CS/SE/CE 3354 Software Engineering

Project Proposal

Student Academic Calendar Software

Team Members:

- Zaid Hilal
- Zuhaib Buchh
- Nicholas Hudson
- Joshua Feng
- Noah Spain
- Omer Tariq

Github Link:

https://github.com/omert-dev/3354-gate8

Details of What Each member is working on

- Zaid: Slides, report, recording, test code
- Zuhaib: Slides, report, recording, implementation code
- Nicholas: No contribution
- Joshua: Slides, report, recording, github
- Noah: Slides, report, recording
- Omer: Slides, report, recording, group organization

4. Address the feedback provided in the Course Team Project Proposal a. List what you are doing / planning to do regarding the feedback provided for your project proposal

Feedback given: The delegated tasks section only refers to implementation tasks. Include tasks with respect to requirements gathering, design and testing. (for example who will create use case diagrams etc)

- We have updated the delegated tasks section to include responsibilities beyond implementation. We are assigning tasks related to gathering requirements , the system design, and testing to ensure full project coverage.
  Recruitment: Omer, Nicholas, Zaid
  Design: Joshua , Omer, Zuhaib
  Implementations:, Joshua, Nicholas, Zuhaib
  Testing: Noah, Omer, Zaid

Describe which software process model is employed in your project and why it was the choice. (Ch 2)

Our team is using the Incremental Process Model because our project is divided into functional modules, like daily, weekly, monthly calendar views, event management, and category features. Each module represents an increment that can be developed and also tested. These modules will

be integrated independently. The model supports parallel work, so we will have flexibility for improvement.

Software Requirements

A. functional requirements

1. Event Creation and Management - the system will allow users to add, edit, and delete events with details like the title, description, and the start and end time
2. Multiple Calendar views-the System will display multiple formats, including daily, weekly, and also monthly views
3. Agenda Board-the system will have an agenda view that lists all the upcoming events in chronological order
4. Category Management - Will allow users to add, modify, or delete categories
5. Zoom and Navigation Controls- the system will allow users to zoom in and out of calendar views and navigate between dates easily

B. Non-Functional

Product Requirements

1. Efficiency Requirement- the system will be able to load and display calendar views within 2 seconds on standard UTD student devices
2. Reliability Requirement:
   The system will have to maintain at least 99% uptime during operation so that we ensure data integrity during event creation, editing, or deletion.

3. Usability Requirement:
   The interface will be able to be easy to navigate for the user, requiring no more than 5 minutes of user exploration to understand the main features.

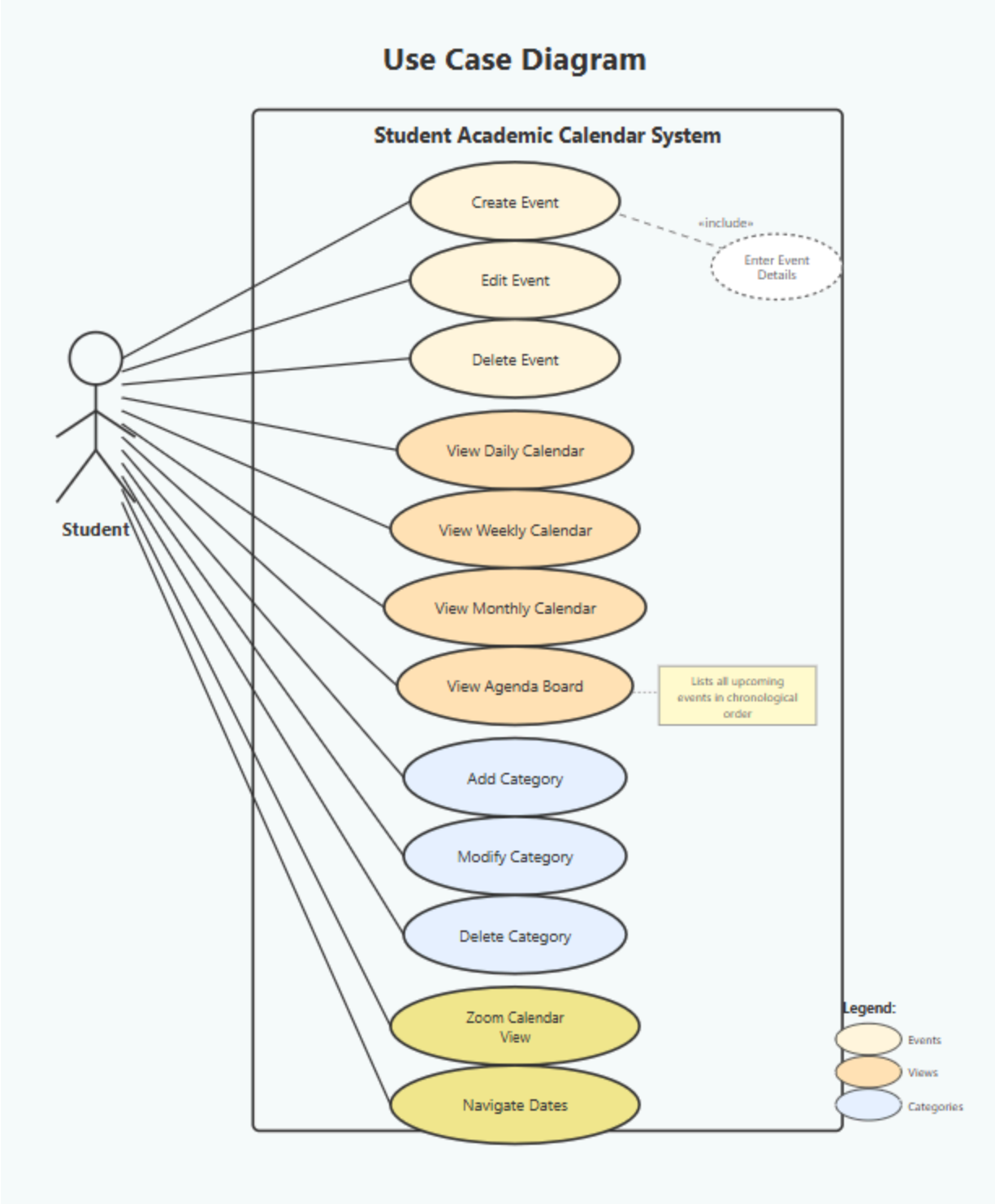
4. Performance Requirement:
   The software will be able to handle at least 500 events per user without any noticeable lag or performance
5. Space Requirements: the application will not exceed 200 MB of local storage for offline use
6. Dependability requirements: The system will maintain 99% uptime during normal operation and ensure no data loss after any shutdown
7. Security Requirements: the system will protect the user data using a secure login authentication and data encryption storage

Organizational Requirements

1. Environmental Requirement- The software will be compatible with Windows, macOS, and Linux operating systems and will be able to run on UTD VPN
2. Operational Requirement- The system will be maintained using version control (our team is using GitHub) and include documentation for setup, usage, and testing procedures.
3. Development Requirements- The project will follow the Incremental Process Model and adopt the MVC (design pattern for modular development.
4. Accounting Requirements- The system will log user actions for example event creation or for debugging and tracking purposes, but without storing sensitive user information.
5. Safety/Security Requirements- The application will ensure that data modification operations (add, edit, delete) are only accessible to the authorized user on the account

External Requirements

1. Regulatory Requirements
   The system will comply with institutional policies on data management and retention for academic tools. (all following UTD guidelines)
2. Ethical Requirements
   The system will respect all user privacy by not collecting any unnecessary personal data from the user and by providing clear consent options on a form before the user starts.
3. Legislative Requirements
   The system shall comply with data protection laws, including FERPA (Family Educational Rights and Privacy Act) and any UTD privacy laws



Explanation of Use Case Diagram

The Use Case Diagram is a visual representation of the Student Academic Calendar System's functionality from the user's perspective. It shows the primary user and the actions (use cases) they can perform within the system to manage their academic schedule.

Actor:

- **Student**: The student is the primary user of the system, and they can manage their academic calendar, creates and organizes events, views their schedule in different formats, and navigates through different times (week, month). The student has control over their personal calendar data and any event categorization they make. UTD is allowed to request addition of school events onto the calendar via email.

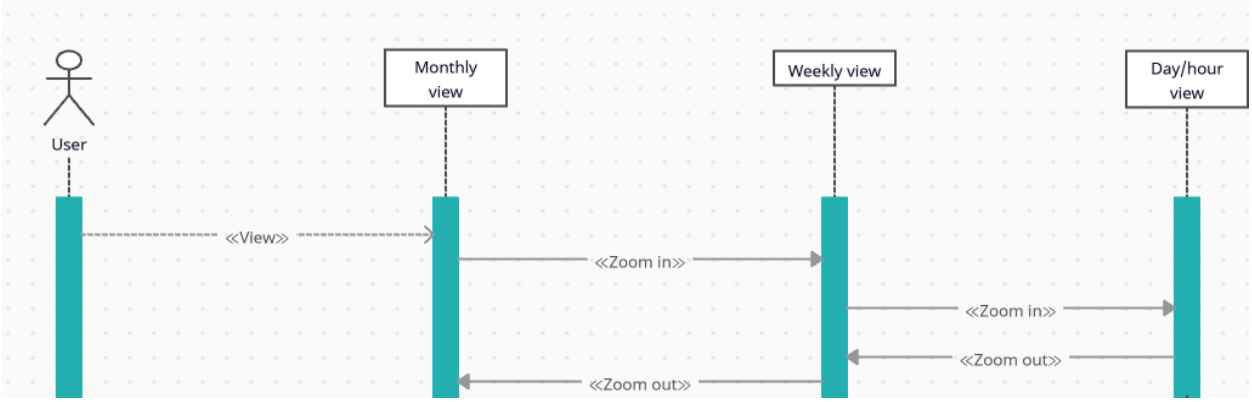
Use Cases:

- **Event Management (Event Creation and Management requirement)**:
    - **Create Events:** the students to add new events to their calendar with details : title, description, start time, and end time.
    - **Edit Event**: Allows students to modify existing event details, including changing times, descriptions, or categories.
    - **Delete Event**: Allows students to remove events from their calendar when they are no longer needed.
    - **Enter Event Details** (included use case): Represents the necessary step of inputting event information during the creation process.
- **Calendar Views (Multiple Calendar Views requirement)**:
    - **View Daily Calendar**: Displays events scheduled for a single day, basically detailed daily planning.
    - **View Weekly Calendar**: Shows an overview of all scheduled events for the week.
    - **View Monthly Calendar**: Shows a monthly overview of events for long term planning.
    - **View Agenda Board**: Lists all the upcoming events in chronological order, having a linear timeline perspective of scheduled activities. (including school)
- **Category Management**:
    - **Add Category**: Allows the students to create custom categories for organizing different types of events for example classes, assignments, extracurricular activities.
    - **Modify Category**: Allows students to edit existing category names or properties.
    - **Delete Category**: Allows students to remove categories that are no longer needed from their system.
- **Navigation and Controls (Zoom and Navigation Controls requirement)**:

- **Zoom Calendar View**: It provides the ability to zoom in and out of calendar views for better detail.
- **Navigate Dates**: Allows the students to move between different dates, weeks, and months to view past or future events.

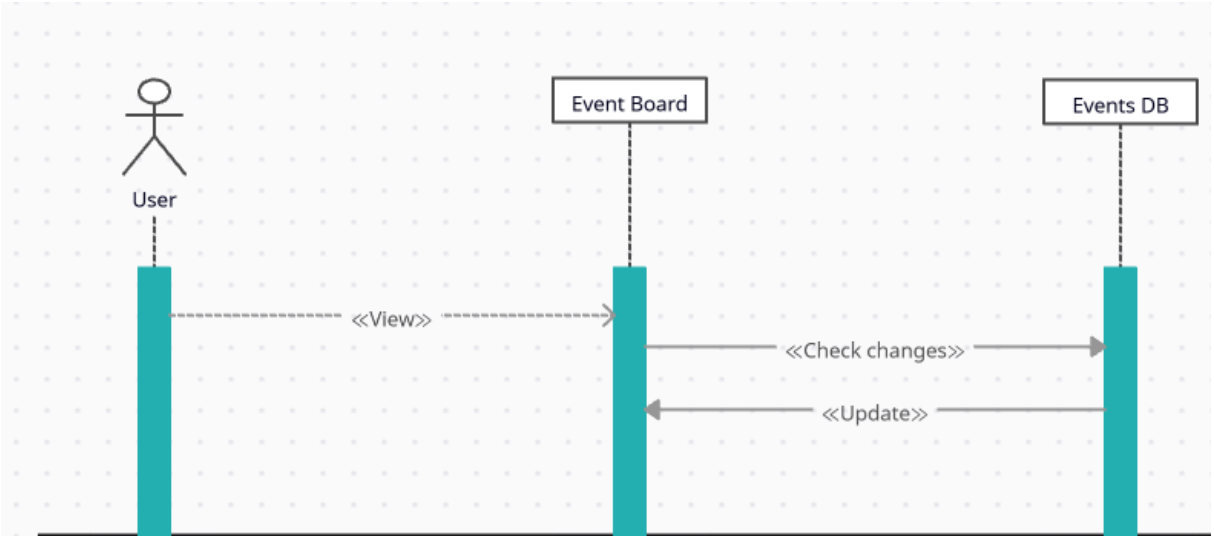
Sequence Diagram

Monthly/weekly/day/hour view:



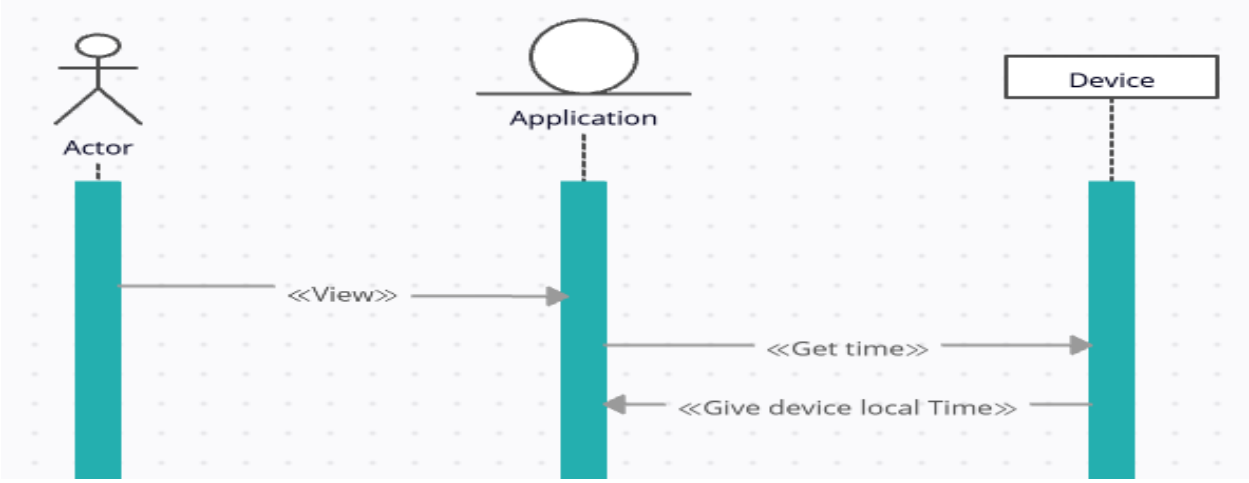
User: Views Calendar
Monthly View: displays days in this month and events correlated to the month
Weekly View: displays days of this week also displays event correlated with this week
Day/hour view: display specific date with events and snippets by hourly

Event Board:



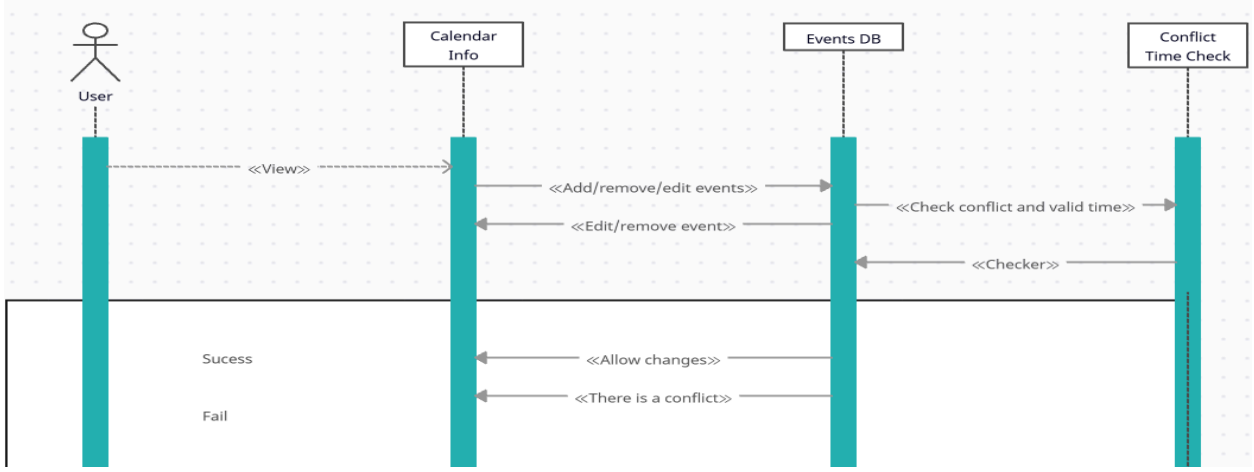
User: views the event board
Event Board: displays events
Event DB: sends stored events to the board to be displayed

Local Time:



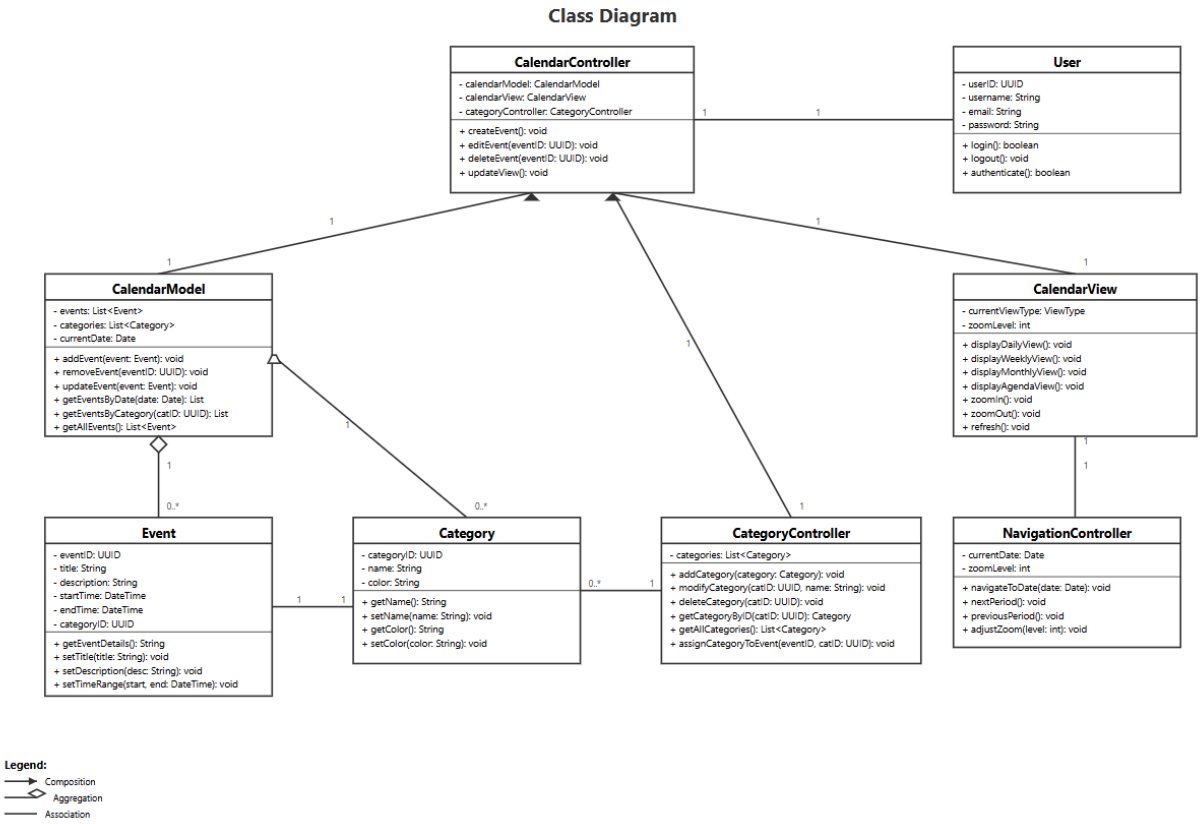
User/Actor: Opens application
Application: runs in the background and while opened
Device: gives application current time and local time

Conflict false time checker:



User: views calendar
Calendar: displays events has interface to add/remove/edit events
Event DB: stores saved events given command edits/removes events
Conflict time check: checks newly added events for conflicts and appropriate time\

Class Diagram



Explanation of Class Diagram

The Class Diagram shows a static view of the Student Academic Calendar System's design, showing the different classes, their attributes, methods, and the relationships between them.

Classes:

- **CalendarController**: The main controller that that makes calendar operations, with some attributes like calendarModel, calendarView, and categoryController, and methods like createEvent(), editEvent(), deleteEvent(), and updateView().
- **CalendarModel**: The core data model with attributes like events, categories, and currentDate, and also with methods like addEvent(), removeEvent(), updateEvent(), getEventsByDate(), and getAllEvents().
- **CalendarView**: Handles the presentation layer with attributes like currentViewType and zoomLevel, and methods: displayDailyView(), displayWeeklyView(), displayMonthlyView(), displayAgendaView(), zoomIn(), zoomOut(), and refresh().

- **Event**: It represents the individual calendar events with attributes like eventID, title, description, startTime, endTime, and categoryID. Methods include getEventDetails(), setTitle(), setDescription(), and setTimeRange().
- **Category**: Represents event categories with attributes like categoryID, name, and color, and methods: getName(), setName(), getColor(), and setColor().
- **CategoryController**: Manages category operations with methods including addCategory(), modifyCategory(), deleteCategory(), getCategoryByID(), getAllCategories(), and assignCategoryToEvent().
- **User**: Represents the authenticated student with attributes like userID, username, email, and password, and methods : login(), logout(), and authenticate().
- **NavigationController**: Handles calendar navigation with attributes like currentDate and zoomLevel, and methods : navigateToDate(), nextPeriod(), previousPeriod(), and adjustZoom().

Relationships:

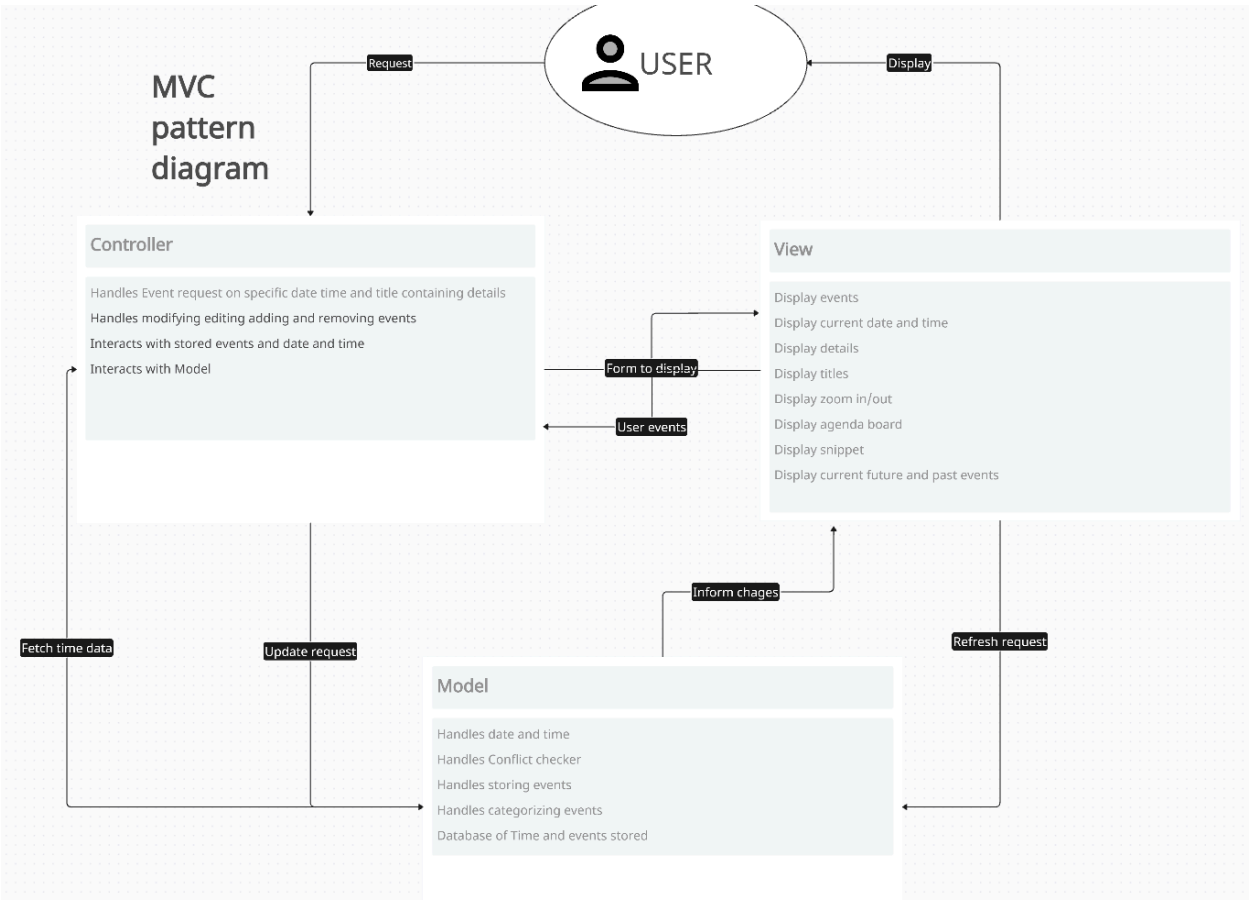
- **Composition**: CalendarController has a strong ownership of CalendarModel and CalendarView (which we indicated by using filled diamond), representing the MVC design pattern where the controller manages both components.
- **Association**: User is associated with CalendarController (1:1), each user has their own controller instance. CalendarController connects to CategoryController (1:1) for category management. NavigationController links to CalendarView (1:1) for navigation control.

The diagram follows the MVC pattern with three layers:

Controller (CalendarController, CategoryController, NavigationController), Model (CalendarModel, Event, Category), and View (CalendarView). The composition relationships show CalendarController as the central component that owns CalendarModel and CalendarView.

The Aggregation allows the CalendarModel to contain many of the of events and categories without tight coupling, this makes it necessary. Association relationships basically define how controllers interact with each other and manage their components. This structure helps us to make sure that the separation of concerns where the Model handles data, the View manages display, and Controllers coordinate operations. The design/UI supports all functional requirements which include (more may be added): event management, multiple calendar views, category organization, and navigation controls.

Architectural design



Explanation of Architectural Design

The Architectural Design diagram is a representation of the high level structure of the system using the MVC (Model View Controller) pattern.

Components:

- **User**: The student who interacts with the calendar system, starting actions like creating, editing, or viewing events.
- **Controller**: Processes user requests/input and manages data flow. It handles any of the event requests, modifies, edits, adds, and removes events, and makes the bridge between the Model and View.

- **View**: The presentation layer that displays calendar information. It shows events in daily, weekly, monthly, and agenda formats, also with current date and time information.
- **Model**: The data layer that manages date and time information, event storage, calendar data, categories, and handles conflicts and event intervals.

Interaction Flow:

- User interacts with the View by selecting options or creating events.
- View sends user actions to the Controller.
- Controller processes requests and communicates with the Model for data operations.
- Model performs data operations and returns results to the Controller.
- Controller updates the View with new data.
- View displays updated information to the User.

This MVC architecture allows the separation of concerns where the Model manages data, the View handles presentation, and the Controller coordinates between them. This design makes the system maintainable and keeps components able to be modified independently.

    -

4. Scheduling, Cost, Effort & Pricing Estimation

a. Project Scheduling

i. Start and end date with justification

Our project started on September 5, 2026, and will end on June 26, 2026.
The duration of the project is reasonable because it includes multiple phases like the requirements gathering, system design, implementation, testing, and deployment. Based on prior academic software project patterns, six months provides our team enough time to build, revise, and validate the solution without rushing any of the critical stages like testing or user feedback.

ii. Additional schedule details

1. **Whether weekends will be counted in the schedule or not**
   Weekends will not be counted as working days. Work will only occur Monday through Friday, following standard professional software development practices. Excluding weekends helps maintain a steady pace, prevents burnout, and ensures consistent productivity.
2. **Number of working hours per day**
   Our team of student developers will work 2 hours per day on the project. This allows us to balance academic coursework with steady project progress, while still maintaining a development schedule that is consistent.

b. Cost, Effort, and Pricing Estimation

i. Method used to calculate estimated cost and price

The cost estimation we are using is the Function Point (FP) cost estimation method.
The Function Point technique basically measures the size of the project using the number of user inputs, outputs, database files, and external interactions. The total number of Function Points is multiplied by a standard number of hours required to implement each FP. This produces an estimate of total development hours, which are then multiplied by an hourly labor rate to determine cost. Finally, a profit margin is added to determine the customer price of the project.

ii. Cost Modeling Technique Applied

The selected technique is:

1. Function Point (FP) (chosen method)

- The system contains inputs, outputs, inquiries, internal database files, and external interfaces.

- After identifying and applying complexity factors, the project was calculated to be approximately **135 Function Points**.

- Industry average: **1 Function Point ≈ 5 development hours**

- Total development effort:
  135 FP × 5 hours = 675 total development hours

This effort estimate will help us calculate labor cost and final project pricing in the next section.

c. Estimated cost of hardware products.

The required hardware includes servers and backup storage for a projected 6 months of testing and then deployment. For calculations, 183 days of time was used instead of 6 months because it is just over half a year. For a server, an Amazon EC2 T2.small server would suffice. On demand prices are $0.023 per hour [https://aws.amazon.com/ec2/instance-types/t2/]. For 183 days this would cost **$101.02**.

For storage, 50GB of Amazon S3 storage would suffice. It costs $0.023 per GB per month [https://aws.amazon.com/s3/pricing/]. For 6 months of storage 50GB would cost **$6.90**.

The total hardware cost is **$108.10**. If a larger server or more storage are required as the project grows, AWS services can easily be scaled to match the growth of the product.

d. Estimated cost of software products.

The software used in this project is free, open-source, or available under licenses that permit their use. For example, tools like IntelliJ IDEA, JUnit, and GitHub can all be used for free. Any database software required for development, like SQL, has no cost other than the servers and storage described in section 4c. Therefore the estimated cost of software products is **$0**.

e. Estimated cost of personnel.

The development hours required for this project is 675 as calculated using function point analysis in section 4b. These hours would be split between developers, quality assurance / testers, and a project manager. With a cost of $45/hour for the developmental tasks, it would cost $30,375 to pay employees. As a buffer we add 5% to account for training and any unforeseen overhead of managing the project. This brings the total estimated cost of personnel to **$31,893.75**.

5.

a.

Unit to test: `checkTimeConflicts(startTime, endTime, excludeEventId)`

Purpose: Ensure the method correctly detects overlapping events and ignores an event with the `excludeEventId`.

Test Strategy:

Provide an array of mock events.

Checks:

1. **No Conflict:** Verify that the method returns an empty array if the proposed event does not overlap with any existing events.
2. **Conflict Detected:** Verify that overlapping events are returned correctly.
3. **Exclude Event ID:** Verify that a specific event can be excluded from conflict checking.

Test Type:
Unit Test — testing a single logical method independently.

Test Environment:

- Module Programming Language: JavaScript
- Testing Tool :Jest
- IDE: VS Code
- No external dependencies required

b.

CheckTimeConflicts Method Code:

```
checkTimeConflicts(startTime, endTime, excludeEventId = null) {
```

```
        return this.events.filter(event => {

            if (event.id === excludeEventId) return false;



            const eventStart = new Date(event.startTime);

            const eventEnd = new Date(event.endTime);



            return (startTime < eventEnd && endTime > eventStart);

        });

    }
```

Jest Test Case

```
import { CalendarApp } from './CalendarApp.js';

describe('CalendarApp checkTimeConflicts', () => {
  let app;

  beforeEach(() => {
    app = new CalendarApp();
    app.events = [
      { id: '1', startTime: '2025-11-13T10:00:00', endTime:
'2025-11-13T11:00:00' },
      { id: '2', startTime: '2025-11-13T12:00:00', endTime:
'2025-11-13T13:00:00' }
    ];
  });

  test('no conflict returns empty array', () => {
    const conflicts = app.checkTimeConflicts(
      new Date('2025-11-13T11:00:00'),
      new Date('2025-11-13T12:00:00')
    );
    expect(conflicts).toEqual([]);
  });
```

```
  test('overlapping event returns conflict', () => {
    const conflicts = app.checkTimeConflicts(
      new Date('2025-11-13T10:30:00'),
      new Date('2025-11-13T11:30:00')
    );
    expect(conflicts.length).toBe(1);
    expect(conflicts[0].id).toBe('1');
  });

  test('excludeEventId ignores that event', () => {
    const conflicts = app.checkTimeConflicts(
      new Date('2025-11-13T10:30:00'),
      new Date('2025-11-13T11:30:00'),
      '1'
    );
    expect(conflicts).toEqual([]);
  });
});
```

c.

Test Cases and Results

Test Case	Input	Expected Result	Actual Result
No conflict	11:00–12:00	`[]`	`[]` ✅
Overlap with Event 1	10:30–11:30	`[Event 1]`	`[Event 1]` ✅
Exclude Event 1	10:30–11:30, exclude '1'	`[]`	`[]` ✅

6. Comparison to Similar Existing Solutions

- We are able to closely match our users' needs and maintain total control over data and customization by providing the exact workflows and data models that they require. The project specification serves as the basis for our project's needs and scope.
  Citation: Projectpart3.Gate8.CS3354.006

- Commercial off-the-shelf (SaaS) solutions are less flexible to project-specific workflows and may result in vendor lock-in or recurring subscription fees, despite providing vendor support, a speedy time to market, and well-designed user interfaces. For general guidance on the advantages and disadvantages of custom versus off-the-shelf software, see FullScale's comparison [1].

- Although open-source platforms are extremely expandable, transparent, and have reduced license costs, they usually require more labor for setup, integration, hardening, and maintenance unless a commercial vendor provides paid support. For a helpful overview of the trade-offs between proprietary and open-source software, see Heavybit's debate [2].

- Usability and UX quality are key differentiators; reputable providers often invest more in UX research and well-designed interfaces. A recommended, practical checklist for evaluating and improving usability in our design is Jakob Nielsen's 10 usability heuristics

[3].

7. Conclusion:

a. Evaluation of Our Work i. Changes Made from the Original Plan: As we worked through the project, we ended up making a few changes from what we originally planned. The biggest one was expanding our delegated tasks section. At first, we pretty much just focused on who was doing what for the actual coding part. After getting feedback from our Professor, we realized we needed to show who was responsible for requirements, design, and testing too. This way, every part of the development process had someone clearly in charge of it.

We also went back and cleaned up our system architecture and UML diagrams (Use Case, Sequence, Class, and Architectural Design) to make sure they actually matched the MVC structure we were using. Honestly, this helped a lot because it made the interactions between different parts of the system way clearer and easier to understand. Another thing we changed was our cost estimation, we switched to using Function Point Analysis instead of our original method because it's more accurate and actually used in the industry.

One last addition we made was a detailed unit test plan with a JUnit example. We didn't have this in our first draft, but it really helped show that we were serious about testing and making sure our code actually works.

ii. Justification for Changes: We made these changes because of the feedback our Professor gave us, and to make our project look more complete and professional. Redoing the delegated tasks helped us spread out the work more evenly and made sure everyone was involved in different parts of the project, not just the coding. Updating our architecture and UML diagrams helped us stay consistent and made our design decisions clearer.

Switching to Function Point for cost estimation just made more sense. It's what actual companies use, and it gave us better numbers. Adding the automated testing showed we weren't just writing code and hoping it worked, but actually had a plan for quality control.

Basically, all these changes made our project feel more like something you'd see in a real workplace instead of just a class assignment. We still hit all the requirements from the project guidelines, but now it's more polished and realistic.

8. References (IEEE Citation Style

[1] M. Watson, “Custom Software vs. Off-the-Shelf Solutions: A Complete Cost-Benefit Analysis for Growing Businesses,” *Full Scale*, Jun. 19, 2025. [Online]. Available: https://fullscale.io/blog/custom-software-vs-off-the-shelf-cost-analysis/

[2] A. Park, “Open-Source Software vs. Proprietary Software: What to Know,” *Heavybit*, Apr. 13, 2023. [Online]. Available: https://www.heavybit.com/library/article/open-source-vs-proprietary

[3] J. Nielsen, “10 Heuristics for User Interface Design,” *Nielsen Norman Group*, Apr. 24, 1994. [Online]. Available: https://www.nngroup.com/articles/ten-usability-heuristics/