Calendar Management System

Team 15

Project Overview

Key Functionalities

- Calendar management system for personal and shared calendars
- Allows users to:
  - Create, edit, and delete events
  - Load and save calendars
  - Filter events by category
  - Display calendar in daily, weekly, and monthly format
- Designed for multi-user access

What Problems Does Our System Solve?

- Difficulty coordinating across multiple schedules
  - Users can share calendars and manage access permissions, reducing confusion when planning events.
- Inconsistent and lacking event creation tools
  - Supports personalized labels, and filtering to meet individual planning needs.

Developmental Approach

Chosen Method: Plan-Driven

- Everything was planned upfront, including tasks and deadlines
- Why it fit our project:
  - Our system’s requirements were clear and stable from the start
  - Predictable timeline and role assignments improved coordination

Use of Deliverables I & II

Stakeholders

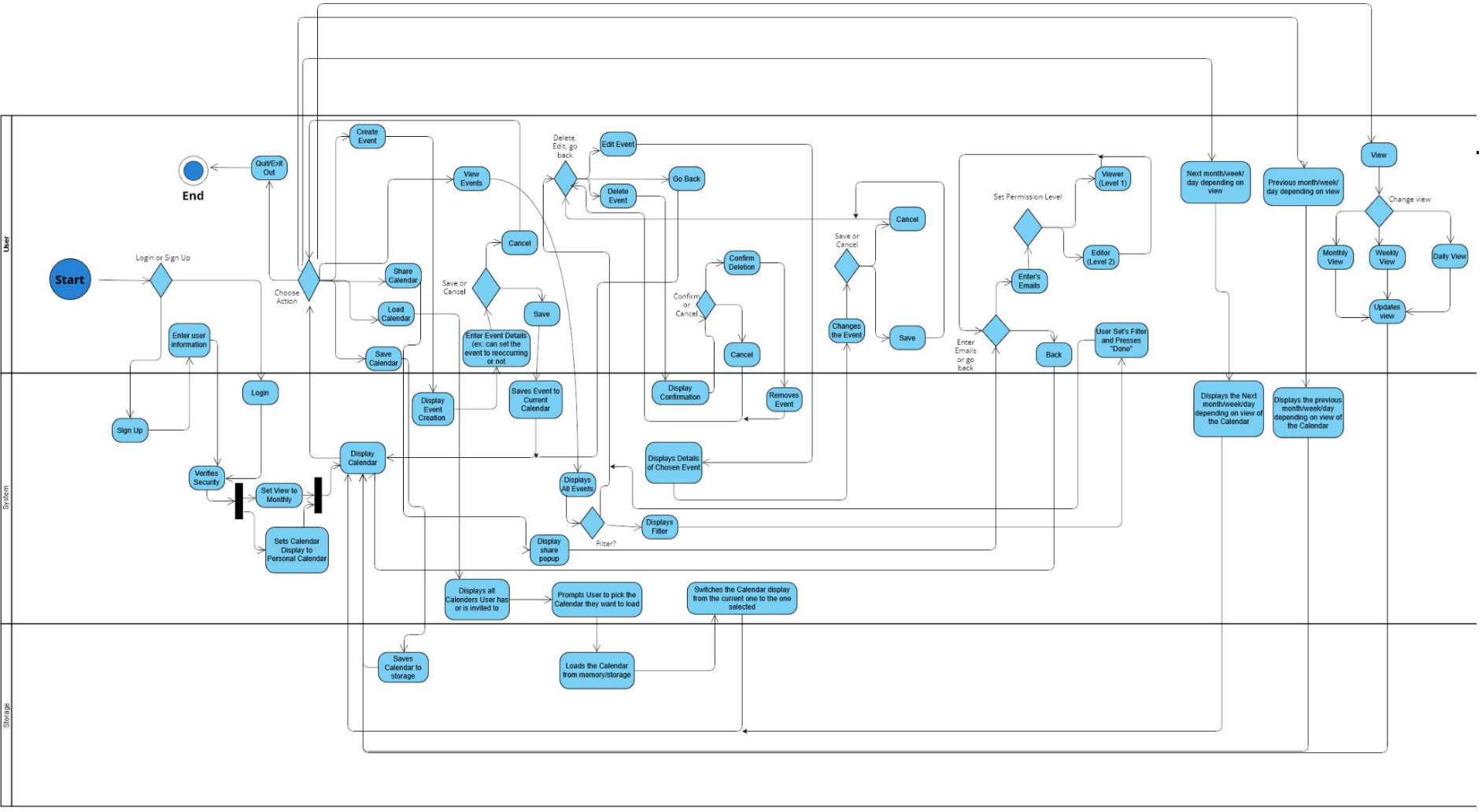
Stakeholders wants and needs:
End users want to be able to add, edit, and delete events so they can organize their tasks and other events in an orderly manner.
Stakeholders don’t want to have to input multiple of the same events, making it more feasible for them to use the calendar as it’s UI friendly.
Students want to be able to filter their events by category so they don’t have to see all their events at once and feel less overwhelmed.
Teams/Groups want to be able to give individual users certain permissions so they don’t have full admin access to the calendar in order to prevent them from messing up the work flow.
End users want to be able to add events and be able to view them whenever they open the application instead of manually adding it everytime. Makes the calendar more usable
Stakeholders want to be able to see events during a certain week or day with more details than just a event marker on the month mode of the calendar. Making it easier to view events they’ve stored.

Functional Requirements

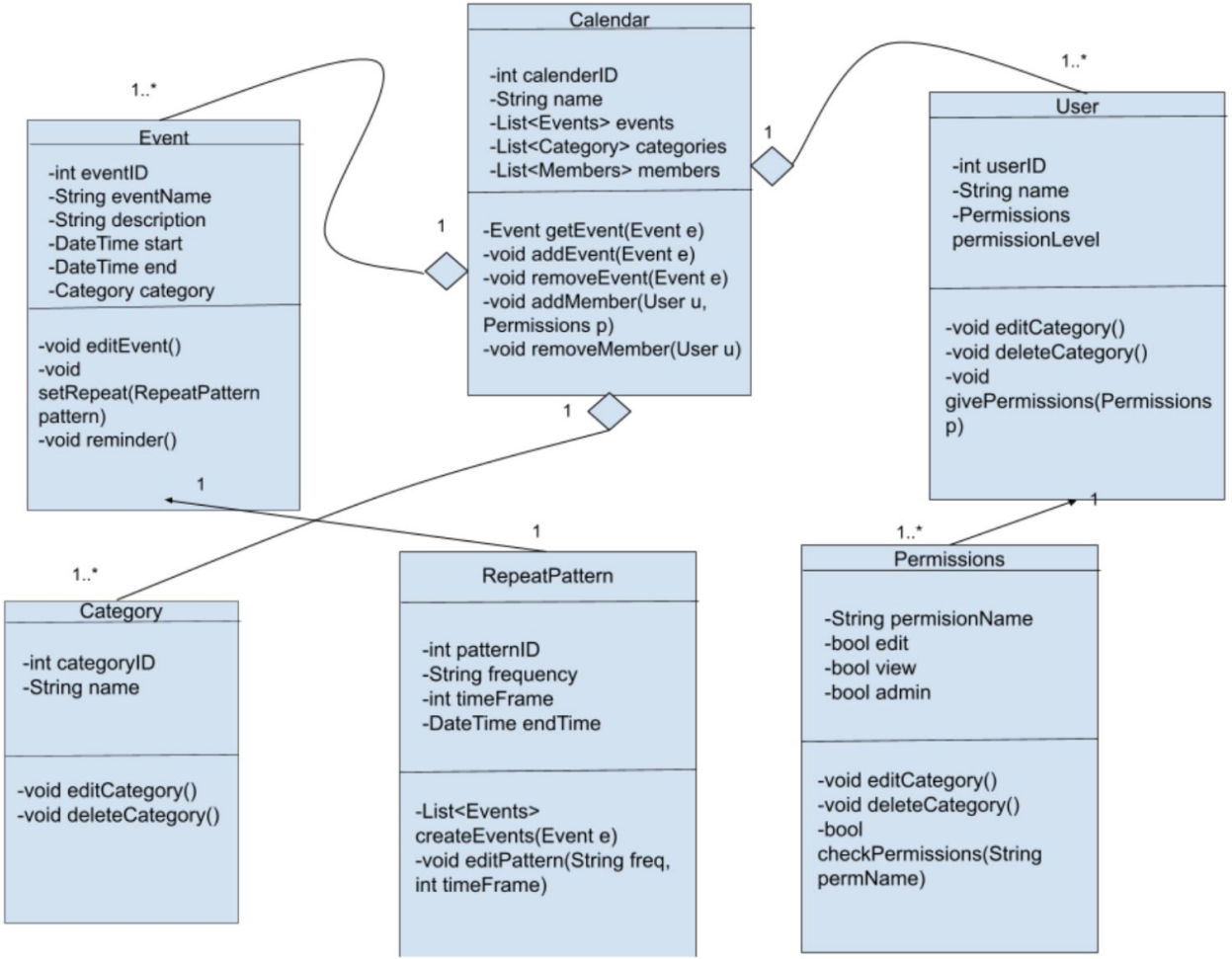
Requirements	*Design Decision*	*Implementation Idea*
The system shall allow the user to create, edit, and delete events.	- Included in behavioral Model (“Create Event,” “Edit Event,” “Delete Event”) - Form dialog and details dialog designed early - Event flow modeled in Process Model and Behavioral Model	addEvent(), editEvent(), deleteEvent() Files: eventForm.js, eventFormDialog.js, eventDetails.js, eventDelete.js, eventStore.js
The system shall allow the user to establish recurring events.	- Added recurrence to our Structural Model - An additional aspect of “create event”	setRepeat() Files: eventForm.js, eventFormDialog.js, eventDetails.js, eventStore.js, date.js
The systems shall allow the user to filter the events by category.	- UI included category color coding - Filter control planned in process model	filter() Files: eventForm.js, event.js, eventStore.js,
The systems shall allow the user to assign permission levels.	- Planned for multi-user scenarios (collaboration with backend) - Permission settings represented in Structural Model and Process Model	setPermission() Files: eventForm.js
The systems shall allow users to load and save calendars.	-Store to local storage temporarily and then connect to backend to store it permanently	eventStore() Files: eventStore.js
The system shall let the user see the calendar in either a daily, weekly, or monthly format.	- Designed three separate rendering modules (month, week, day) - Included navigation bar and view-switcher in UI design - Week view required special plan for overlapping events - View change flow represented in Process Model	selectView() Files: calendar.js

Non Functional Requirements

Requirements	***Design Decision***	***Implementation Idea***
The system shall store user data securely.	- Chose a client-side storage model (localStorage then to backend) to ensure load balancing so traffic doesn't heavily impact the client	eventStore.js stores only non-sensitive scheduling data.
The system shall provide an intuitive experience such that a new user can be able to create and edit an event within the first five minutes.	- Created dialog-based UI for event creation and editing. - Used simple form fields with clear labels (Title, Date, Start Time, End Time, Category, Recurrence). - Planned consistent icons and buttons (edit, delete, close).	eventFormDialog.js opens the form instantly with prefilled defaults. eventDetails.js makes editing accessible through a single click.
The system shall respond to users' interaction within 3 seconds.	- Planned separation of views (month/week/day) for faster re-rendering - Create simple methods that don't take much time to compute	Event creation/edit/delete operations trigger rerenders instantly via calendar.js


User
System
Storage
Start
Login or Sign Up
Enter user information
Sign Up
Login
Verifies Security
Set View to Monthly
Sets Calendar Display to Personal Calendar
Choose Action
Quit/Exit Out
End
Create Event
Share Calendar
Load Calendar
Save Calendar
Display Calendar
Saves Calendar to storage
View Events
Cancel
Save or Cancel
Save
Enter Event Details (ex: can set the event to reoccurring or not
Display Event Creation
Saves Event to Current Calendar
Displays all Calenders User has or is invited to
Prompts User to pick the Calendar they want to load
Loads the Calendar from memory/storage
Switches the Calendar display from the current one to the one selected
Delete, Edit, go back
Edit Event
Delete Event
Go Back
Confirm Deletion
Confirm or Cancel
Cancel
Display Confirmation
Removes Event
Displays Details of Chosen Event
Displays All Events
Filter?
Displays Filter
Display share popup
Save or Cancel
Cancel
Changes the Event
Save
Set Permission Level
Viewer (Level 1)
Editor (Level 2)
Enter's Emails
Enter Emails or go back
Back
User Set's Filter and Presses "Done"
Next month/week/day depending on view
Previous month/week/day depending on view
Displays the Next month/week/day depending on view of the Calendar
Displays the previous month/week/day depending on view of the Calendar
View
Change view
Monthly View
Weekly View
Daily View
Updates view

Structural Model



Review Check

-***Verifiability***
- Is the requirement realistically testable?

-***Comprehensibility***
- Is the requirement properly understood?

-***Traceability***
- Is the origin of the requirement clearly stated?

-***Adaptability***
- Can the requirement be changed without a large impact on other requirements?

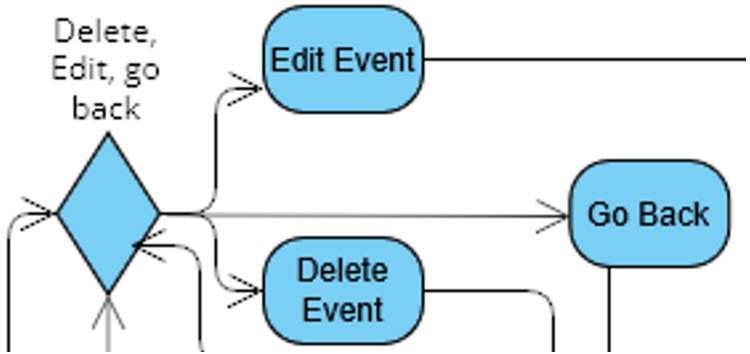
FR-1	Functional	The system shall allow the user to create, edit, and delete events.	Event Management

```
[Test]
0 references
public void AddEvent_ShouldAddEventToEventsList()
{
    var calendar = new Cal("Bob", "Personal");

    var start = DateTime.Now;
    var end = start.AddHours(1);
    var newEvent = new CalendarEvent("Meeting", "Project planning", start, end);

    calendar.AddEvent(newEvent);

    Assert.That(calendar.Events.Count(), Is.EqualTo(1));
    Assert.That(calendar.Events.First(), Is.EqualTo(newEvent));
}
```



Architectural Pattern

Chosen Pattern: **Client Server Model**

- ***Client***:
- Runs the user interface in the browser (HTML, CSS, JavaScript).
- Sends login credentials (username/password) to the server for verification.
- Requests user-specific event data from the backend after authentication.
- Sends create/edit/delete event requests to the server using API calls
- Renders the calendar views (day/week/month) using data received from the server.

- ***Server***
- Stores user event data in a server-side database (instead of local storage).
- Enforces permission levels, ensuring users can only modify their own events.
- Handles read/write operations securely, ensuring data integrity.
- Responds to API requests with JSON data for the client to render.

- ***Client ↔ Server***
- Server responds with JSON objects representing event data.
- The client updates the UI based on server responses.
- All users see consistent data across devices, because data lives on the server.

Development Process and Timeline

Process Model

Chosen Model: **Waterfall Model**

Requirement Definition → System/Software Design → Implementation → System Testing → Maintenance

- Each phase was completed before progressing onto the next step to ensure stability and clear documentation at every stage
- Deliverable 1: Got requirements
- Deliverable 2: Made models and design ideas and how to implement them
- Started implementing them after models were finished
- Deliverable 3: Tested the system
- Pre Presentation: Fixed up the code and any bugs from the test
- No going back to our requirements and changing them as they were already fixed and didn't need to evolve

Waterfall Phases - Complete Timeline

Requirements Definition	System Design	Implementation	System Testing
1. Functional requirements specification 2. Non-functional requirements specification 3. Requirements validation and review **Deliverable 1:** Requirements Specification Document (Sept 16th - Sept 26th)	1. High-level architectural design 2. Low-level system design 3. Design review and approval **Deliverable 2:** Design Document / UML Diagrams (Sept 27th - Oct 19th)	1. Development environment setup 2. Module development 3. Unit testing for components 4. Code review and integration **Deliverable 3:** 50% of requirements developed (Oct 20th - Nov 9th)	1. Integration testing 2. System testing 3. Acceptance testing 4. Bug tracking and resolution **Deliverable 3.5:** Test Report and Verified System (Nov 10th - Nov 17th)

Reflection: Planning, Iteration, and Teamwork

What we did	What we would do differently
➢ At the end of each phase we would meet and verify the results before beginning the next phase ➢ Did most of the refining and documentation at the end of each phase which sometimes led to overlooked details ➢ Roles and responsibilities were decided at the beginning of each phase	➢ Meet more frequently. Because waterfall is so strict with scheduling it's important to constantly verify that everyone is progressing smoothly so weekly checkups/progress reports would have been helpful ➢ Spent more time refining and documenting requirements and design details early to avoid later adjustments ➢ We would have reassessed responsibilities more frequently to ensure workload balance and prevent any team member from becoming overburdened with difficult tasks

Implementation Summary

Main Features Mapped to Requirements

Feature Implemented	Description	Requirement ID
Create, edit, and delete events	The system shall allow the user to create, edit, and delete events	FR-1
Filter events by category	The systems shall allow the user to filter the events by category	FR-3
Permission Assignment	The systems shall allow the user to assign permission levels	FR-4
Serialization Support	The systems shall allow users to load and save calendars	FR-6
Calendar View Modes	The system shall let the user see the calendar in either a daily, weekly, or monthly format	FR-7

Modularity, simplicity, and reusability

- Modularity
  - Calendar, Event, Permissions, and Serialization were implemented as independent components to reduce coupling and improve maintainability
- Reusability
  - Backend logic for filtering, permissions, and event updates is shared across daily/weekly/monthly views to avoid duplicate code
- Simplicity
  - All core logic centralized on the server to reduce complexity and avoid duplicated code

Programming Languages And Frameworks

Tool / Technology	**Version**
.NET SDK	8.0
ASP.NET core	8.0
C#	12
HTML	W3C standard
JavaScript	ES14
CSS	CSS3
VScode	1.94+
VS / Windows	11
JetBrains Rider	2024.3
NUnit	3.14.0

Testing & Validation

Testing Approach

- Used NUnit testing framework to test backend code
  - Manually Written Automated Tests
  - Tests based on our requirements
- Manual UI testing for frontend code
  - Focused on functionality
  - Verified Error Handling
  - Tested Edge Cases
  - Interaction Testing

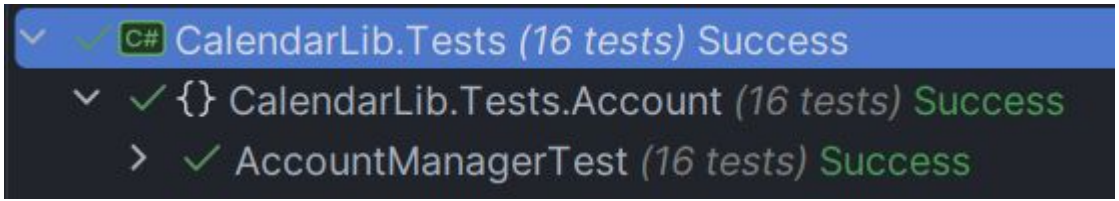
Testing Results

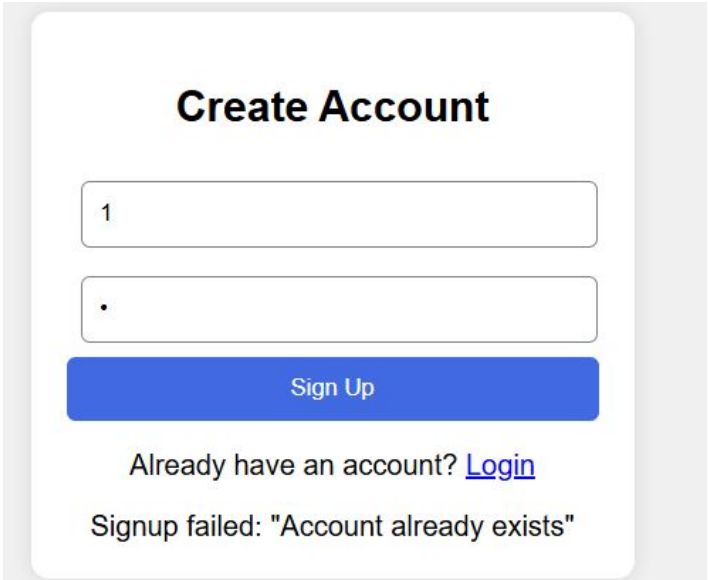
```
[TestFixture]
[TestOf(typeof(AccountManager))]
Error-String-Expected-Got-Nil
public class AccountManagerTest
{
    private readonly IDataStorage<User> _storage = new MemoryStorage<User>();
    private AccountManager _manager;

    [SetUp]
    Error-String-Expected-Got-Nil
    public void SetUp()
    {
        _manager = new AccountManager(_storage);
    }

    [Test]
    Error-String-Expected-Got-Nil
    public void CreateAccount_Basic()
    {
        // Ensure that creating an account works and the account is placed into storage
        var result = _manager.CreateAccount( username: "test",  password: "password");
        Assert.That(result,  expression: Is.EqualTo( expected: AccountResult.Success));
        Assert.That( actual: _storage.IsExtant( id: "test"), Is.True);

        // While we're at it, let's also verify the storage is working correctly. Will cause problems later if it isn't.
        Assert.That( actual: _storage.Delete( id: "test"), Is.True);
        Assert.That( actual: _storage.IsExtant( id: "test"), Is.False);
    }
```





Testing Benefits

- Caught Bugs Early
- Ensured Consistent Behavior Across System
- Improved User Experience

Demo

Core Functionalities

- Create Account and Login
- Create a calendar and event
- Share a calendar, Viewer or Editor
- Edit and Delete events
- Delete accounts

Reflection

Major Challenges

- Time Management
- Requirement Planning

Improvements

- More meetings
- Better communication among teammates