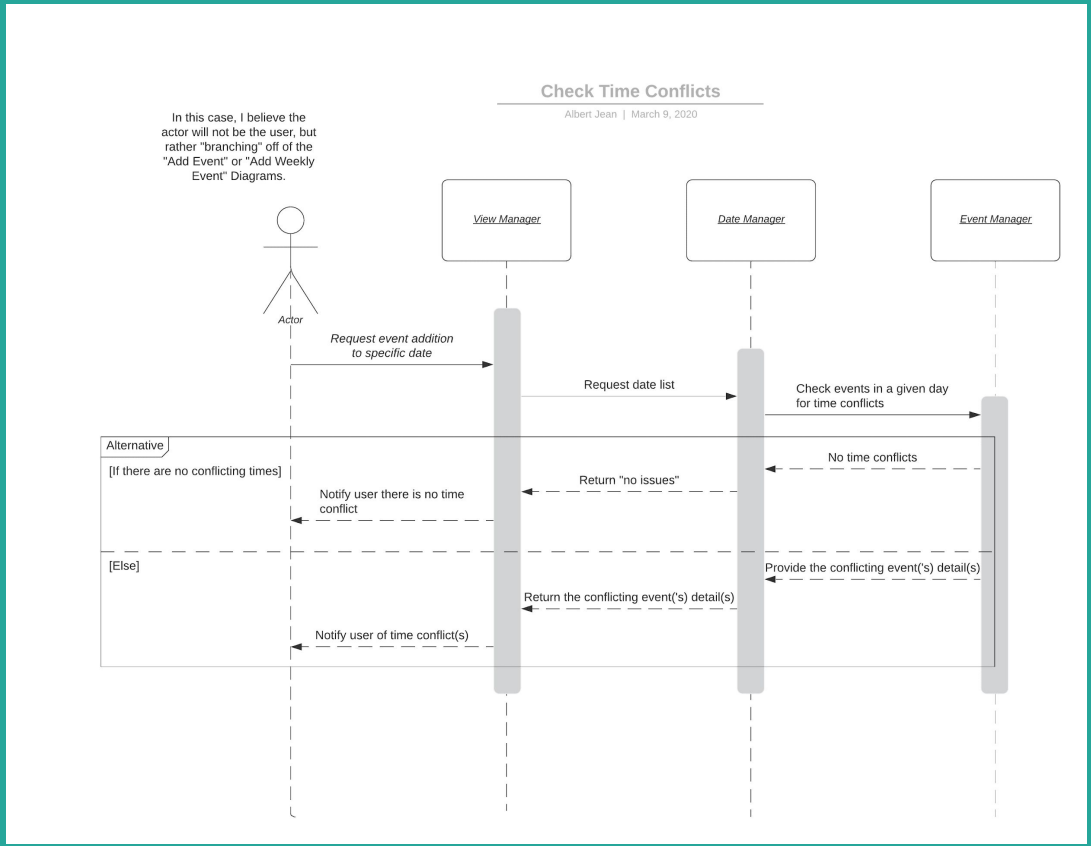
Event Alert

CalTool will alert the user with a notification of an event with its included details of the event



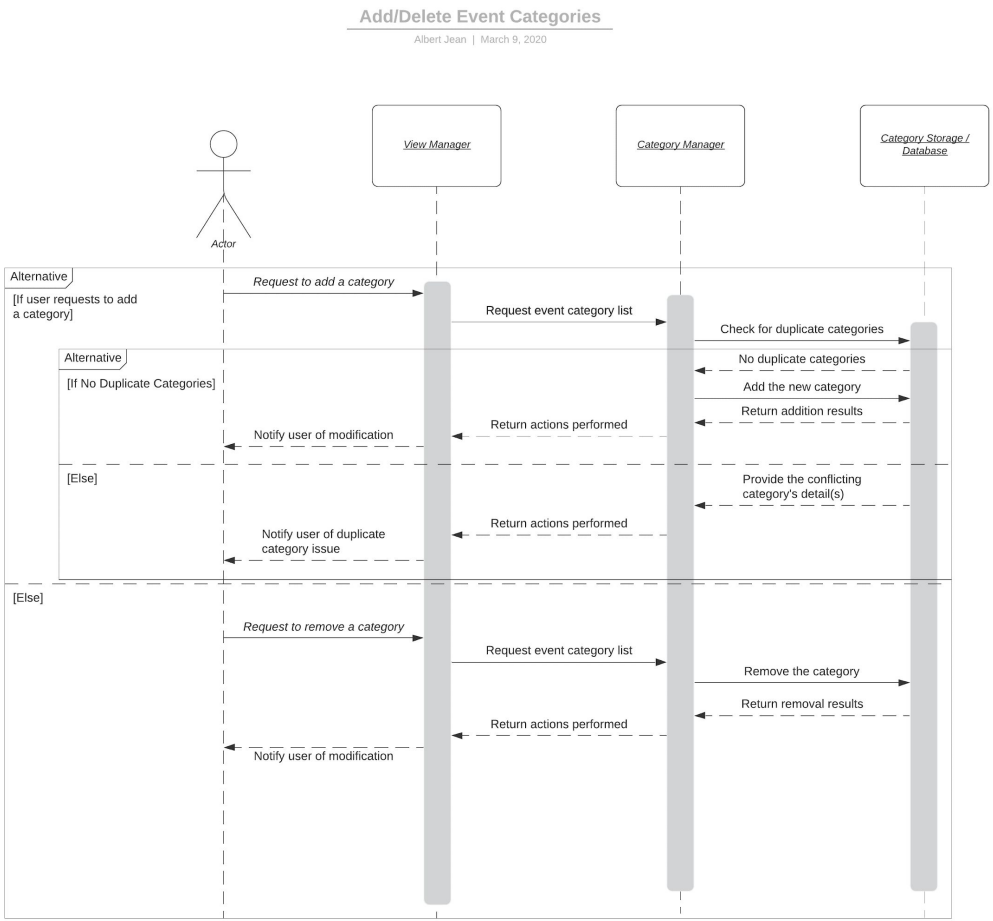
Check Time Conflict

CalTool will also check added events from the user to detect any overlap or conflicts with the user's added events.



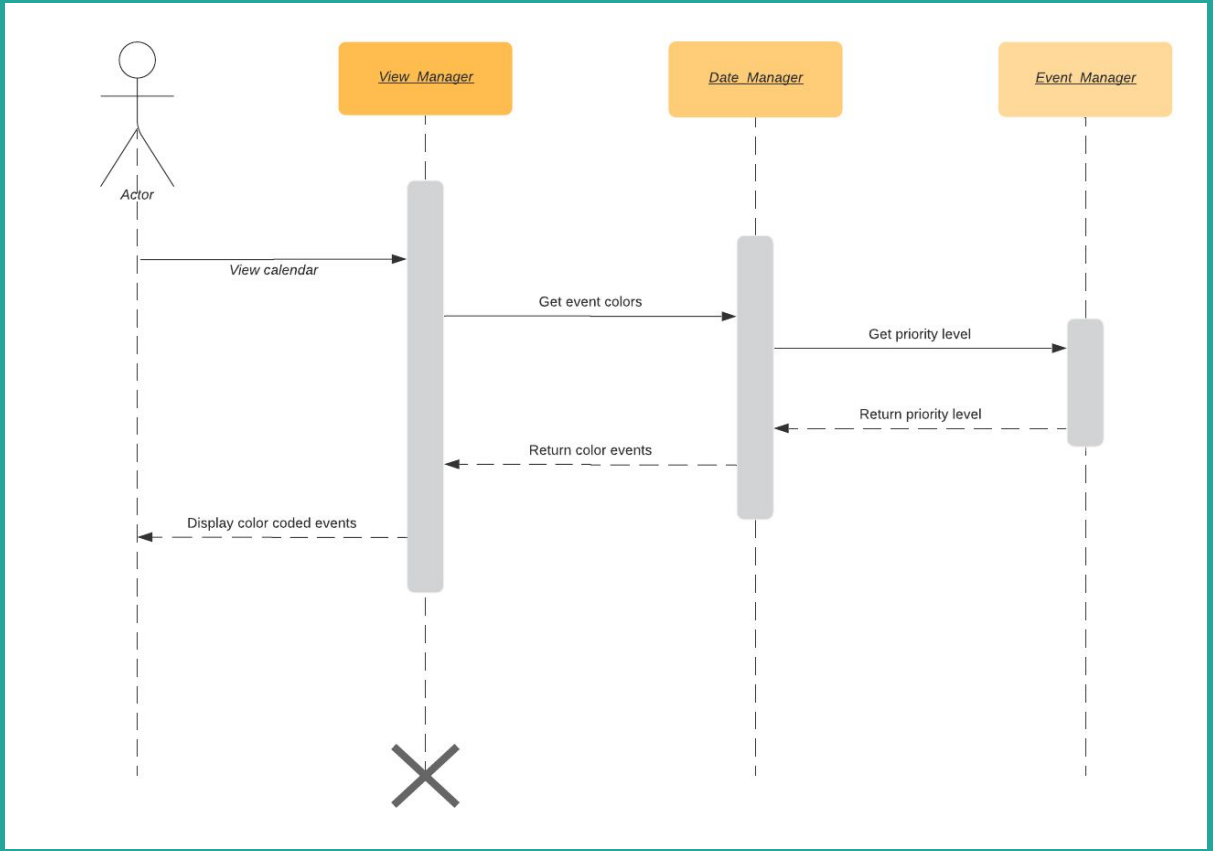
Add/Delete Event Categories

User will be able to add categories after checking if the category has been added previously. If a duplicate event is created the user is notified of a duplicate category. User also has the option to remove/delete the category.



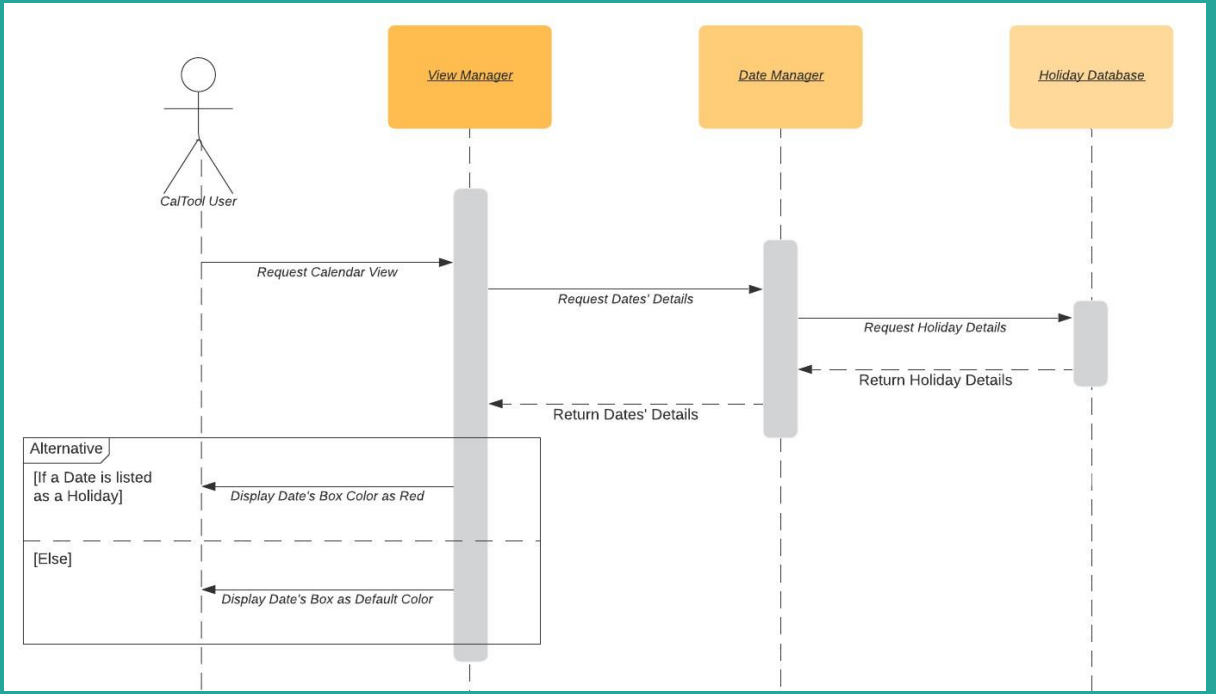
Color Mark Events

User will be able to mark events by their priority



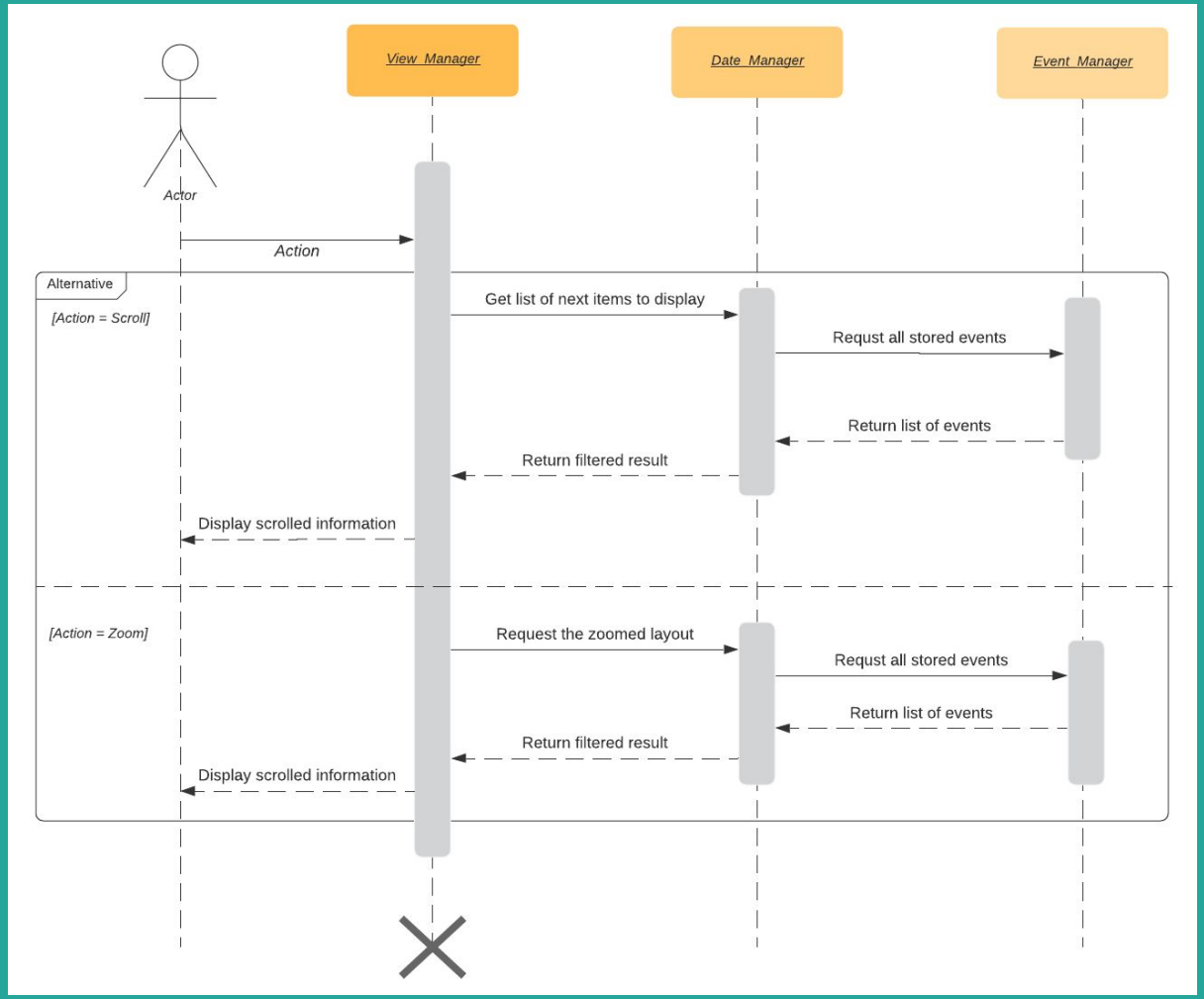
Special Holiday Mark

Holidays will have different event colors along with its listed details



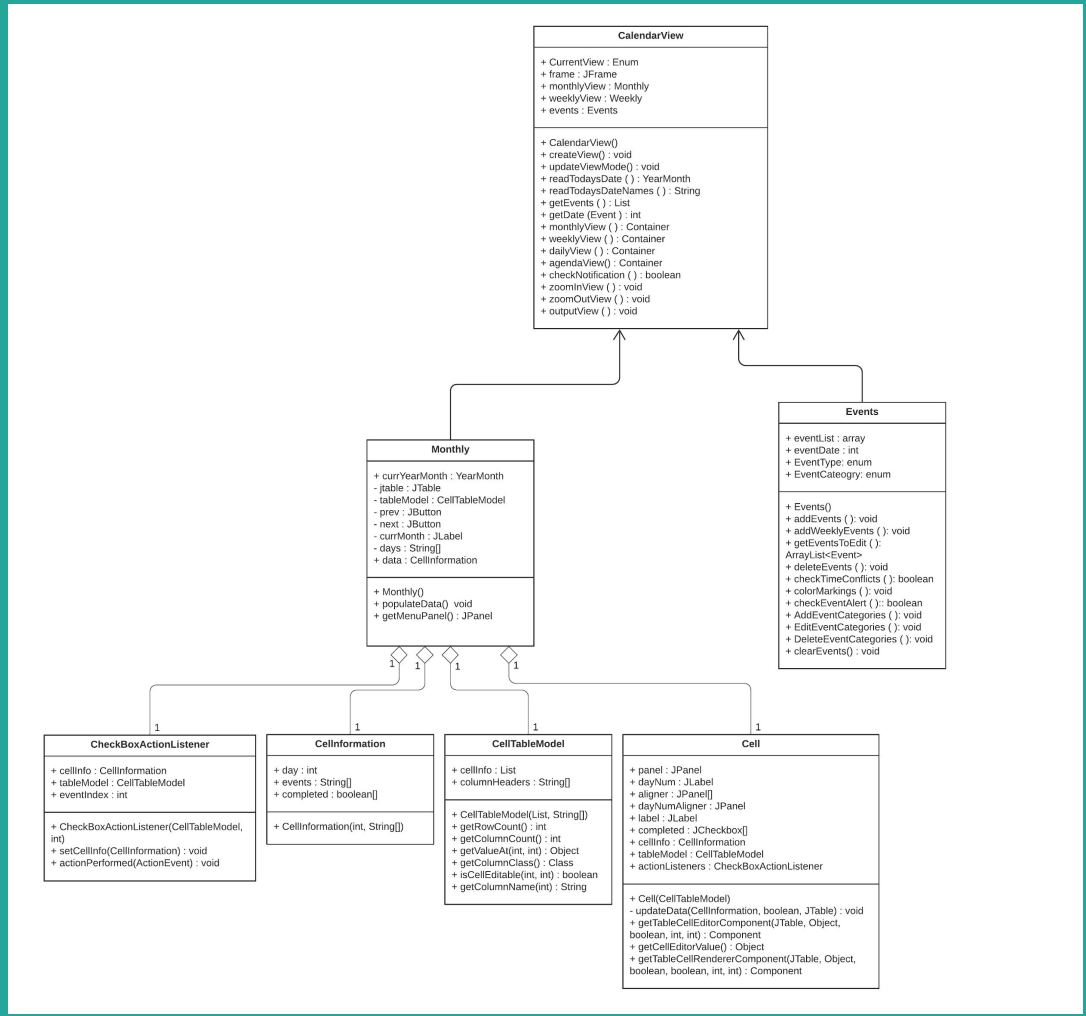
Zoom and Scroll

CalTool will allow the user to scroll past their events and calendar. Zoom support will also be included.



Class Diagram

//anything info we need to include

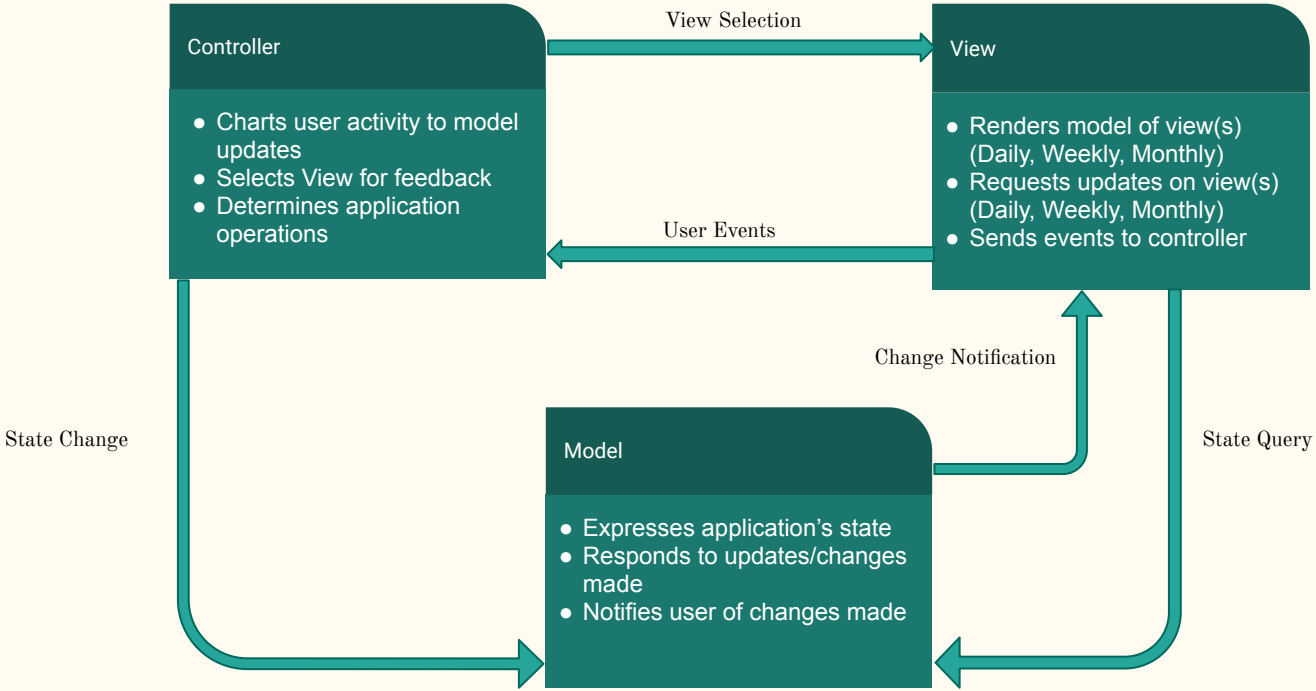


Architectural Design: Model View Controller

Advantages for using the MVC Architectural Design :

- Data can be manipulated in various ways to the view which interacts with the data aspect
- Support the presentation of our CalTool in different ways for various days/weeks/months with multiple events
- MVC is heavily used future exchanges and presentation of data are unknown
- Future development and alterations are easier
- Supports multiple views of the models for CalTool
- MVC facilitates good cohesion with alterations on the controller
- Separation of managements allows more flexibility with the MVC

Model View Controller



Deliverable 2

Project Scheduling

Task	Effort (Person Days)	Duration (Days)	Dependencies
T1 Build Monthly.java	1	3	
T2 Build Weekly.java	1	2	T1 (M1)
T3 Build Daily.java	1	2	T1 (M2)
T4 Build Main.java	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 (M6)
T7 Write Deliverable 2 Document	3	4	
T8 Create Slide Presentation	2	3	T7 (M3)
T9 Record Presentation	1	2	T1, T2, T3, T4, T5, T6, T8 (M7)

Project Scheduling

Cost, Effort, and Price Estimation

Cost - $24,780

Effort - 13.4 person-weeks

Price Estimation - $0.00

- To compete with other calendar apps CalTool will be free
- Source of revenue, advertisements
  - If each ad gained $0.01, and 1,000,000 people saw a single ad, we would gain $10,000
  - If each of the 1,000,000 people saw 3 ads, we would gain $30,000

Estimated Costs

Hardware Products:

- 3 Developers and each need a computer
- Cost of Computer - $600
- Server not needed, store data on client
- **Cost of Hardware** - $1,800

Software:

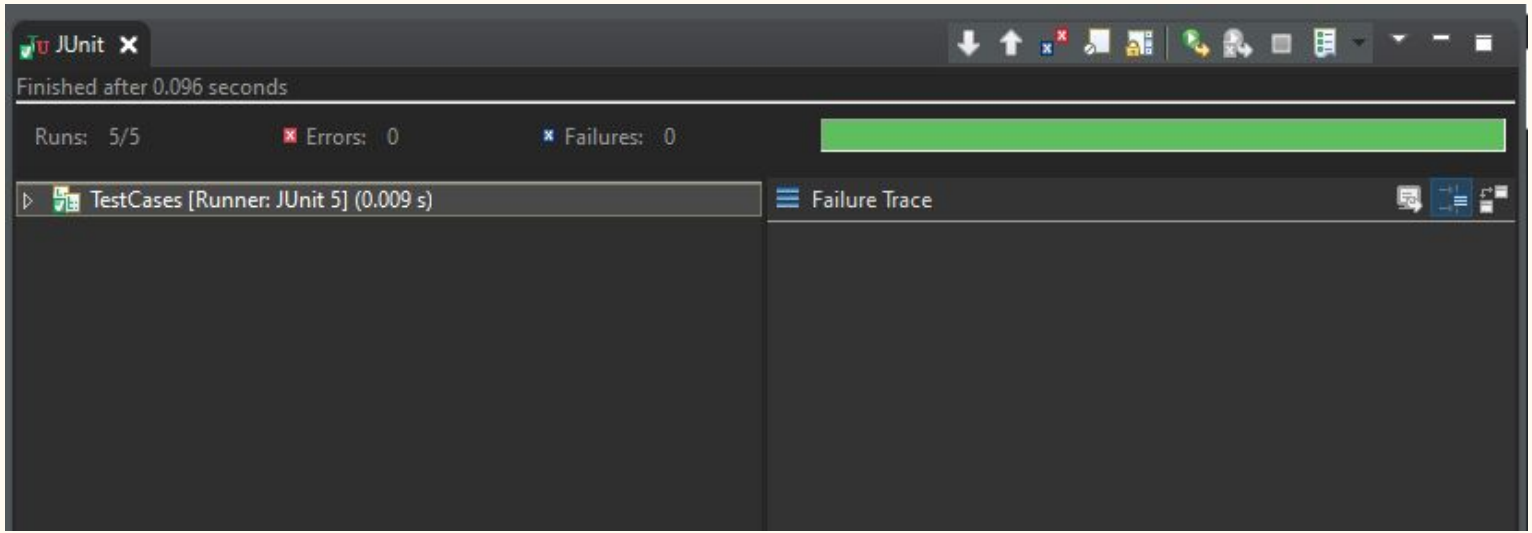
- Java has a GNU GPL licence, thus free to use and distribute
- **Cost of Software** - $0.00

Personnel:

- 7 people, no training needed, 2 weeks
- Average Software Engineer Cost - $1,700/week
- **Cost of Personnel** - $24,780

Test Plan

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



Test Plan

- We chose to test the Events class within our project.
- These are the test cases generated:
  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.

1. Testing if events add correctly (instantiation)

```java
// 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());
}
```

2. Testing if only valid events are added correctly

```java
// 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());
}
```

3. Testing if event attributes are added correctly

```java
// 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);
}
```

4. Checking if time conflicts are checked correctly

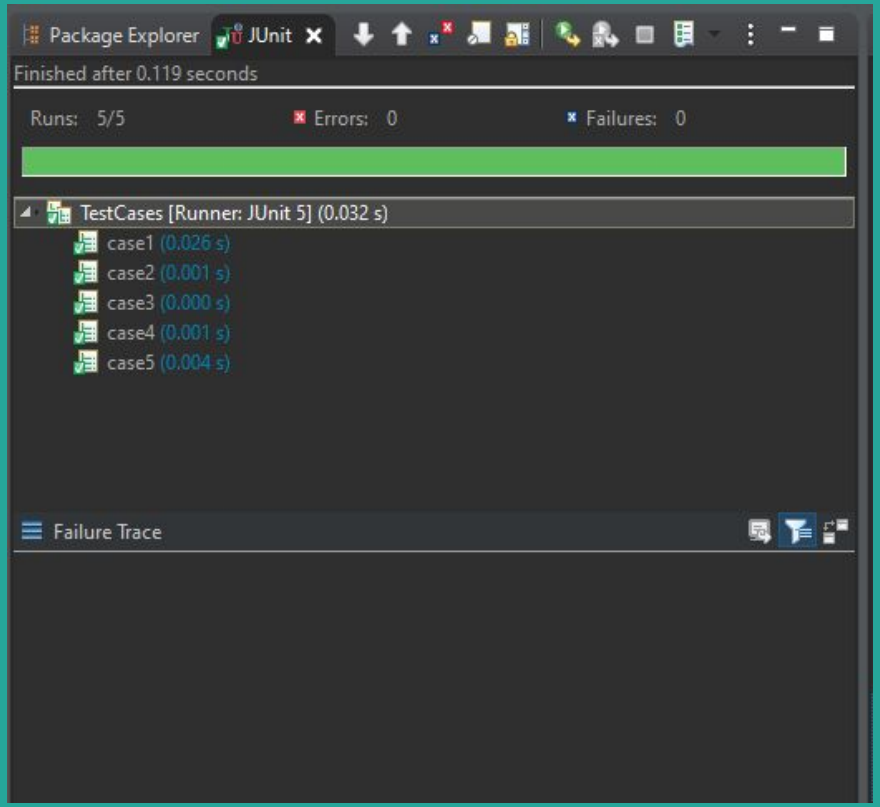
```java
// This case will test for time conflict
@Test
public void case4() throws ClassExceptionError {
    Events events = new Events();

    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());
}
```

5. Testing to see if events can be cleared without any errors.

```java
// 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());
}
```

Test Plan

- Result of unit testing the Events class with Junit
- Results showed everything was successful



Comparison of Our Work with Similar Designs

Outlook Calendar

- Similar Design
- Extended use cases
  - Gives users ability to plan with others
- Developed by larger company

CalTool

- Focused on single user planning
- More simplistic use cases

Comparison of Our Work with Similar Designs

Outlook Calendar

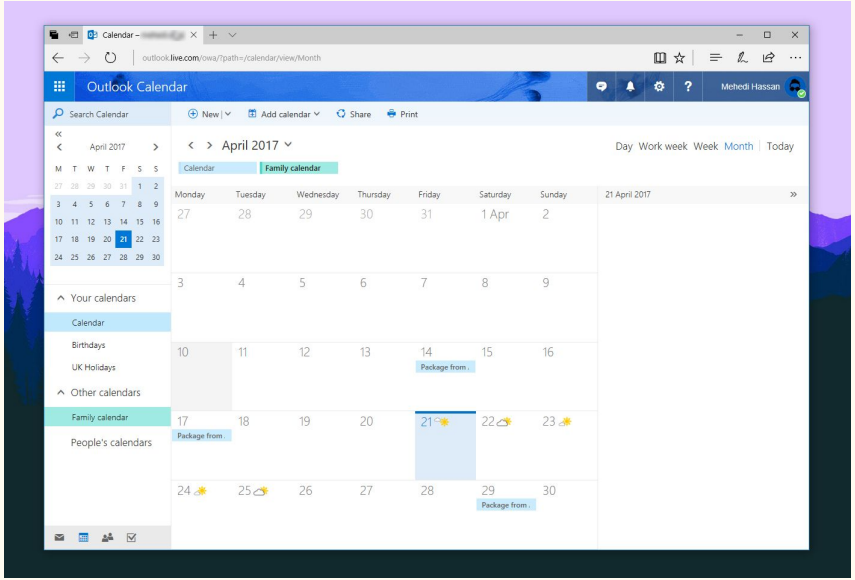


Figure 1: Outlook Monthly View
Source: [1]

CalTool

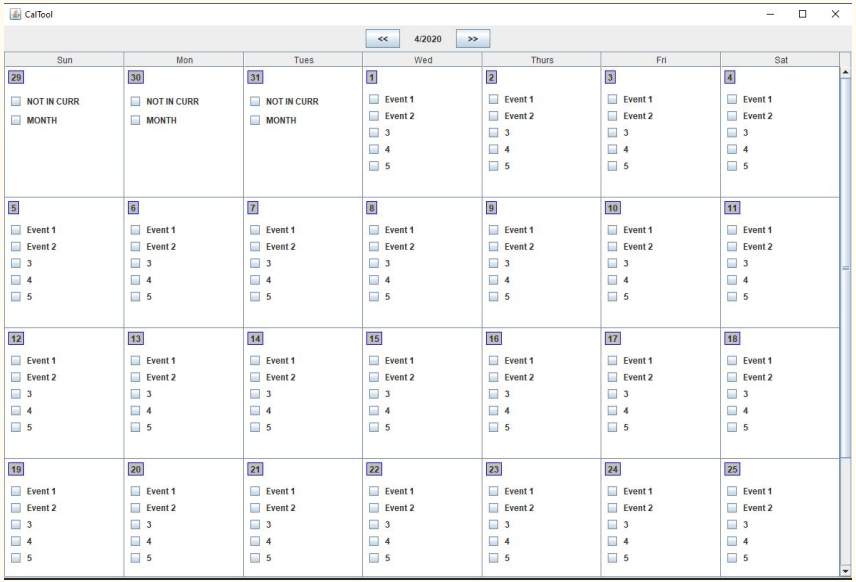


Figure 2: CalTool Monthly View

Conclusion

- CalTool was more complex than we thought
  - Required 4 additional classes
  - Difficult to make good looking GUI in Java
  - Did not realize JUnit testing was required
- Initial motivation was to help plan/schedule our semester
  - Calendar software would help with time management
- Single user ease of use will be helpful to our fellow classmates to plan for their semesters

References

[1] M. Hassan, “Microsoft adds a Family Calendar to Outlook on Windows 10, Android, iOS and more,” MSPoweruser, 21-Apr-2017. [Online]. Available: https://mspoweruser.com/microsoft-adds-family-calendar-outlook-windows-10-android-ios/. [Accessed: 12-Apr-2020].

[2] “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].

[3] “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].