

# CS 320 Course Project Final Report

for  
**Group Calendar**

**Prepared by**

**Group Name:** Team JJ

Jakob Miner  
Joseph Van Boxtel

11659517  
11690601

[jakob.miner@wsu.edu](mailto:jakob.miner@wsu.edu)  
[joseph.vanboxtel@wsu.edu](mailto:joseph.vanboxtel@wsu.edu)

**Date:** December 13, 2019

<b>CONTENTS</b>	<b>ii</b>
<b>1 INTRODUCTION</b>	<b>1</b>
1.1 PROJECT OVERVIEW	1
1.2 DEFINITIONS, ACRONYMS AND ABBREVIATIONS	1
1.3 REFERENCES AND ACKNOWLEDGMENTS	1
<b>2 DESIGN</b>	<b>2</b>
2.1 SYSTEM MODELING	2
2.2 INTERFACE DESIGN	2
<b>3 IMPLEMENTATION</b>	<b>3</b>
3.1 DEVELOPMENT ENVIRONMENT	3
3.2 TASK DISTRIBUTION	3
3.3 CHALLENGES	3
<b>4 TESTING</b>	<b>4</b>
4.1 TESTING PLAN	4
4.2 TESTS FOR FUNCTIONAL REQUIREMENTS	4
4.3 TESTS FOR NON-FUNCTIONAL REQUIREMENTS	4
4.4 HARDWARE AND SOFTWARE REQUIREMENTS	4
<b>5 ANALYSIS</b>	<b>5</b>
<b>6 CONCLUSION</b>	<b>6</b>
<b>APPENDIX A - GROUP LOG</b>	<b>7</b>

# 1 Introduction

## 1.1 Project Overview

Our Group Calendar project, which we have named Cobra Calendar, is a web application that allows groups of users to quickly and easily view a calendar displaying shared availability. Its primary use is for groups of users to plan meetings and events around multiple busy schedules, minimizing reschedules and rain-checks. By highlighting only available times, it is easy to find an appropriate time block, even with large groups of busy individuals.

Users are able to create groups, and invite their friends and coworkers to said groups. The groups allow users to see a simplified view of any or all members of the group's events, making it easy to identify time periods that do not conflict with anyone's schedule without invading the privacy of any of the group members.

## 1.2 Definitions, Acronyms and Abbreviations

*iCal: Internet Calendaring and Scheduling Core Object Specification (iCalendar) file format.*

## 1.3 References and Acknowledgments

*"iCalendar.org - iCalendar Resources, Specifications and Tools," iCalendar.org - iCalendar Resources, Specifications and Tools. [Online]. Available: <https://icalendar.org/>. [Accessed: 26-Oct-2019].*

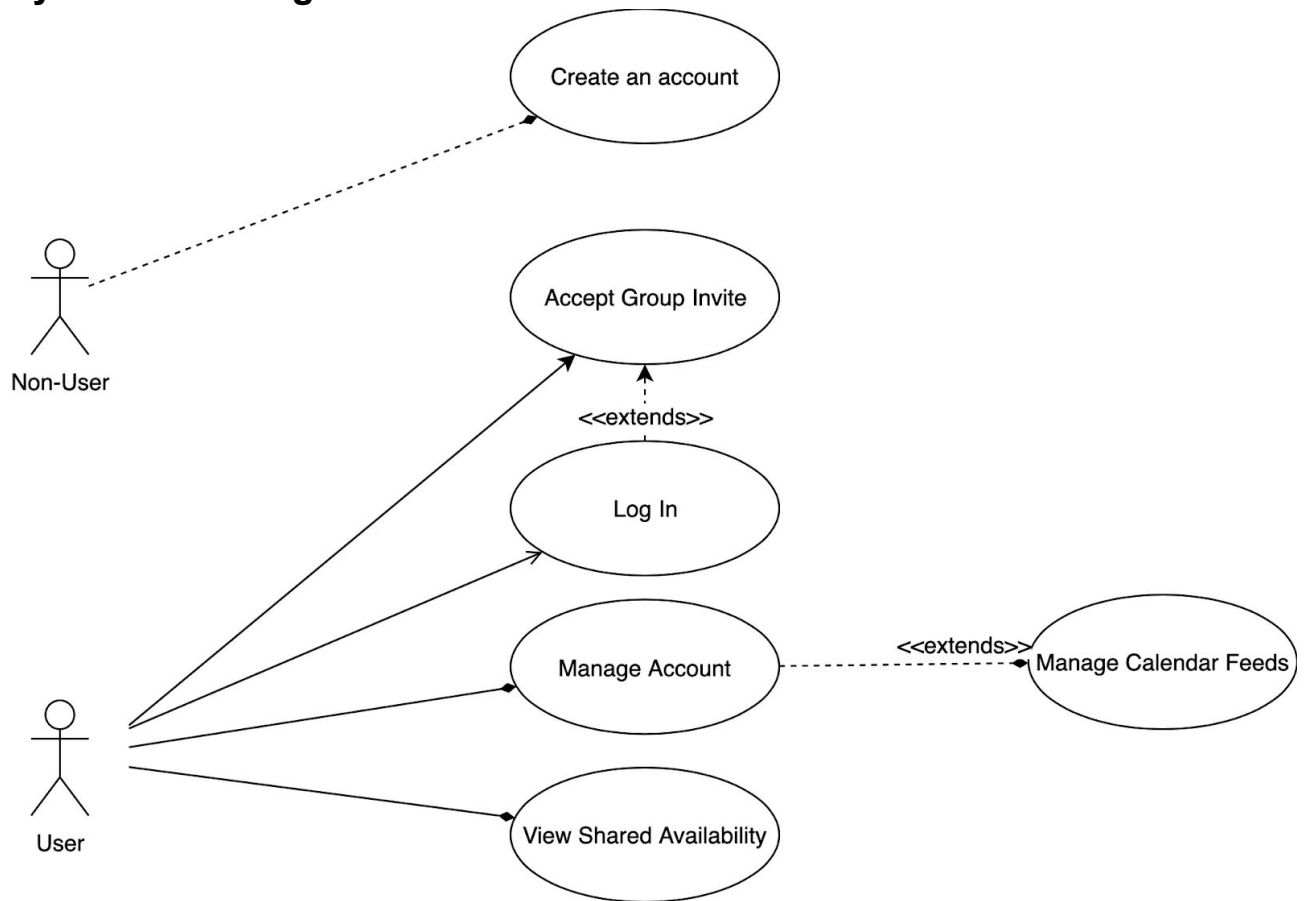
*P. Braden, "ical," npm. [Online]. Available: <https://www.npmjs.com/package/ical>. [Accessed: 12-Dec-2019].*

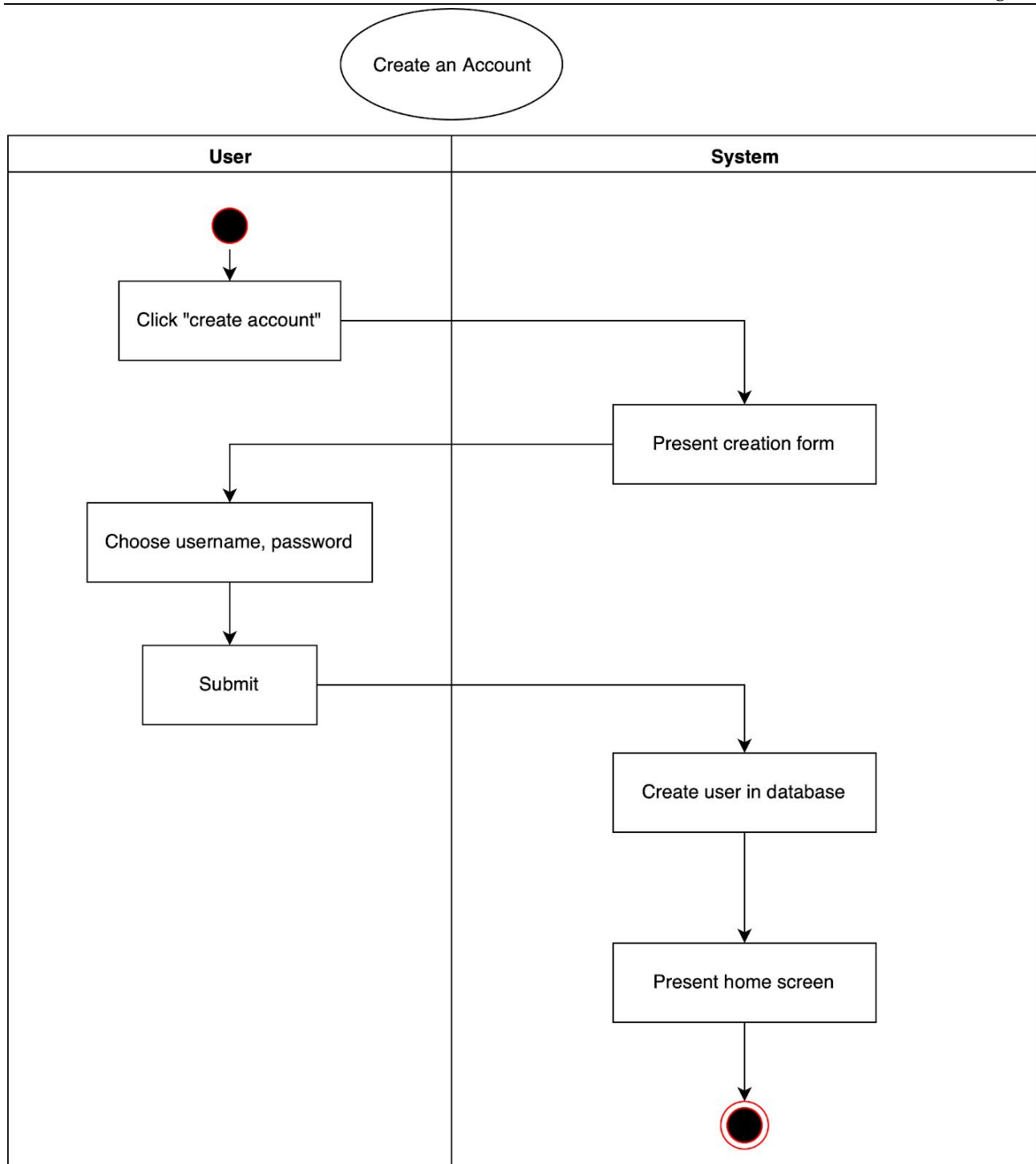
*"Creating an App," Meteor Blog RSS. [Online]. Available: <https://www.meteor.com/tutorials/react/creating-an-app>. [Accessed: 12-Dec-2019].*

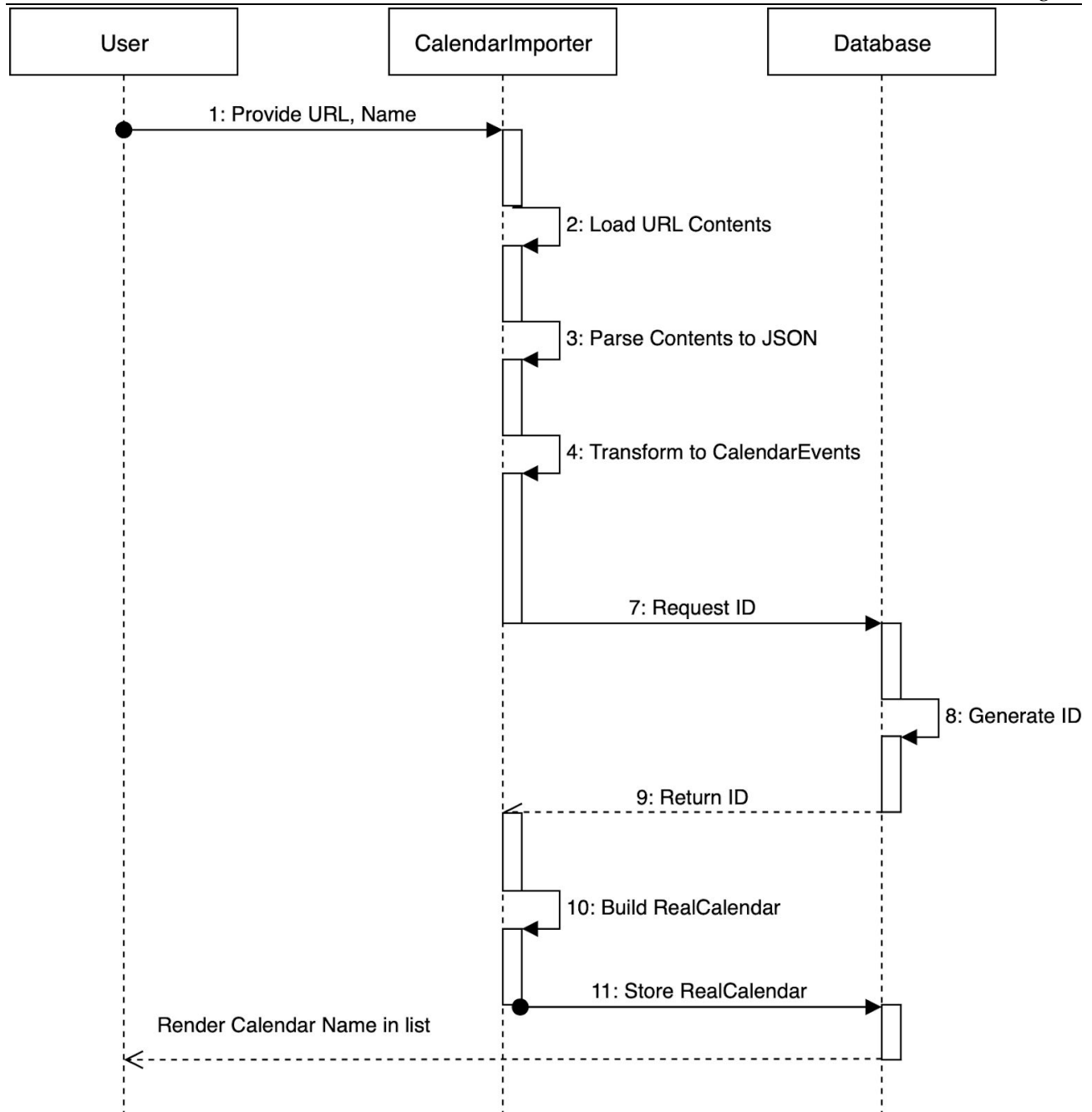
*E. B. Desruisseaux, "Internet Calendaring and Scheduling Core Object Specification (iCalendar)," IETF Tools, Sep-2009. [Online]. Available: <https://tools.ietf.org/html/rfc5545>. [Accessed: 12-Dec-2019].*

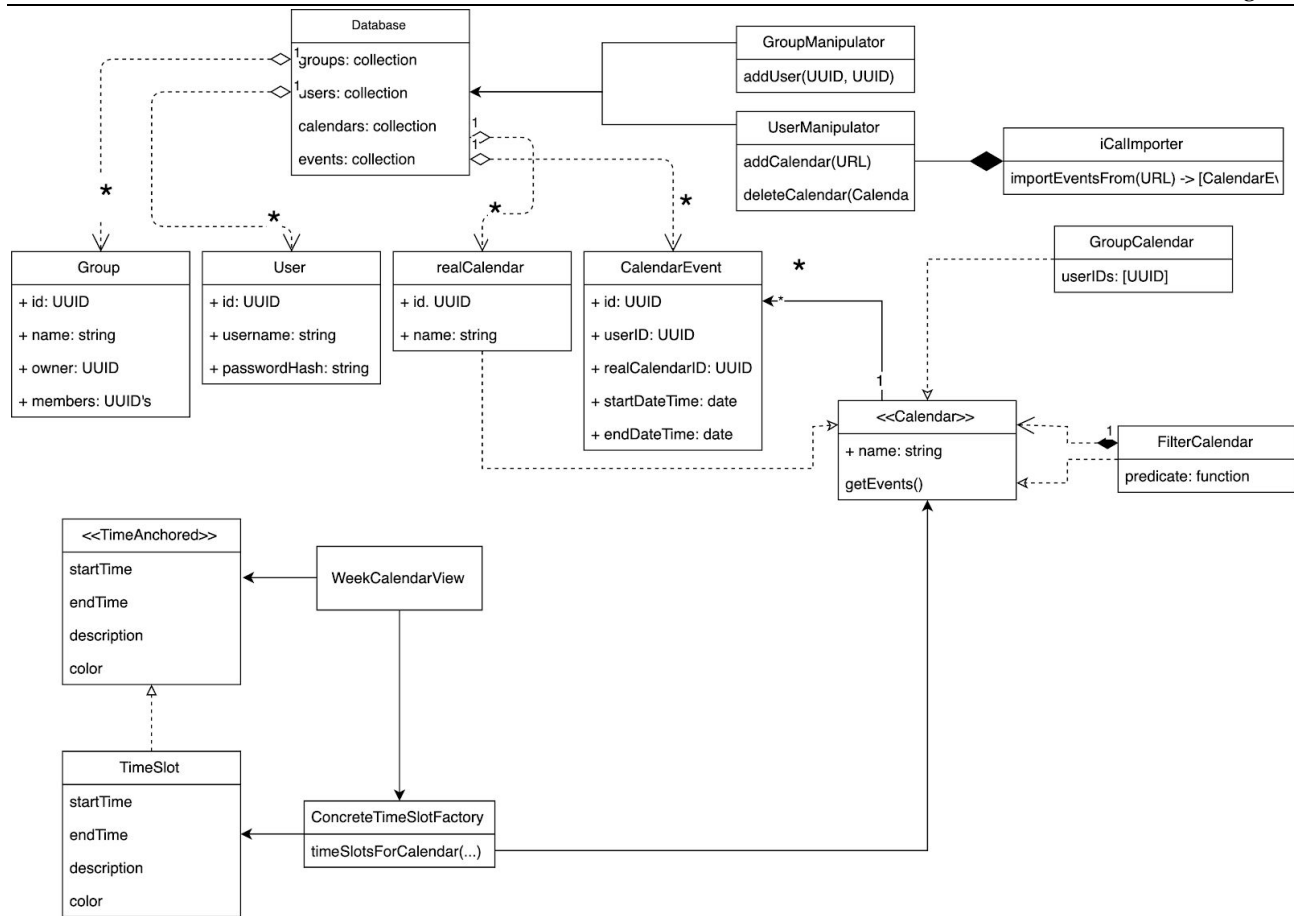
## 2 Design

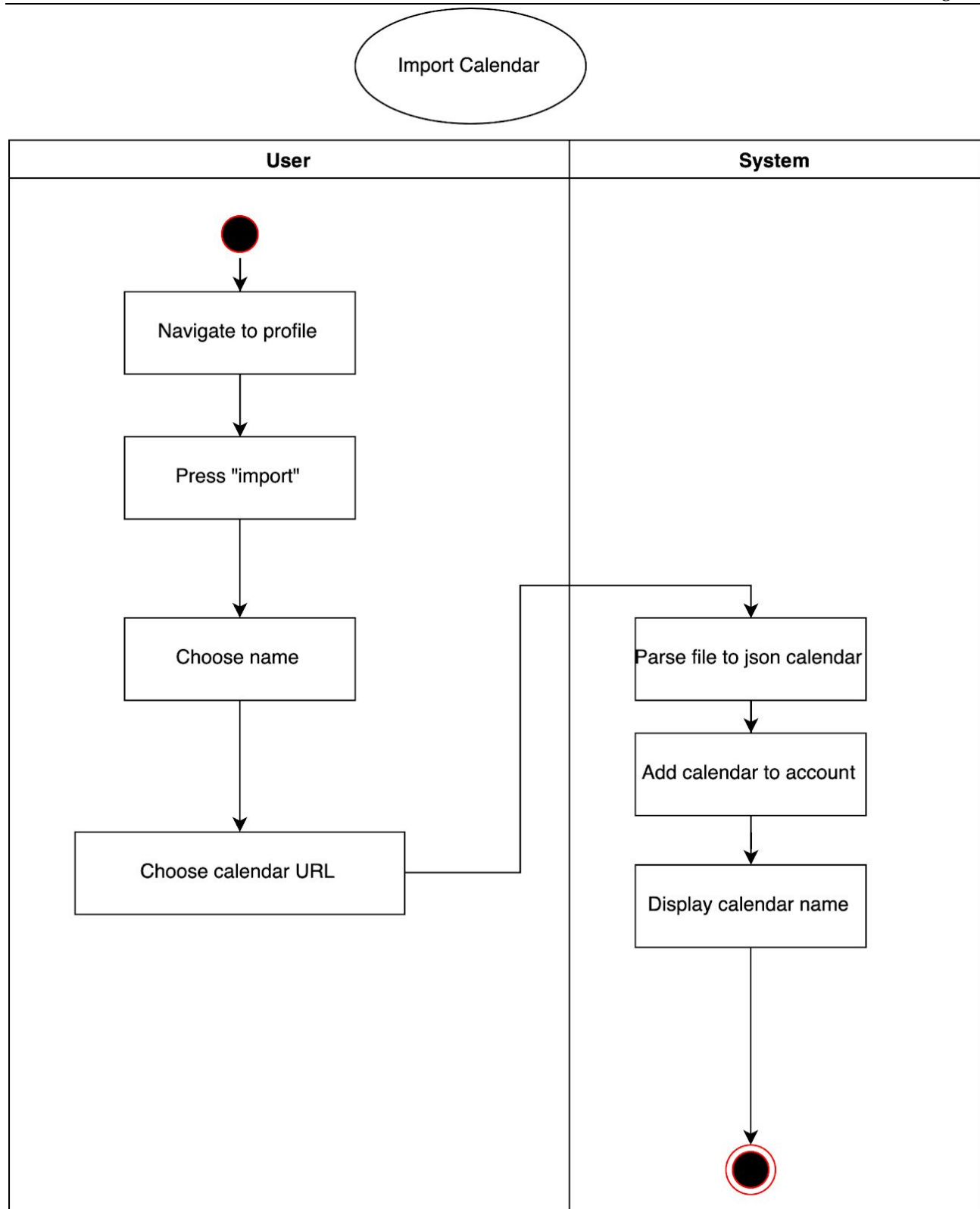
### 2.1 System Modeling



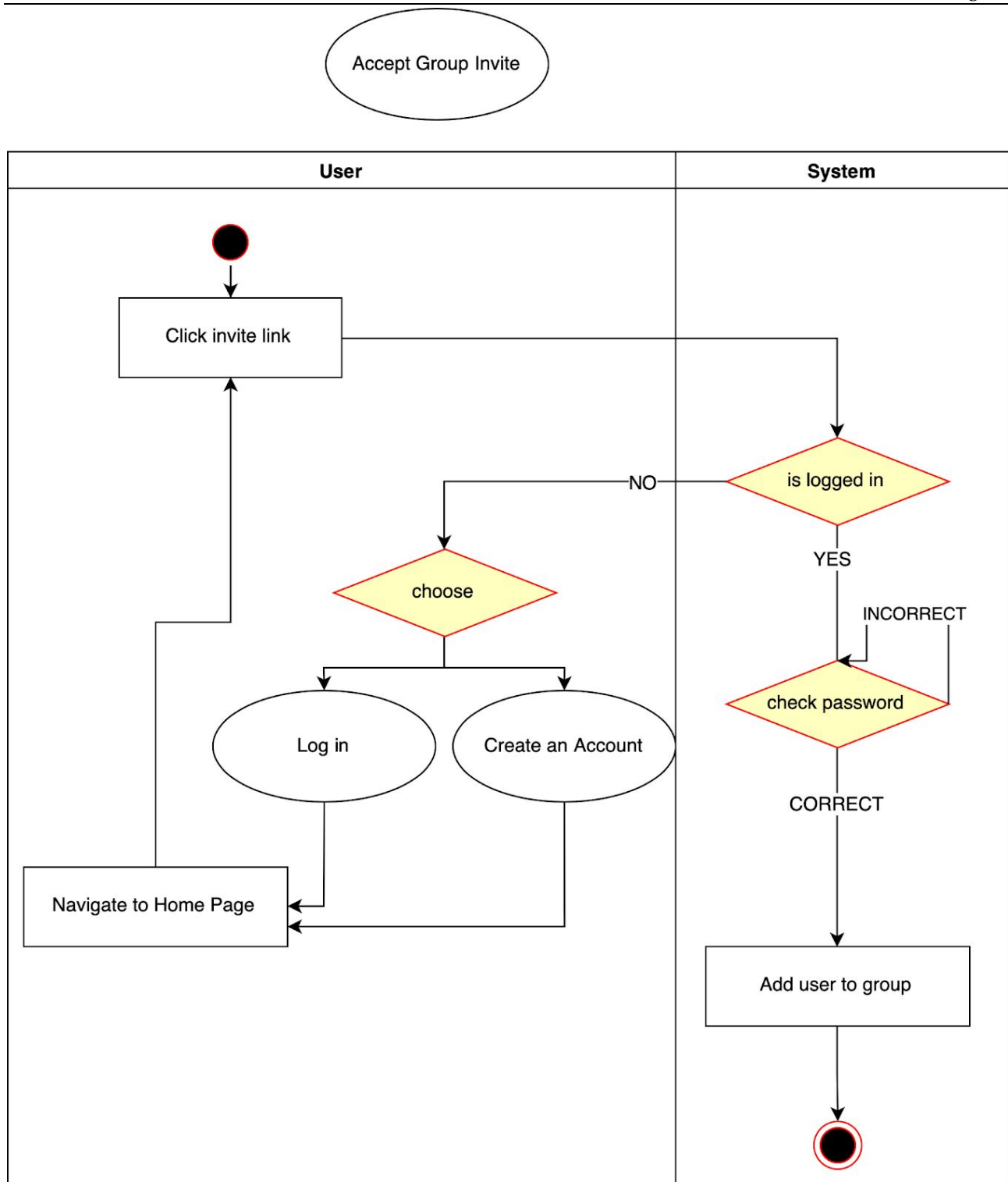


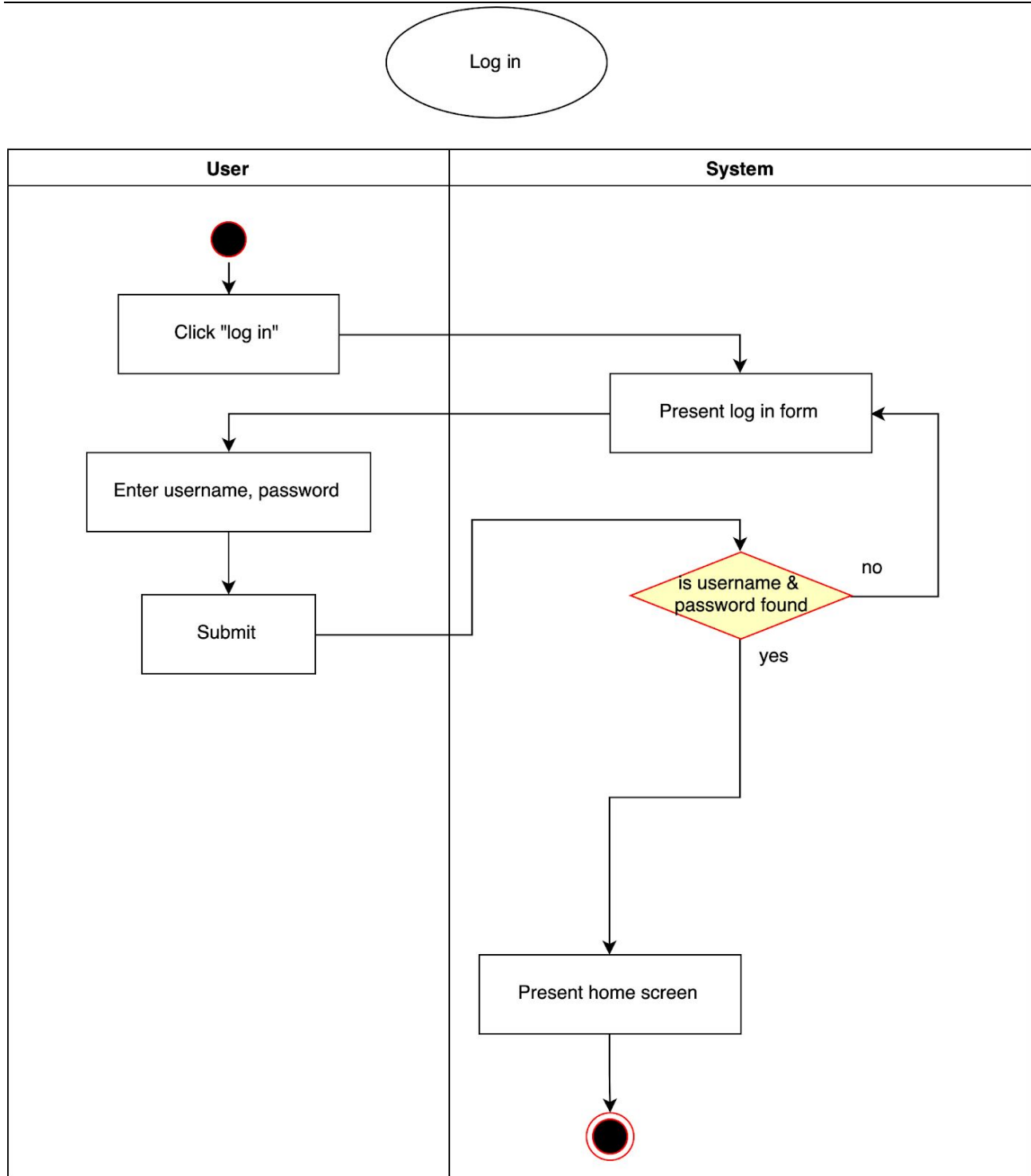






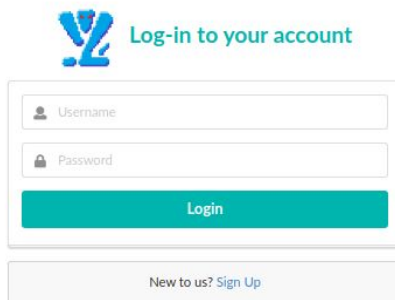






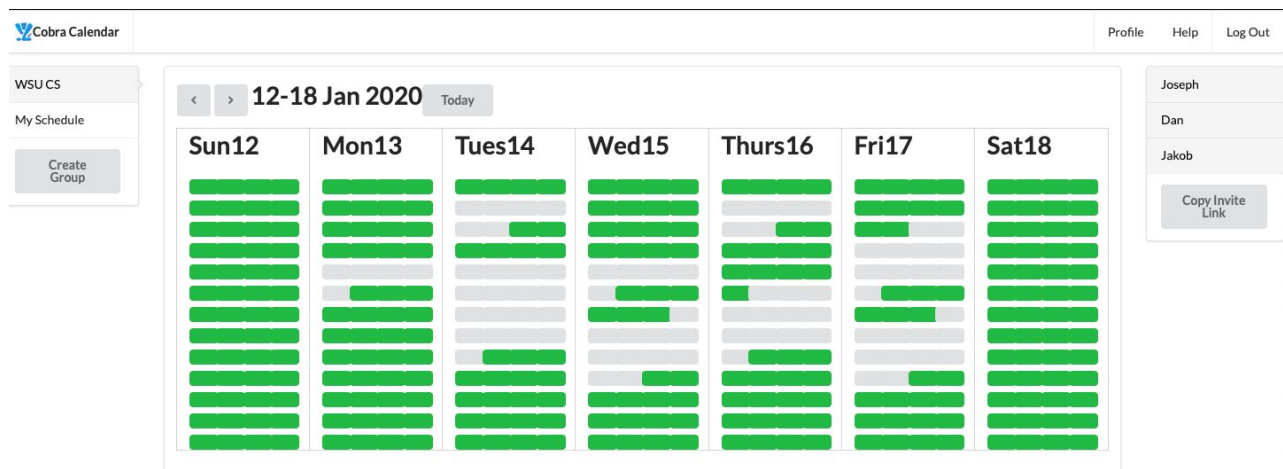
## 2.2 Interface Design

The log-in page allows a user to log in with an existing account, or easily switch to the sign-up page to create a new account. Logged-in users will be redirected to the home page. Users not logged in are always redirected to the log-in page.



The log-in form features a blue logo with a stylized 'Y' and the text 'Log-in to your account'. Below this is a white box containing two input fields: 'Username' with a person icon and 'Password' with a lock icon. A prominent blue 'Login' button is positioned below the fields. At the bottom of the box, a link reads 'New to us? Sign Up'.

The main home page displays the primary App interface. Along the top is the menu, which allows the user to switch between pages and log out. To the left lies the group menu; it shows the user all of their groups. Along the right edge of the screen is the members menu. the Group calendar is between them, displaying availability in green.



The main interface of the Cobra Calendar app. At the top, a header bar includes the 'Cobra Calendar' logo, navigation links for 'Profile', 'Help', and 'Log Out', and a user profile section showing 'Joseph', 'Dan', and 'Jakob' with a 'Copy Invite Link' button. The central area is a calendar grid for the week of January 12-18, 2020. The days are labeled: Sun12, Mon13, Tues14, Wed15, Thurs16, Fri17, and Sat18. The grid shows various green bars representing events or availability. On the left side, there is a sidebar with 'WSU CS' and 'My Schedule' sections, including a 'Create Group' button.

The profile screen allows the user to add and remove calendars to/from their profile, by providing an iCal URL.



A screenshot of the 'My Calendars' section. It shows a list of calendars. The first entry is 'Class Schedule', which has a plus sign icon to its right. Below it, there is a partially visible entry with an 'x' icon to its right, indicating a remove or delete action.

## 3 Implementation

### 3.1 Development Environment

Cobra Calendar was developed using a combination of Javascript, HTML, and CSS. It is built upon React and Meteor Frameworks, and developed using the IntelliJ IDE.

### 3.2 Task Distribution

For this project, Joseph set up the frameworks and the database, while Jakob implemented the URL parsing. The core functionalities and UI were developed using pair programming, by both Joseph and Jakob simultaneously.

### 3.3 Challenges

One of the main challenges in developing the application was using the React and Meteor Framework. Separating the server from the client was a new approach for us, and required us to change the way we thought about how we were programming. We dealt with these challenges by following the many tutorials provided by Meteor as well as other experienced users online.

Another piece of the application that gave us trouble was in parsing the ICal URL files. The ICal handling package we worked with had very little explanation and documentation on how to use it, and provided only a few small examples. Getting it to work was mostly a case of trial and error, starting with a small piece that we knew worked and expanding on it slowly until we got what we needed from it.

The reason these two parts were likely the most difficult is do to them being the components we worked on individually. The parts where we worked together using the pair programming approach went by much smoother, with each issue that came up being solved with relative ease.

## 4 Testing

### 4.1 Testing Plan

*We decided to focus on unit tests as to ensure we handle the date logic correctly. The functional requirements were mostly tested manually. In the future it would be wise to implement some integration tests for the functional requirements that are mostly concerned with the database.*

### 4.2 Tests for Functional Requirements

#### 4.2.1 User Management System

4.2.1.1 *The system shall allow non-Users to create accounts using a username and password.*  
**This functionality was verified manually - PASS**

4.2.1.2 *The system shall allow Users to log in with their username and password*  
**This functionality was verified manually - PASS**

4.2.1.3 *The system should allow Users to change their password*  
**This functionality was not implemented. It would be tested manually.**

4.2.1.4 *The system should allow Users to change their username*  
**This functionality was not implemented. It would be tested manually.**

4.2.1.5 *The system should allow Users to delete their account*  
**This functionality was not implemented. It would be tested manually.**

#### 4.2.2 Calendar Import System

4.2.2.1 *The system shall support importing calendars from Apple's implementation of the iCalendar format.*  
**The parsing logic was tested with unit tests. The tests we wrote passed, but did not cover recurring events and some other edge cases. The network portion was tested in a system-level manual test.**

4.2.2.2 *The system shall support importing calendars from Google's implementation of the iCalendar format.*  
**The parsing logic was tested with unit tests. The tests we wrote passed, but did not cover recurring events and some other edge cases. The network portion was tested in a system-level manual test.**

4.2.2.3 *The system should support importing calendars from arbitrary implementations of the iCalendar format.*  
**The parsing logic was tested with unit tests. The tests we wrote passed, but did not cover recurring events and some other edge cases. The network portion was tested in a system-level manual test.**

- 
- 4.2.2.4 *The system shall list the name and URL of the User's previously imported calendars*  
**This was tested manually, although we decided not to display the URL. It worked without issue.**
- 4.2.2.5 *The system shall allow the User to remove a calendar from their profile*  
**This was tested manually and it works as expected. Though we haven't yet implemented a system to delete the events themselves from our database so that would build up over time.**
- 4.2.2.6 *The system should allow the User to manually refresh their calendars.*  
**This functionality was not implemented. It would be tested manually or with database level integration tests.**
- 4.2.2.7 *The system will allow refreshing of calendars automatically as changes become available*  
**This functionality was not implemented. It would be tested manually or with database level integration tests.**
- 4.2.3 The Groups System**
- 4.2.3.1 *The system shall allow Users to create Groups with a name and password.*  
**This was tested manually and it works as expected.**
- 4.2.3.2 *The system shall allow Group owners to issue invitations by a link.*  
**This was tested manually and it works to a limited extent. The link is not a full URL, but rather a path relative to the host.**
- 4.2.3.3 *Non-Users who click an invite link will be directed to the registration page.*  
**This was tested manually and it works as expected.**
- 4.2.3.4 *After registering for an account through an invitation link, the system shall add the User to the Group.*  
**This functionality was not implemented. It would be tested manually.**
- 4.2.3.5 *The system shall allow Users to join Groups by clicking an invitation link.*  
**This functionality was implemented in such a way that after using the link they still needed to enter the password. It would be tested manually. It works as implemented.**
- 4.2.3.6 *The system shall allow Group owners to remove members from the Group.*  
**This functionality was not implemented. It would be tested manually.**
- 4.2.3.7 *The system should allow Group owners to change the Group name and password.*  
**This functionality was not implemented. It would be tested manually.**
- 4.2.3.8 *The system shall allow Group owners to delete a Group.*  
**This functionality was not implemented. It would be tested manually.**

4.2.3.9 *The system will allow Group owners to leave a Group.*

**This functionality was not implemented. It would be tested manually.**

4.2.3.10 *The system shall not allow Group members to remove other members.*

**No special functionality was required. This requirement is met.**

4.2.3.11 *The system shall allow Group members to leave a Group.*

**This functionality was not implemented. It would be tested manually.**

#### **4.2.4 The Calendar System**

4.2.4.1 *The system shall allow the User to view time blocks that are available on a week view.*

**This functionality was tested thoroughly through many unit tests and manual testing.**

4.2.4.2 *When in week mode, the system shall allow the User to change the selected week forward or backward.*

**This functionality was tested manually and works as expected. It was designed in such a way that writing unit tests in the future would be trivial.**

4.2.4.3 *The system shall allow the User to view time blocks that are available on a month view.*

**This functionality was not implemented. But much work was done to abstract our implementation of the week view so this view would be easy to implement. It would be tested manually and with unit tests on the various date functions it requires.**

4.2.4.4 *When in month mode, the system shall allow the User to change the selected month forward or backward.*

**This functionality was not implemented. It would be tested manually and with unit tests.**

4.2.4.5 *The system shall allow the User to view time blocks that are available by selecting specific Users.*

**This functionality was tested manually. It works as expected.**

4.2.4.6 *The system will allow the User to view an aggregate set of time blocks from a selection Users and calendars.*

**This functionality was tested manually and the implementation was done in such a way to enable unit and integration tests in the future.**

4.2.4.7 *The system should treat events that are not marked 'busy' as nonexistent and should consider them available.*

**This functionality was not implemented but would be tested with unit tests for our parse function.**

### **4.3 Tests for Non-functional Requirements**

#### **4.3.1 Performance Requirements**

4.3.1.1 *Any action should not take more than 10 seconds to perform, assuming User has a healthy internet connection.*

**PASS, no operation took longer than about a second. Tested manually. Repeatedly pressing the week forward button would make the UI lag a bit while catching up.**

4.3.1.2 *The system should clearly identify when it is loading or processing information with a loading indicator.*

**The application is very responsive and doesn't have long running tasks. So no loading indicator would not be necessary.**

4.3.1.3 *The system shall run without issue on Washington State University Vancouver lab computers.*

**The system was tested both manually and by running the automated test suite on the lab machines. It worked without issue.**

#### **4.3.2 Safety and Security Requirements**

4.3.2.1 *The system shall not access a User's calendar event names or descriptions, only the dates and times of individual events.*

**This was verified through inspection of the database schema and the implementation of the parser.**

4.3.2.2 *The system shall not make any changes to a User's original calendar import, only copy relevant information from the file into a simplified form.*

**This was verified through inspection of the implementation of the parser. Not only that most iCal links are read-only.**

4.3.2.3 *The system shall not store a User's calendar file in any way, and shall keep it only until the simplified file is created.*

**This was verified through inspection of the implementation of the parser.**

## **4.4 Hardware and Software Requirements**

*Requires suitable memory on the host machine. We experienced issues running the server on a laptop with 2GB of memory.*

# **5 Analysis**

In drafting the SRS document, we worked together for approximately 10 hours, and individually for approximately an hour and a half each. The design document took us a little less time than the SRS, and we wrote it entirely together. It took us approximately 8 hours to complete. The milestone that took us the most time by far was the actual implementation and final report for milestone 3. In



setting up the frameworks and database, Joseph put in approximately 10 hours of work. To parse the iCal URL, Jakob worked on the project for approximately 8 hours. The remainder of the application and report were developed simultaneously using pair programming. Implementing the UI, testing, and drafting the final report took the two of us approximately 30 hours to complete.

Milestone 3 took a significantly larger amount of time and effort compared to the previous milestones. This is the case because in our requirements and designing phase, we were coming up with ideas for what we wanted to do and where we wanted to go with our project that we didn't know how to do at the time. This project was a learning experience for both of us, and nearly every part of our application that we implemented required us to learn how to do it before we could proceed. We also chose to do an application that had a large amount of features that took time to implement but not necessarily any time to come up with.

## 6 Conclusion

While working on this project, we learned how to develop software using the pair programming approach, and the benefits of doing so. By working together on the same workstation, we were able to focus our attention to both the small-scale implementation and the bigger picture system interaction simultaneously. We also had two sets of eyes when debugging, and were even able to catch errors before they happened. Overall, we learned that working together seemed to take us less time than working separately, and our code was better for it as well.

We learned how to set up Meteor and React frameworks in order to create a web application that worked like a native application, seamlessly communicating between the client and server without having to reload the page after each action.

We learned how to engineer software from the ground up correctly using requirements engineering and use case modeling, ensuring the code we developed worked together correctly. Had we just jumped into the project head first and started programming immediately, we would likely have ran into compatibility issues and had to go back and redo many hours of work.

## **Appendix A - Group Log**

For this project, we met up 2-3 times weekly. Every Tuesday and Thursday after class, and on Fridays as well when deadlines drew near. Our team communication was very efficient and clear. We set up a group Slack channel for messaging and sharing resources, and linked our github project to the channel so our commits were pushed as notifications. This kept both of us up to date on the progress we were making when doing independant development.