

**Spring 2021**

# Senior Project Report

Works for Me (*Web Application*)

**Computer Science Department**

**California State University, Dominguez Hills**

# Table of Contents

<b>Section 1: Introduction.....</b>	<b>5</b>
Background and Significance .....	5
Description .....	6
Target Audience .....	7
<b>Section 2: Prior Related Work.....</b>	<b>8</b>
Prior work by others .....	8
My prior work .....	8
Interval Scheduling and Partitioning Project: .....	9
<b>Section 3: Resources and References .....</b>	<b>10</b>
Resources .....	10
References .....	11
<b>Section 4: Design and Analysis.....</b>	<b>12</b>
Functional Requirements.....	12
Non-Functional Requirements.....	12
App Use Case Diagram: .....	13
Use Cases .....	14
1. Login.....	14
2. Check Scheduled Events .....	15
3. Add Event.....	16
4. Add Friend.....	17
5. Select Friend.....	18
Sequence Diagrams.....	19
1. Login.....	19
2. Check scheduled events.....	20
3. Add Event.....	21
4. Add Friend.....	22
5. Select Friend.....	23

Class Diagram .....	24
<b>Section 5: Implementation and Evaluation.....</b>	<b>25</b>
Components.....	26
Check Scheduled Events .....	26
Add an Event .....	27
Add Friend .....	28
List of Friends .....	29
Delete Friends .....	30
Challenges faced .....	31
Notable Bugs .....	32
Back-End.....	33
Adding an Event .....	33
Adding a Friend .....	35
Connecting to the database.....	37
What I plan to do next .....	38
Evaluation .....	39
<b>Section 6: Source Code .....</b>	<b>40</b>
Components/AddEvent.jsx .....	40
Components/AddFriend.js .....	42
components/FindTime.js .....	44
Components/Hompage.js .....	48
Components/Calendar.jsx.....	51
Components/navbar.js.....	54
Components/Calendar.css .....	55
Components/styles.js.....	56
App.js .....	57
App.css .....	57
Index.js .....	58

Index.css.....	58
Server/models/event.model.js .....	59
Server/models/user.model.js .....	60
server/routes/event.js .....	60
server/routes/users.js.....	62
server/sever.js.....	62
Works-for-me/package.json .....	63

# Section 1: Introduction

## Background and Significance

Working as a team takes proper coordination to ensure that everything goes according to plan. However, one of the most common problems that many employers face when arranging a group meeting is gathering information about everyone's availability. It is simply one more task to deal with, and given that some people have very busy schedules, this step may be frustrating at times. In some cases, colleagues work from remote states/countries which can introduce different time zones into the mix of confusion. Scheduling meetings is usually too time consuming and is often handed to a separate employee to handle. This option is not available to everyone, but even those who do have this option will benefit from this product. This product will attempt to address this problem by taking multiple schedules and outputting different time intervals in which most people are available to attend.

This product is significant because it aims at simplifying this task while giving the user ease of mind that their meetings are scheduled the way they should be. Many times, when scheduling a group meeting the risk of people missing out on your meeting is present. To minimize this risk, this product will attempt to output a time frame which will result in the least number of missing attendees for the meeting.

## Description

The project will consist of a web application that hosts this service. This will include user profiles that save the user's personal schedule and security preferences. Every user will have the option to start a group and allow other users to join the group. The user who starts the group will be set as the group leader by default. The application will then ask the group leader to give more details about the task that they would like to schedule, information such as: Meeting Topic and description. The application will then take the schedules of the users within the group to compare. Using interval scheduling and interval partitioning this will output a specific time frame that is ideal for all members of the group. If this ideal time is not possible it will choose a time frame that results in the least number of attendees missing.

The application will have all the basic features that you would expect from any other schedule planner. The plan is to also add a feature that would let users add each other as friends. The user will be able to choose an individual friend from their friend list and toggle the privacy preferences. This will control how much they would like to share with each person on their friends list. If you add a colleague you may not want them to see everything you have planned for the day, but perhaps you would like that information to be visible to your spouse. Despite the privacy filter, there will be a feature to request access to a friend's schedule.

## Target Audience

This solution should be available to use within a professional environment as well as for personal use. The primary target audience for this product are users in a professional environment. Whether that be a supervisor managing a group of remote employees or a group of colleagues looking to collaborate on a project. This is especially useful for remote work that may include the possibility of being in different time zones.

The secondary target audience is for people who want to use it for personal use. As previously mentioned, this should still function like any other schedule planner. This can still be useful among a group of friends who would like to know when its best to go out for dinner or any other event. Those who want to use it as a planner will be able to see a glimpse of their day before it happens. This will be useful for people who choose not to use to use a calendar to stay organized because they want to spend time figuring out when they are free. When a user has a calendar full of events, they may find it overwhelming to see that they must figure out where they can fit another event. This tool aims to help this task.

## Section 2: Prior Related Work

### Prior work by others

This is a problem that has been addressed before. I will be attempting to recreate what already exists while adding more functionality. Tools such as Calendy, FindTime, and WhenAvailable are all tools that are intended to solve this exact problem. All these tools use their own approach to the issue. I plan to recreate what seems efficient and leave out features and options that can be improved on.

### My prior work

I have previously worked on a project that involved algorithms for interval scheduling and interval partitioning. This project dealt with comparing time intervals of courses to find the most compatible intervals that maximize the use of the classrooms. I believe I can apply some of those concepts to this project to find efficient time frames. I have previous experience with HTML and CSS from my Web Programming course to create a simple GUI for this web application. I have seen other developers work with React to create a user interface but I have not experimented with this at all.



## Interval Scheduling and Partitioning Project:

```
Enter the amount of intervals to be generated:
8
Enter start time:
10
Enter end time:
30
===== Initial Intervals =====

[0, [14, 17]]
[1, [24, 26]]
[2, [16, 22]]
[3, [13, 22]]
[4, [10, 11]]
[5, [17, 20]]
[6, [20, 21]]
[7, [21, 26]]

===== Initial Scheduling =====

Class [id=0, intervals=[[4, [10, 11]], [0, [14, 17]], [5, [17, 20]], [6, [20, 21]], [1, [24, 26]]]]

===== Initial Partitioning =====

Class [id=0, intervals=[[4, [10, 11]], [3, [13, 22]], [1, [24, 26]]]]
Class [id=1, intervals=[[0, [14, 17]], [6, [20, 21]]]]
Class [id=2, intervals=[[2, [16, 22]]]]
Class [id=3, intervals=[[5, [17, 20]], [7, [21, 26]]]]
```

```
Enter the amount of intervals to be generated:
7
Enter start time:
7
Enter end time:
19
===== Initial Intervals =====

[0, [11, 15]]
[1, [8, 16]]
[2, [8, 11]]
[3, [8, 15]]
[4, [7, 8]]
[5, [7, 15]]
[6, [11, 12]]

===== Initial Scheduling =====

Class [id=0, intervals=[[4, [7, 8]], [2, [8, 11]], [6, [11, 12]]]]

===== Initial Partitioning =====

Class [id=0, intervals=[[4, [7, 8]], [0, [11, 15]]]]
Class [id=1, intervals=[[5, [7, 15]]]]
Class [id=2, intervals=[[2, [8, 11]]]]
Class [id=3, intervals=[[1, [8, 16]]]]
Class [id=4, intervals=[[3, [8, 15]]]]
Class [id=5, intervals=[[6, [11, 12]]]]
|
```

```

Enter the amount of intervals to be generated:
5
Enter start time:
1
Enter end time:
20
===== Initial Intervals =====

[0, [1, 4]]
[1, [6, 7]]
[2, [8, 11]]
[3, [1, 11]]
[4, [1, 3]]

===== Initial Scheduling =====

Class [id=0, intervals=[[4, [1, 3]], [1, [6, 7]], [2, [8, 11]]]]

===== Initial Partitioning =====

Class [id=0, intervals=[[3, [1, 11]]]]
Class [id=1, intervals=[[0, [1, 4]], [1, [6, 7]], [2, [8, 11]]]]
Class [id=2, intervals=[[4, [1, 3]]]]

```

## Section 3: Resources and References

### Resources

Some of the resources that I will utilize include:

The M.E.R.N stack:

- **mongoDB** – I will use this database program for my web application.
- **Express JS** – This back-end web application framework will work well with Node.js.
- **React JS** – This will be used for my front-end web framework.
- **NodeJS** – I will use Node.js for back-end runtime environment. This is open source and cross-platform.

For styling I used material-ui, which is a React UI framework. I used elements such as Paper, Button, Icons, and Typography to make my components more user friendly.

## References

Angell, Chris. "11 Meeting Scheduler Apps to Boost Your Productivity This Year." *Lifehack*, Lifehack, 14 Dec. 2020, [www.lifehack.org/801699/meeting-scheduler](http://www.lifehack.org/801699/meeting-scheduler).

"Building Web Apps with WordPress." *O'Reilly Online Learning*, O'Reilly Media, Inc., [www.oreilly.com/library/view/building-web-apps/9781449364779/ch01.html](http://www.oreilly.com/library/view/building-web-apps/9781449364779/ch01.html).

Goyal, Arun. "Best Web Development Stacks 2021: Popular Development Tech Stacks." *Octal IT Solution*, [www.octalsoftware.com/blog/best-web-development-stacks](http://www.octalsoftware.com/blog/best-web-development-stacks).

Mashaal, Stuart. "Interval Scheduling." *Stuart Mashaal*, [stumash.github.io/Algorithm\\_Notes/greedy/intervals/intervals.html](http://stumash.github.io/Algorithm_Notes/greedy/intervals/intervals.html).

"Documentation." FullCalendar, [fullcalendar.io/docs](http://fullcalendar.io/docs).

Mikoo, director. FullCalendar Tutorial for Beginners. YouTube, 29 Sept. 2020, [youtu.be/Q8NV4koY7nU](https://youtu.be/Q8NV4koY7nU).

"How To Use The Google Calendar API In Node.js." YouTube, The Life of a Dev, 5 Mar. 2020, [youtu.be/zrLf4KMs71E](https://youtu.be/zrLf4KMs71E).

## Section 4: Design and Analysis

### Functional Requirements

- The user should be able to use this application as a stand-alone planner for their events.
- The user should be able to change privacy preferences for each individual friend.
- The program should be able to compare several schedules and return available times that work for all schedules.
- The program should be able to take a specific time frame entered by the user to analyze free time in schedules.
- The program should be able to take a preferred day from the user when proposing an event to a friend.
- If the program is unable to find free time within the specified time and day, then it should say so to the user.

### Non-Functional Requirements

- The design of the application must be intuitive and clear to avoid confusing the user.
- The user should be able to customize how they their events are displayed, whether it be daily/weekly/monthly.
- The prompts given to the user must be clear so that they know when an event is scheduled or not.

## App Use Case Diagram:

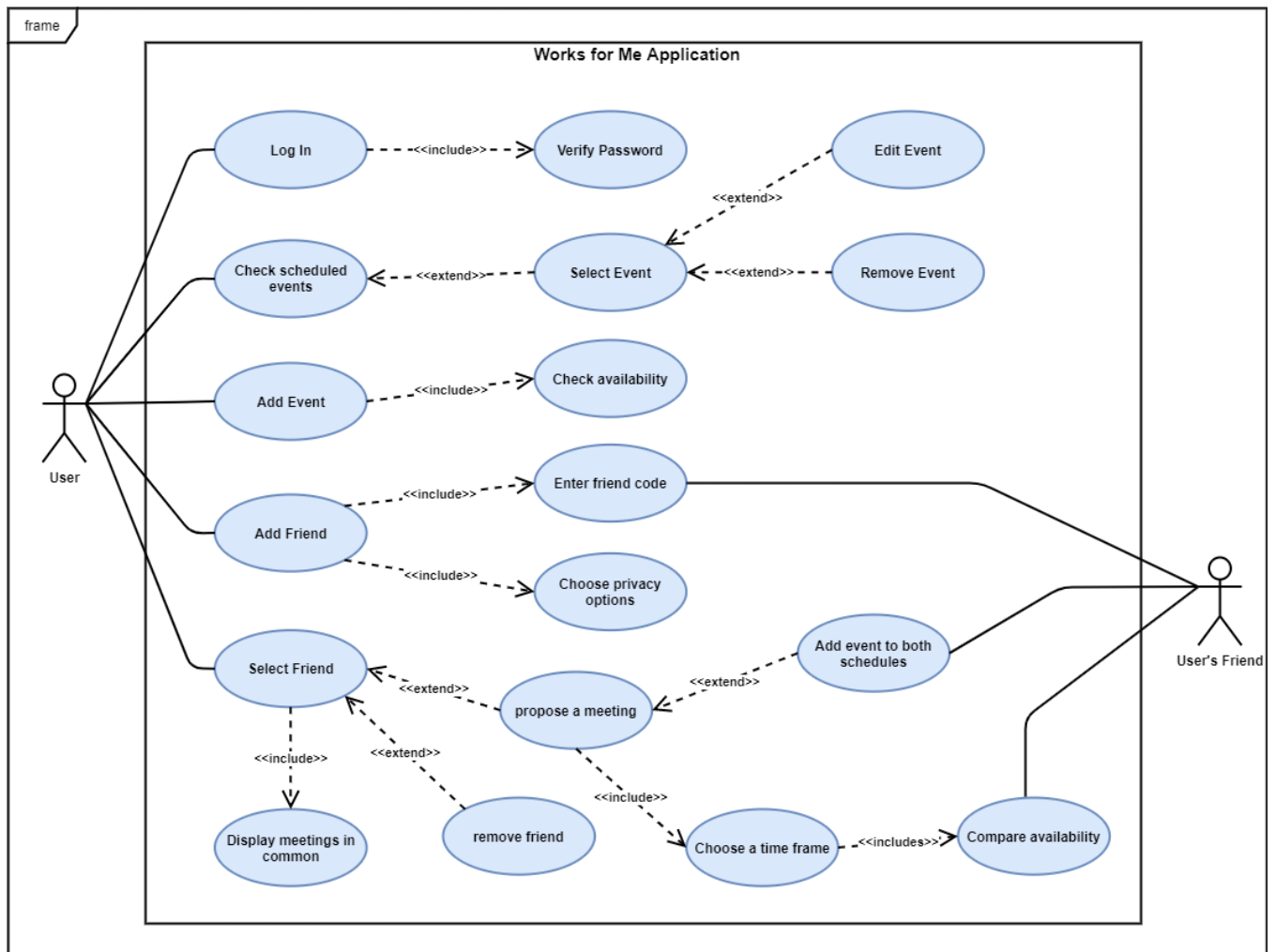


Figure 1: Use Case Model

## Use Cases

### 1. Login

<i>Use case name</i>	<b>Login</b>
<i>Participating Actors</i>	<b>User</b>
<i>Flow of events</i>	<ol style="list-style-type: none"><li>1. To access their account, the program requests a username and password from the user.</li><li>2. The user inputs their login information.</li><li>3. The program verifies the login information. If the information matches the one on record, then the user is granted access to their account. If the login information is incorrect it denies them entry and allows the user to try again.</li></ol>
<i>Entry condition</i>	The user enters their login information and clicks “sign in”
<i>Exit condition</i>	<p>If the user successfully logs in, they will get access to their account and this process will be completed.</p> <p style="text-align: center;">OR</p> <p>If the process is unsuccessful then, the program will ask the user to input the correct information. If they choose to do so, the entire process will repeat itself.</p>

## 2. Check Scheduled Events

<i>Use case name</i>	<b>Check scheduled events</b>
<i>Participating Actors</i>	<b>User</b>
<i>Flow of events</i>	<ol style="list-style-type: none"><li>1. After logging in, the user has the option of looking at their calendar to get an overview of their scheduled events.</li><li>2. From this view, the user has the option to select one of their scheduled events.</li><li>3. After selecting a specific event, the user has an option to edit or remove this event from their schedule.</li></ol>
<i>Entry condition</i>	This is initiated by a user when trying to look at their daily/weekly/monthly scheduled events.
<i>Exit condition</i>	<p>The event is over if the user finishes editing an event or by removing it.</p> <p>OR</p> <p>When the user closes their calendar and returns to the homepage.</p>

### 3. Add Event

<i>Use case name</i>	<b>Add Event</b>
<i>Participating Actors</i>	<b>User</b>
<i>Flow of events</i>	<ol style="list-style-type: none"><li>1. The user begins by clicking “add event”.</li><li>2. They are prompted to enter an event name and time frame.</li><li>3. The system checks to see if that time frame is available. If an event is already scheduled for that time frame, it will let the user know. It will fail to add the event.</li><li>4. If the time frame is available, it will add the event and let the user know that the event was successfully added.</li></ol>
<i>Entry condition</i>	This process begins when a user attempts to create an event by clicking “add event” from the homepage.
<i>Exit condition</i>	The process ends when the event succeeds or fails at adding the new event.



#### 4. Add Friend

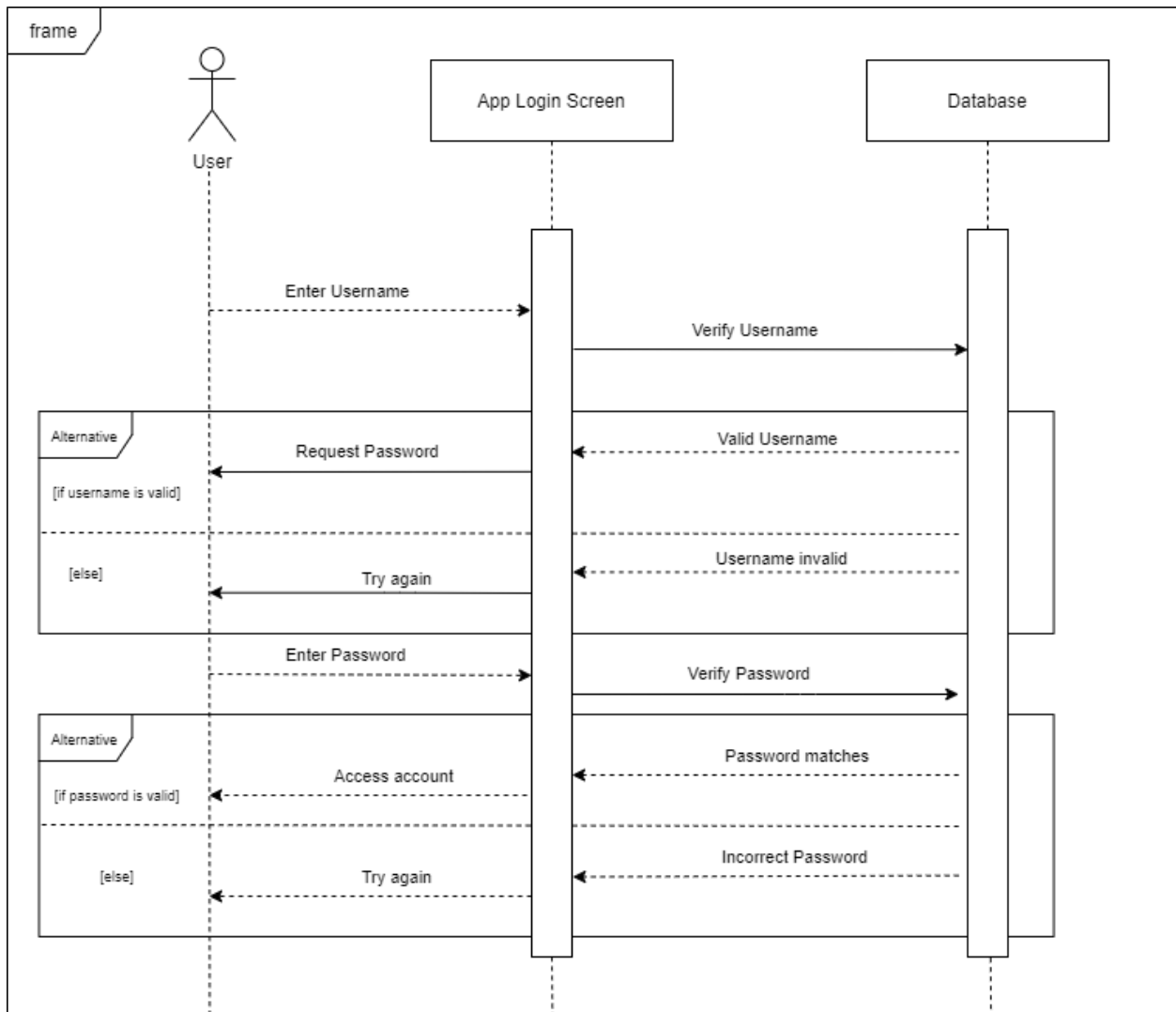
<i>Use case name</i>	<b>Add Friend</b>
<i>Participating Actors</i>	<b>User</b>
<i>Flow of events</i>	<ol style="list-style-type: none"><li>1. The user begins by clicking “add friend” from the homepage.</li><li>2. The user can then add their friend by using the unique “friend code” that each use is given. After the user’s friend provides the friend code, the friend request can be sent out.</li><li>3. The user is immediately asked to choose a level of privacy for that specific friend. This privacy is defined as certain predefined levels of transparency. The user can set a strict privacy not allowing the friend to see their availability. If the user selects moderate level of privacy the friend may be able to see when they have free time.</li></ol>
<i>Entry condition</i>	This is initiated by a user when they click “add friend” and add their information.
<i>Exit condition</i>	This process ends when the friend request is sent out.

## 5. Select Friend

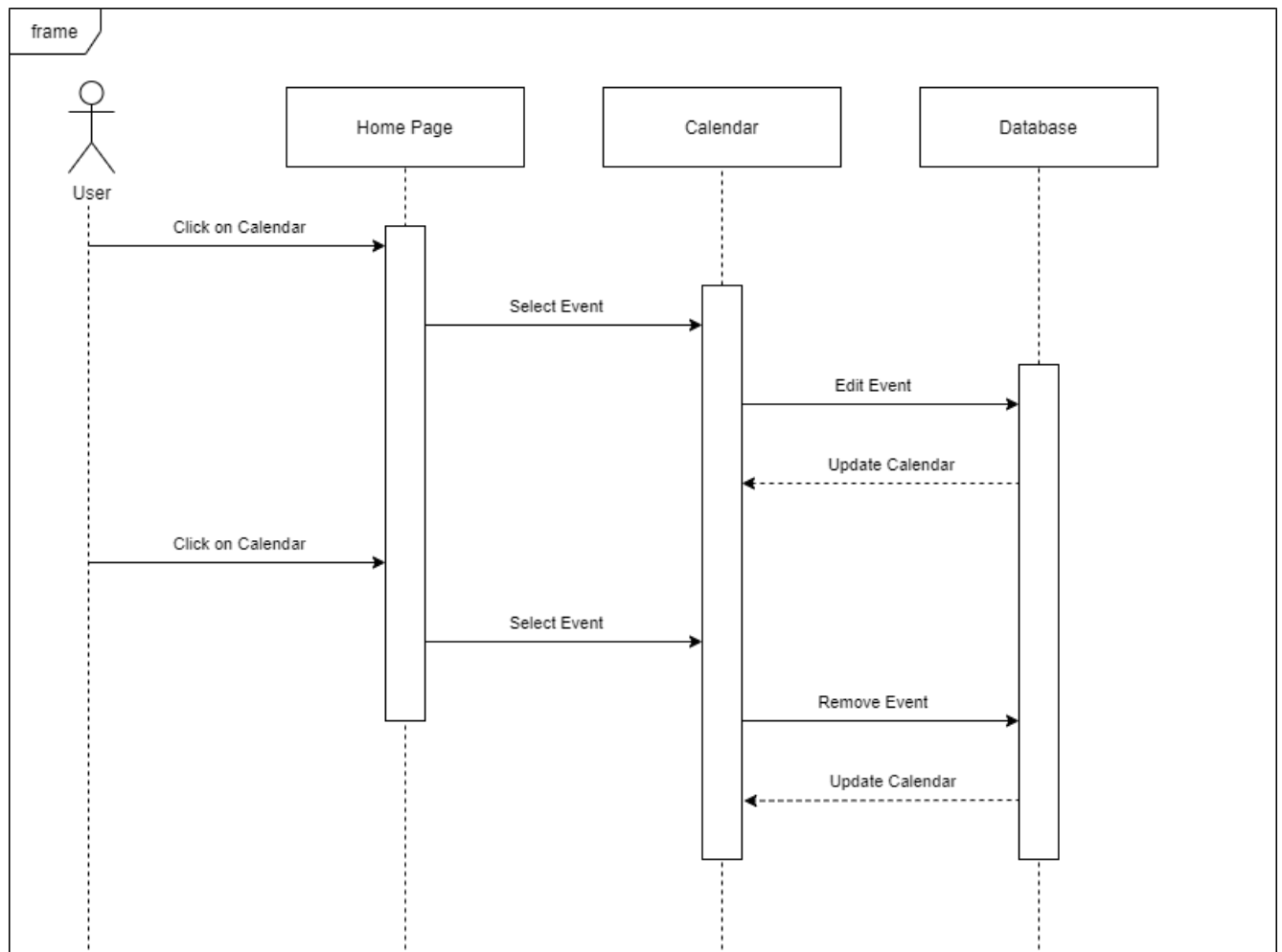
<i>Use case name</i>	<b>Select Friend</b>
<i>Participating Actors</i>	User, User's friend
<i>Flow of events</i>	<ol style="list-style-type: none"><li>1. If the user selects one of their friends from the friend list on the homepage, they get an overview of meetings that they have in common with their friends.</li><li>2. They also have an option to propose a meeting with that friend. After filling in some of the details for the event the user must pick a time frame that they would prefer.</li><li>3. This time frame is considered when comparing schedules between both users. If the system finds that this time does not work for both users, then it will let the users know.</li><li>4. If the time frame is available and the friend accepts the proposal, then the event will be added to both schedules.</li></ol>
<i>Entry condition</i>	The user must choose one of their friends
<i>Exit condition</i>	The process ends if user clicks away from their friend or if they successfully schedule a meeting with this friend.

## Sequence Diagrams

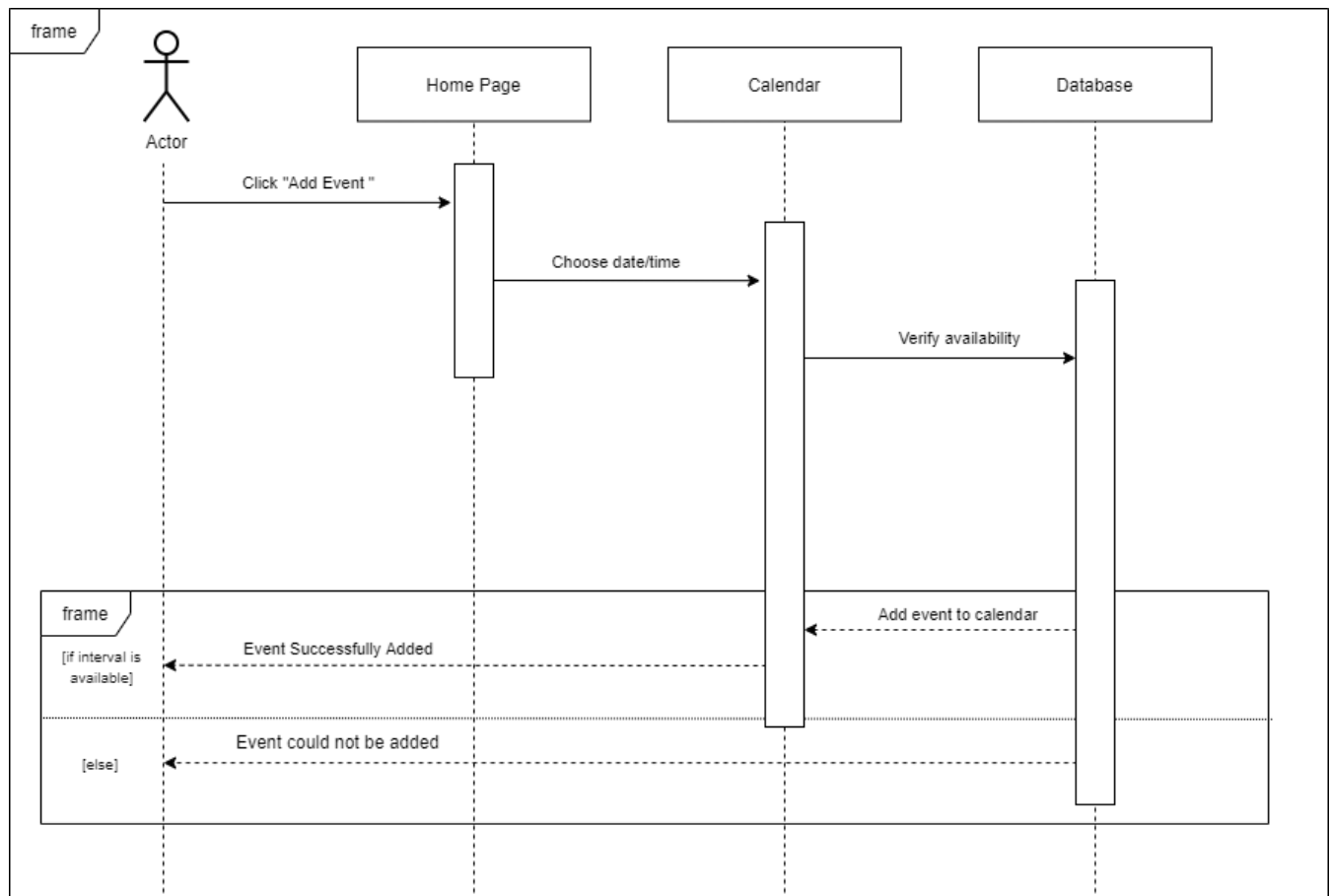
### 1. Login



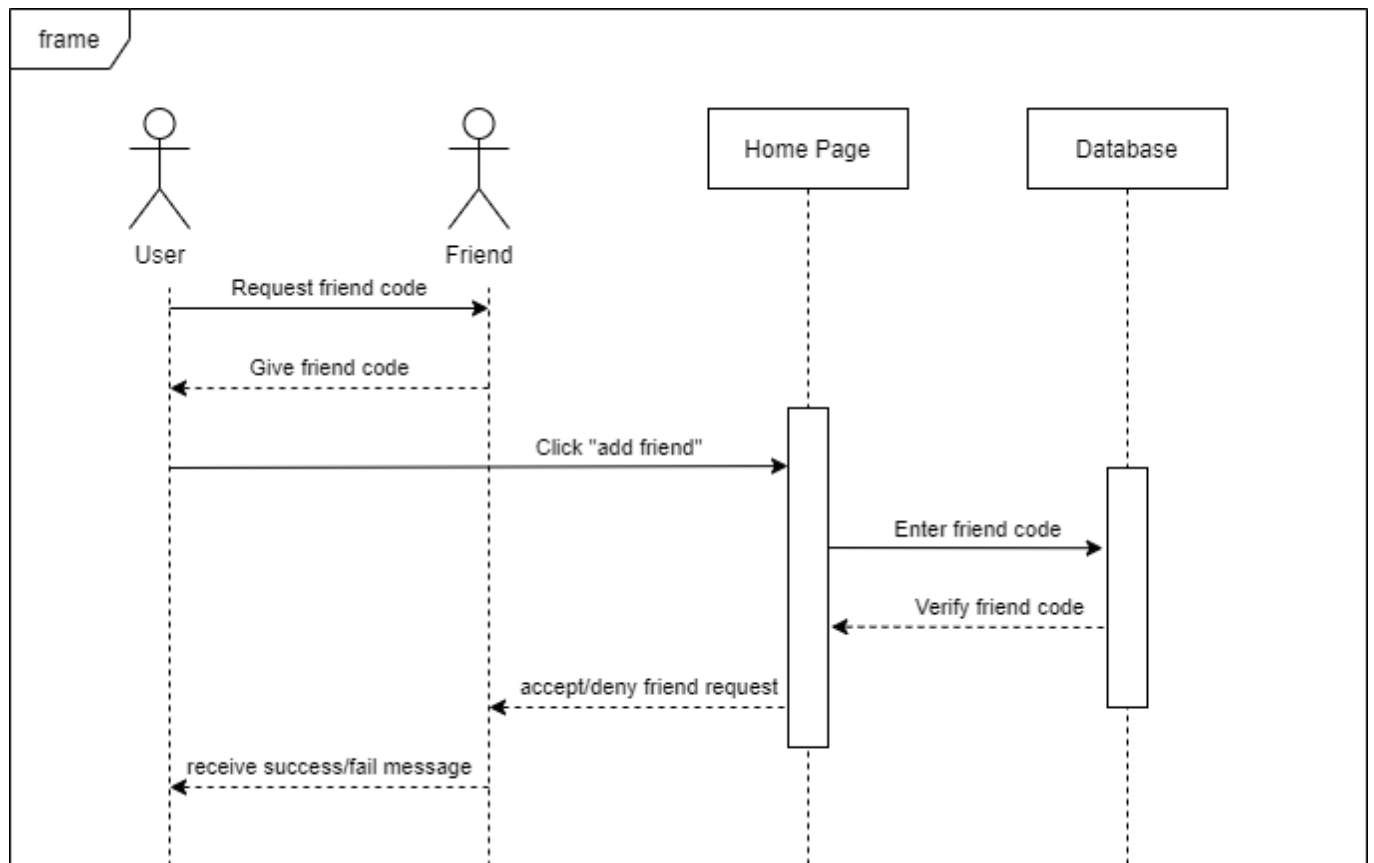
## 2. Check scheduled events



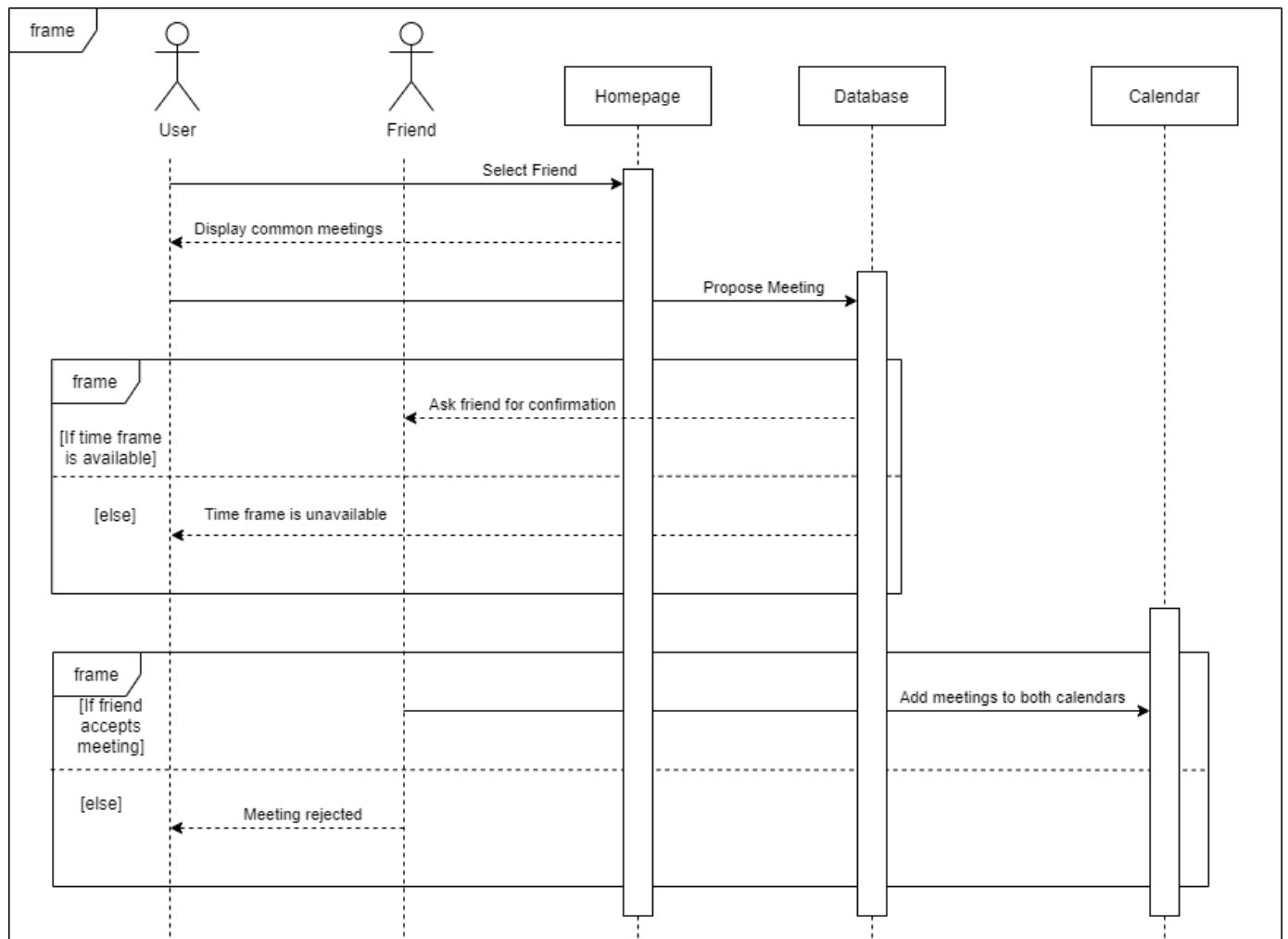
### 3. Add Event



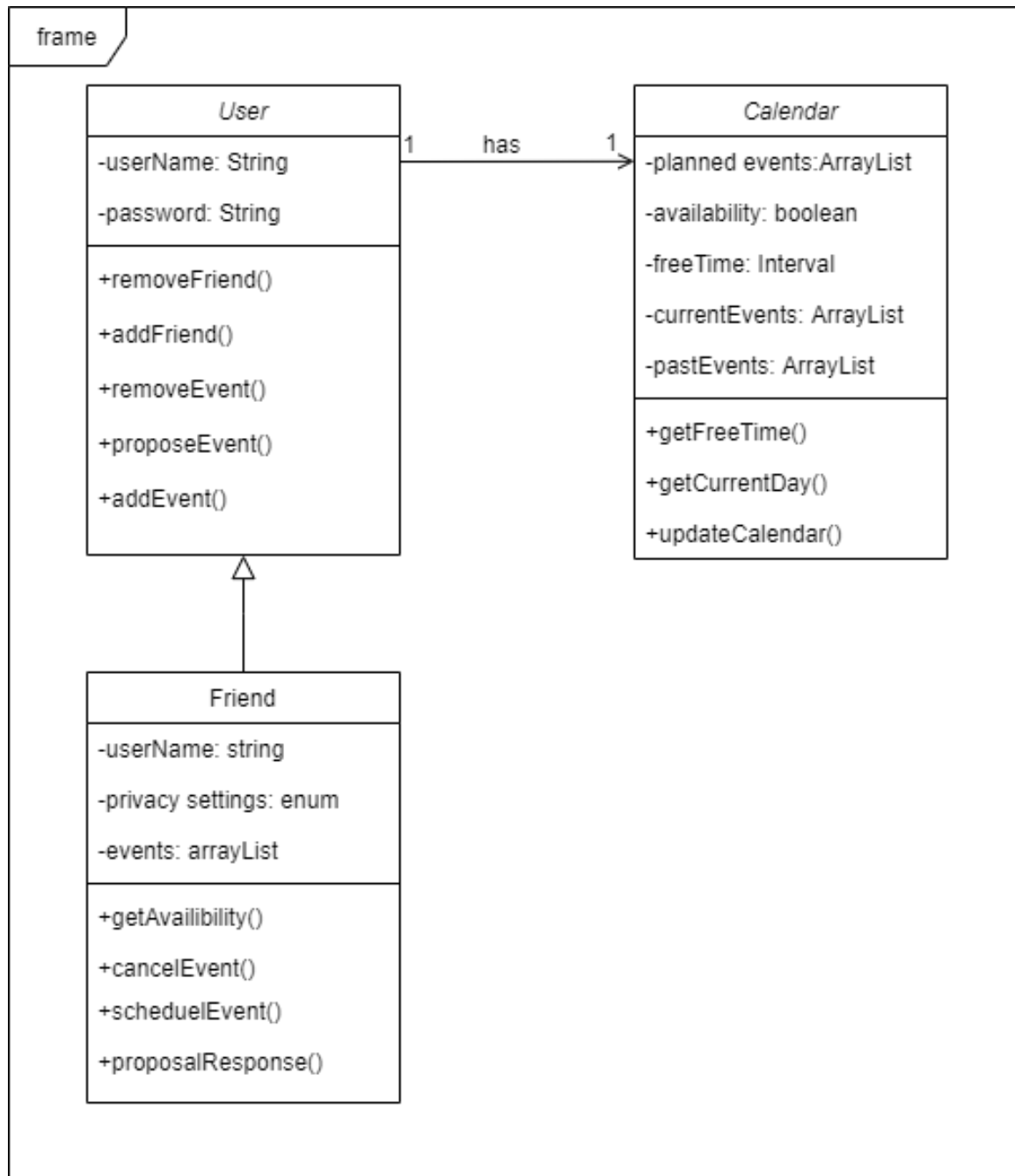
#### 4. Add Friend



## 5. Select Friend



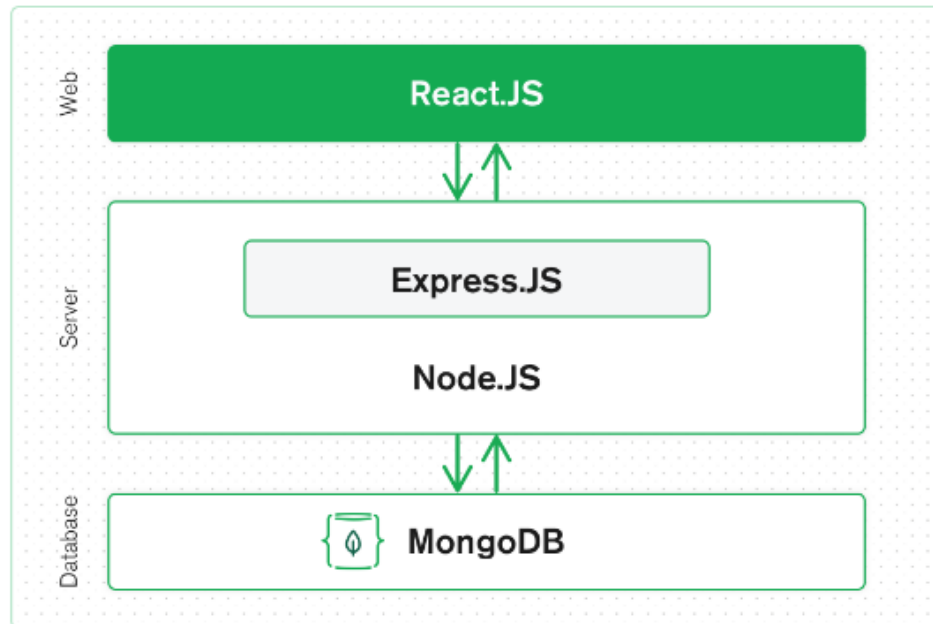
## Class Diagram





## Section 5: Implementation and Evaluation

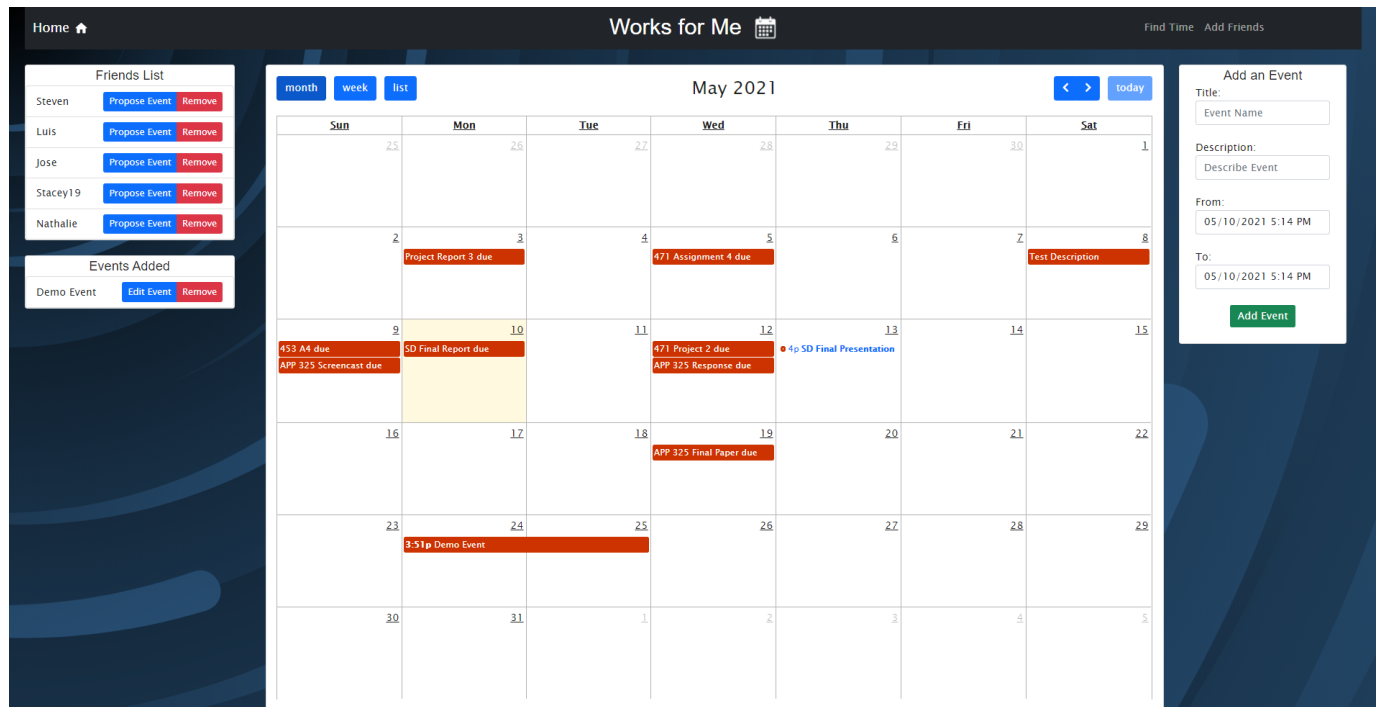
Project Title: Works for Me (*Web Application*)



I began this project by following a tutorial on how to set up a M.E.R.N stack project. First, I started by creating a layout for my project. This layout is ideal because it keeps everything organized as well as allow the project to be scalable. I separated my client and server directories to avoid future confusion. In the client directory I create components that make up the pieces that go on my website. I separate them all so that they function independently from each other, and this also allows me to customize the front-end independently. From the “add event” form I created a Form.js file in which I allow the user to input the date for an event. This information is dispatched when submitted and taken to the createEvent() function found in controllers folder. All the components from my project follow this type of logic.

## Components

### Check Scheduled Events



- I implemented this Calendar using FullCalendar.io, a JavaScript Calendar Library.
- Using the Google Calendar API V3, user's Google Calendar events are synced with my calendar before rendering the calendar.
- For using Google Calendar Events, by default, clicking events takes us to Google Calendar. I plan to change this so that the user gets a pop-up window that shows them the details of that specific event.
- I have not implemented the edit or delete functionality yet, but I plan to have that working soon.

## Add an Event

The image shows a mobile application form titled "Add an Event". The form is contained within a white rounded rectangle with a dark blue border. It features four text input fields and a green submit button. The fields are labeled "Title:", "Description:", "From:", and "To:". The "From:" and "To:" fields contain the date and time "05/10/2021 5:14 PM". The submit button is green with the text "Add Event" in white.

Add an Event

Title:  
Event Name

Description:  
Describe Event

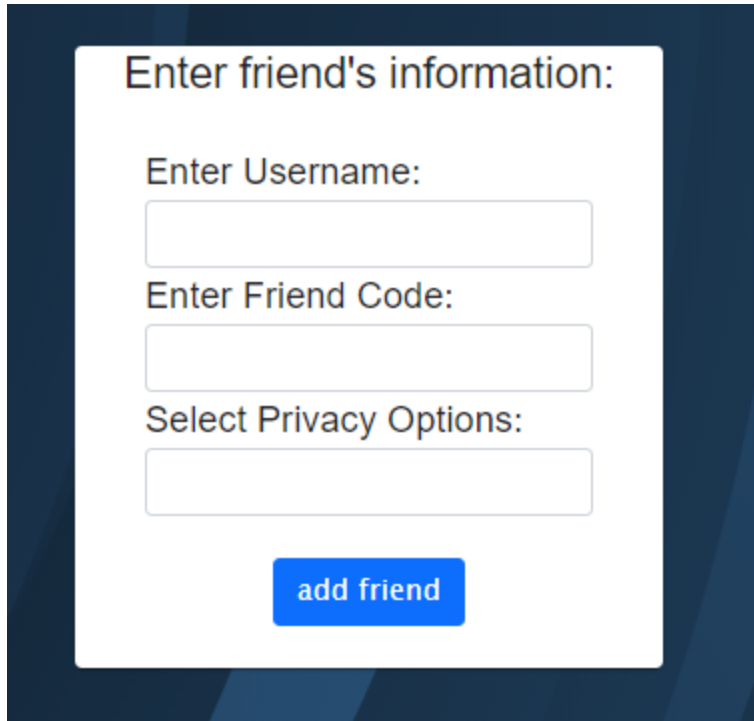
From:  
05/10/2021 5:14 PM

To:  
05/10/2021 5:14 PM

Add Event

- I created this form using material-ui
- Once the user enters the event information into the fields and clicks "Submit", the data is sent to the database.
- A function checks to see if the date/time is available. If so, then it continues with no errors.
- From there we can handle the data and parse it into our Full Calendar.
- After an event is added, it is also sent to Google Calendar. That way when the user exits the application their changes are saved.

## Add Friend



Enter friend's information:

Enter Username:

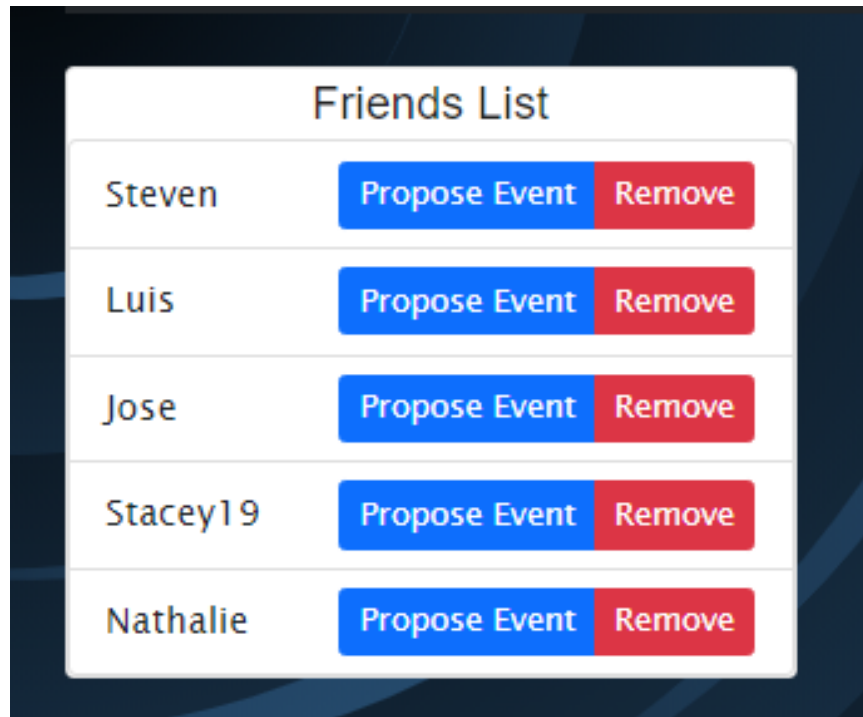
Enter Friend Code:

Select Privacy Options:

add friend

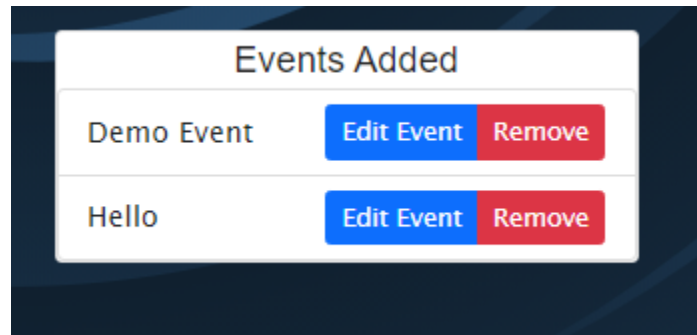
- For the Friend component I plan to make a unique combination of username and a four-digit code that users will exchange with others to add them as friends. This username will be required field when creating an account.
- Once they click the button “send friend request”, the system will check to see if this account exists within the database. If it does exist then it will move on to the next step, otherwise the user will get an error popup letting them know that the account does not exist.
- The recipient of the friend request will have to accept or reject it. Once the recipient accepts the friend request then both users should have each other added to their respective friend’s list. For this I have only setup the front-end part.

## List of Friends



Friends List		
Steven	Propose Event	Remove
Luis	Propose Event	Remove
Jose	Propose Event	Remove
Stacey19	Propose Event	Remove
Nathalie	Propose Event	Remove

- For the friend's list I wanted to create a dropdown menu on each friend that gives the user a couple of options. After selecting a friend, the user should be able to remove friend as well as be able to propose event.
- This "Propose event" option should allow the user to propose an event to a specific friend. I have not yet implemented this functionality.
- I have plans to merge this menu and the previous "add friend" menu into one
- Once added, each of the user's friends should render onto the friends list every time they open the application.
- This form was created using Typography, Button, IconButton, and Paper components from material-ui.



- After the user submits an event from the “Add an Event” form, this group of new events is displayed in the “Events Added” window.
- These events are retrieved directly from the database(MongoDB) and displayed.
- The user can choose to delete the event from their calendar by the clicking “Remove” button. This finds the event by id and removes it from the database, and it reloads the page. Once the page is reloaded, the event should not display on the calendar or the “Events Added” list.
- The user can choose to edit event from their calendar, by clicking the “Edit Event” button. Hypothetically, they should be able to change any field from their event, and once they submit the changes it should save to the database. This feature is currently not working.

## Challenges faced

One of the main challenges that I have faced while working on this project is the sending of data to the database. For the “add Event” form I have a component called React Datepicker that allows the user to choose the date and time that their new event will span over. So, as of now only the text data fields are connecting with the Database. When trying to send the “Date” data types to my mongo DB cluster, I am told that it is not in ISO format. So, I want to figure out a way to modify the date format so that it can be correctly added to the corresponding date on the calendar. I got the date in the correct format but once I received the date information I was struggling to attach that with the selected time. In React-DatePicker there is a time picking addition that is optional, but I suspect that I have to mess with the date format to include hours and minutes.

I plan to eventually create a login page where the user creates an account and selects a username. During this step they will be assigned a friend code and be asked to permit the application to access their Google Calendar Events. Using Google’s OAuth2 authentication I can let the user choose what specific email to import onto the calendar.

I would like to send the events directly onto my calendar and then sync them back up with Google Calendar. As a backup plan I can also send events directly to Google Calendar so that they will show on my calendar. As mentioned previously the events sync from Google Calendar to my calendar every time the page loads. Another thing that I have thought of doing was to include the names of friends in the event

details of friends who have the same event as the user. For example, when a user decides they want to propose an event to a friend and the friend accepts, this event gets added to both calendars. On the calendar this event should have a reminder of what friend it was scheduled with.

I would like to separate the content on the page into multiple pages, as of now it looks a bit cluttered, but I first want to focus on the functionality of the program. I also plan to use Bootstrap to customize many of the components.

### Notable Bugs

- When resizing the page, the calendar's height is compressed. This happens when the window is minimized or when the console is opened.
- After picking a date it sometimes reverts to the default value.
- For the week and day views of the calendar the events appear but there is no corresponding time frame.
- When adding an event where the start date is after the end date it does not return an error like it should. Instead, it adds the event with the date of the start date.



## Back-End

### Adding an Event

```
const onSubmit = (event) =>{  
  event.preventDefault();  
  
  onEventAdded({  
    title,  
    start,  
    end  
  })  
  refreshPage()  
}
```

The process of the adding an event begins when a user fills out the “Add an Event” form. After submitting, the onSubmit function is triggered. The “title, start, end” values are filled in from the form. Every time they are changed they are updated to the values of title, start, and end. These saved values are only sent after the user submits the information on the form.

```
export default function({ onEventAdded}){
```

```
import AddEventModal from "../AddEventModal";
```

```

const calendarRef = useRef(null);

const onEventAdded = event=>{
  //save to database
  axios.post('http://localhost:5000/event/add', event)
    .then(res => console.log(res.data));

  let calendarApi = calendarRef.current.getApi()
  calendarApi.addEvent(event);
}

```

This function is exported and imported into Calendar.jsx. A reference of the Calendar is made called “calendarRef”. Then an axios post call is made to post the event data to the database. This information is also sent to the console as an event object. The Event object is also added to the calendar reference that we made.

```

useEffect(() => {
  axios.get('http://localhost:5000/event/').then(response => {

    const requiredDataFromResponse = response.data;
    const data = requiredDataFromResponse.map(event => ({
      start: event.start,
      end: event.end,
      title: event.title
    }));

    setEvents(data);
  })
  .catch(error => { setEvents([]); })
}, []);

```

In the useEffect hook, axios get call is used to get retrieve the array of events that have been recently added to the database. The calendar then renders all the events onto the appropriate dates.

This works to display the complete set of events, but I think it can be implemented more efficiently. Ideally it would be better if we only must retrieve the new event rather than all the ones already on the calendar. I have not invested too much time trying to find a different approach, but it is in my plans.

### Adding a Friend

```
render(){
  return (
    <div class = "container-fluid col-2">
      <Paper>
        <div align = "center">
          <Typography variant = "h5">Enter friend's information:</Typography>
        </div>
        <br></br>
        <form onSubmit={this.onSubmit}>
          <div className="form-group" class = "container-fluid col-10">
            <Typography variant = "h6">Enter Username:</Typography>
            <input type="text"
              required
              className="form-control"
              value={this.state.username}
              onChange={this.onChangeUsername}
            />
            <Typography variant = "h6">Enter Friend Code:</Typography>
            <input type="text"
              className="form-control"
            />
            <Typography variant = "h6">Select Privacy Options:</Typography>
            <input type="text"
              className="form-control"
            />
          </div>
          <br></br>
          <div className="container-fluid col-12 form-group" align = "center">
            <input type="submit" value="add friend" className="btn btn-primary" />
          </div>
          <br></br>
        </form>
      </Paper>
    </div>
  )
}
```

This is the form that the user uses to input their friend's information. When a new user registers his account, they create a username, enter their email, and are given a

unique “friend code”. All this information is prompted when someone wants to add another user. I also need to add the field of email that will be used to retrieve the user’s Google Calendar data. This should be optional, if the user would like to start with an empty Calendar then that should be available too.

This does not behave the way it should. I currently have no interaction between two users. As previously mentioned users should not be able to add each other without a “friend code”, however I have not yet implemented a login page. Until I integrate that part of the project, I will have to simulate the process of adding a friend. In the database I only have a collection of users but have no way of identifying a specific user other than their username.

```
onSubmit(e) {  
  e.preventDefault();  
  
  const user = {  
    |   username: this.state.username  
  }  
  
  console.log(user)  
  
  axios.post('http://localhost:5000/users/add', user)  
    .then(res => console.log(res.data));  
  
  this.setState({  
    username: '',  
  })  
}
```

```
getUsers = () =>{
  axios.get('http://localhost:5000/users/')
    .then(response => {
      this.setState({ friends: response.data })
    })
    .catch((error) => {
      console.log(error);
    })
}
```

When a user enters the information, this is added to the database as a user. And then this information is also retrieved from the database using a get axios call. The information of friends is put into the form of a list on “Friends List”.

### Connecting to the database

I began by creating a server directory to keep everything I would need for the server separate from my client. I created a “server.js” file that let me choose the port in which the server would run from. This file handled opening the server up and requesting the information it needed. One of the things that it requires is a dotenv file, and this file allows the application to directly link to my account on mongoDB to send, retrieve, and delete data from my cluster. In the server directory I created a “models” folder that creates models for my database schema. I made a event model and a user model and each respective file hold the data fields that each table needs. For the event model it holds a start date (required), end date, and a title (required). For now, the user model only has a username data field.

Next I made “routes” folder in the server directory where I kept an “event.js” file and a “users.js” file. Here I would declare all the type of requests that I would be making to the database. For users I will be needing a get, post, and delete requests. For the events

```
1  const router = require('express').Router();
2  let User = require('../models/user.model');
3
4  router.route('/').get((req, res) => {
5    User.find()
6      .then(users => res.json(users))
7      .catch(err => res.status(400).json('Error: ' + err));
8  });
9
10 router.route('/add').post((req, res) => {
11   const username = req.body.username;
12
13   const newUser = new User({username});
14
15   newUser.save()
16     .then(() => res.json('User added!'))
17     .catch(err => res.status(400).json('Error: ' + err));
18 });
19
20
21 router.route('/:id').delete((req, res) => {
22   User.findByIdAndDelete(req.params.id)
23     .then(() => res.json('User deleted.'))
24     .catch(err => res.status(400).json('Error: ' + err));
25 });
26
27 module.exports = router;
```

What I plan to do next

I have many plans to finish what I could not complete for this project. I think I would like to start by creating a layout for registering accounts for new users for saving

their information and preferences. Once each user has an account, under the user collections I want to add a field that hold an array of other users. This will determine which users are friends. This will open the possibility to get the “Propose event” feature to work. Currently I cannot send a proposal between two friends without an actual way of knowing if they are on each other’s friends list.

## Evaluation

I believe that my implementation is appropriate for now. It allows me to show what I plan to do in the future with this project. There is plenty of work that can be done here, some of the approaches that I took to complete functionality may not be efficient for a large-scale project with many users. Most of the functionality that I wanted to add was missing from this final iteration. I think I spent too much time on figuring out how to structure the backend. If I were to do it again, I would prioritize my time working on functionality over design. As I previously mentioned I plan to continue working on this project until I deem it is finished.

Overall, I believe this project was mostly successful. Although I was not able to accomplish many of the things I set out to do, I am satisfied with the number of things I learned. Getting the backend linked up with the components on the application was tedious, but necessary. The front-end design can be overwhelming because of how many different options are available. It is sometimes confusing when you can use many things to get to the same result. The current UI is very simplistic, but I would like to make it more of a smooth experience for the end user. There is a couple of visual bugs that although may not affect the functionality, may ruin the user experience.

## Section 6: Source Code

### Components/AddEvent.jsx

```
import React, { useState } from "react";
import DatePicker from 'react-datepicker';
import {Paper, Typography, createMuiTheme} from "@material-ui/core"
import useStyles from './styles'

const buttonStyle = {
  color: "white",
  borderColor: "white"
};

export default function({ onEventAdded}){

  const classes = useStyles();
  const theme = createMuiTheme({
    spacing: 2,
  })

  theme.spacing(3)

  const [title, setTitle] = useState("");
  const [start, setStart] = useState(new Date());
  const [end, setEnd] = useState(new Date());

  const refreshPage = () => {
    window.location.reload();
  }

  const onSubmit = (event) =>{
    event.preventDefault();

    onEventAdded({
      title,
      start,
      end
    })
    refreshPage()
  }
}
```



```

return(
  <Paper variant = "outlined" className = {classes.Paper}>
    <Typography variant = "h6" align = "center">Add an Event</Typography>
    <form
      onSubmit={onSubmit}>

      <div class = "container-fluid col-12">
        <Typography variant = "h7" >Title:</Typography>

        <input

          placeholder="Event Name"
          value={title}
          onChange={e=> setTitle(e.target.value)}
          className="form-control"
        />

      </div>
      <br></br>
      <div class = "container-fluid col-12">
        <Typography variant="h7">Description:</Typography>
        <input
          type="text"
          placeholder = "Describe Event"
          className="form-control"
        />
      </div>
      <br></br>
      <div class = "container-fluid col-12" align = "justify">
        <Typography variant = "h7">From:
        <DatePicker
          closeOnScroll = {true}
          value = {start}
          selected = {start}
          onChange={date=>setStart(date)}
          timeInputLabel = "Time:"
          dateFormat = "MM/dd/yyyy h:mm aa"
          showTimeInput
          className="form-control"
        />
        </Typography>
      </div>
      <br></br>
      <div class = "container-fluid col-12" align = "justify" >
        <Typography variant = "h7">To:

```

```

        <DatePicker
          closeOnScroll = {true}
          value = {end}
          selected = {end}
          onChange={date=>setEnd(date)}
          timeInputLabel = "Time:"
          dateFormat = "MM/dd/yyyy h:mm aa"
          showTimeInput

          className="form-control"
        />
      </Typography>
    </div>
    <br></br>
    <div class = "container-fluid col-12" align = "center">
      <button className = {classes.buttonSubmit}
        style = {buttonStyle}
        class="btn btn-success"
        variant = "contained"
        color = "primary"
        size = "small"
        fullWidth
        >Add Event</button>
    </div>

    <br></br>
  </form>
</Paper>
)
}

```

## Components/AddFriend.js

```

import React, { Component } from 'react';
import axios from 'axios';
import {Paper, Typography} from '@material-ui/core'

export default class CreateUser extends Component {
  constructor(props){
    super(props);

    this.onChangeUsername = this.onChangeUsername.bind(this);
    this.onSubmit = this.onSubmit.bind(this);

    this.state = {
      username: ''
    }
  }

```

```

    }
  }
  onChangeUsername(e) {
    this.setState({
      username: e.target.value
    });
  }

  onSubmit(e) {
    e.preventDefault();

    const user = {
      username: this.state.username
    }

    console.log(user)

    axios.post('http://localhost:5000/users/add', user)
      .then(res => console.log(res.data));

    this.setState({
      username: '',
    })
  }
}

render(){
  return (
    <div class = "container-fluid col-2">
      <Paper>
        <div align = "center">
          <Typography variant = "h5">Enter friend's information:</Typography>
        </div>
        <br></br>
        <form onSubmit={this.onSubmit}>
          <div className="form-group" class = "container-fluid col-10">
            <Typography variant = "h6">Enter Username:</Typography>
            <input type="text"
              required
              className="form-control"
              value={this.state.username}
              onChange={this.onChangeUsername}
            />
            <Typography variant = "h6">Enter Friend Code:</Typography>
            <input type="text"

```

```

        className="form-control"
      />
      <Typography variant = "h6">Select Privacy Options:</Typography>
      <input type="text"
        className="form-control"
      />
    </div>
    <br></br>
    <div className="container-fluid col-12 form-group" align = "center">

      <input type="submit" value="add friend" className="btn btn-primary" />

    </div>
    <br></br>
  </form>
</Paper>
</div>
)
}
}

```

components/FindTime.js

```

import React, { Component } from 'react';
import axios from 'axios';
import DatePicker from 'react-datepicker';
import "react-datepicker/dist/react-datepicker.css";
import {Paper, Typography, createMuiTheme} from '@material-ui/core'
import Calendar from './Calendar';
import useStyles from './styles'

const buttonStyle = {
  color: "white",
  borderColor: "white"
};

export default class CreateEvent extends Component {

  constructor(props){
    super(props);

    this.onChangeStart = this.onChangeStart.bind(this);
    this.onChangeEnd = this.onChangeEnd.bind(this);
  }

```

```

    this.onChangeTitle = this.onChangeTitle.bind(this);
    this.onSubmit = this.onSubmit.bind(this);

    this.state = {
      start: new Date(),
      end: new Date(),
      title: ''
    }
  }

  componentDidMount() {
    // this.setState({
    //   start: '2021-05-07T03:50:19.274+00:00',
    //   end: '2021-05-07T03:50:19.274+00:00',
    //   title: ''
    // })
  }

  onChangeStart(date) {
    this.setState({
      start: date
    });
  }

  onChangeEnd(date) {
    this.setState({
      end: date
    });
  }

  onChangeTitle(e) {
    this.setState({
      title: e.target.value
    });
  }

  onSubmit(e) {
    e.preventDefault();

    const event = {
      title: this.state.title,
      start: this.state.start,
      end: this.state.end,
    }

    console.log(event)
  }

```

```

    axios.post('http://localhost:5000/event/add', event)
    .then(res => console.log(res.data));

    window.location = '/';

  }
  render(){
    return (
      <div class = "container-fluid col-5">
        <Paper variant = "outlined">
          <form >
            <div>
              <Typography variant = "h6" align = "center">Need help scheduling an event?</Typogr
aphy>

            </div>
            <br></br>
            <div class = "container-fluid col-10">
              <Typography variant = "h7" >Fill out some preferences:</Typography>

              <input
                placeholder="Weekdays/Weekends"
                className="form-control"
              />

            </div>
            <br></br>
            <div class = "container-fluid col-10">
              <Typography variant="h7">Type of event:</Typography>
              <input
                type="text"
                placeholder = "Personal/Business"
                className="form-control"
              />
            </div>
            <br></br>
            <div class = "container-fluid col-10" align = "justify">
              <Typography variant = "h7"> Looking for available time between this interval:</Typ
ography>

            </div>
            <br></br>
            <div class = "container-fluid col-10" align = "justify">
              <Typography variant = "h7">From </Typography>
              <DatePicker
                closeOnScroll = {true}

```

```

        // value = {start}
        // selected = {start}
        // onChange={date=>setStart(date)}
        timeInputLabel = "Time:"
        dateFormat = "MM/dd/yyyy h:mm aa"
        showTimeInput
        className="form-control"
    />

</div>
<br></br>
<div class = "container-fluid col-10" align = "justify" >
    <Typography variant = "h7">To </Typography>
    <DatePicker
        closeOnScroll = {true}
        // value = {end}
        // selected = {end}
        // onChange={date=>setEnd(date)}
        timeInputLabel = "Time:"
        dateFormat = "MM/dd/yyyy h:mm aa"
        showTimeInput

        className="form-control"
    />

</div>
<br></br>
<div class = "container-fluid col-12" align = "center">
    <button
        style = {buttonStyle}
        class="btn btn-primary"
        variant = "contained"
        color = "primary"
        size = "small"
        fullWidth
    >Add Event</button>
</div>

<br></br>
</form>
</Paper>

</div>

```

)

```
}  
}
```

## Components/Homepage.js

```
import React, { Component } from 'react';  
import axios from 'axios';  
import "bootstrap/dist/css/bootstrap.min.css";  
import Calendar from "../Calendar"  
  
import {Paper, Typography} from "@material-ui/core"  
  
const Friend = props => (  
  <li class="list-group-item d-flex justify-content-between align-items-center">  
    {props.user.username}  
    <div class="btn-group btn-group-toggle" data-toggle="buttons">  
      <button  
        name="options" id="option2" autocomplete="off"  
        class="btn btn-primary btn-sm"  
      >Propose Event</button>  
      <button  
        name="options" id="option1" autocomplete="off"  
        class="btn btn-danger btn-sm"  
        color = "secondary"  
        onClick={() => { props.deleteFriends(props.user._id) }}  
      >Remove</button>  
    </div>  
  </li>  
)  
  
const Event = props => (  
  <li class="list-group-item d-flex justify-content-between align-items-center">  
    {props.name.title}  
    <div class="btn-group btn-group-toggle" data-toggle="buttons">  
      <button  
        name="options" id="option2" autocomplete="off"  
        class="btn btn-primary btn-sm"  
      >Edit Event</button>  
      <button  
        name="options" id="option1" autocomplete="off"  
        class="btn btn-danger btn-sm"  
        color = "secondary"  
        onClick={() => { props.deleteEvents(props.name._id) }}  
      >Remove</button>  
    </div>  
  </li>  
)
```



```

        </div>
      </li>
    )
  }

export default class EventList extends Component {

  constructor(props){
    super(props);

    this.deleteFriends = this.deleteFriends.bind(this);
    this.deleteEvents = this.deleteEvents.bind(this);

    this.state = {friends: [], events:[]};
  }

  componentDidMount() {
    // get users
    this.getUsers();
    // get events
    this.getEvents();
  }

  reloadPage = () => {
    window.location.reload()
  }

  getUsers = () =>{
    axios.get('http://localhost:5000/users/')
      .then(response => {
        this.setState({ friends: response.data })
      })
      .catch((error) => {
        console.log(error);
      })
  }

  getEvents = () =>{
    axios.get('http://localhost:5000/event/')
      .then(response => {
        this.setState({ events: response.data })
      })
      .catch((error) => {
        console.log(error);
      })
  }
}

```

```

deleteFriends(id) {
  axios.delete('http://localhost:5000/users/'+id)
    .then(response => { console.log(response.data)});

  this.setState({
    friends: this.state.friends.filter(el => el._id !== id)
  })
}

deleteEvents(id) {
  axios.delete('http://localhost:5000/event/'+id)
    .then(response => { console.log(response.data)});

  this.setState({
    events: this.state.events.filter(el => el._id !== id)
  })

  // setTimeout(function () {
  //   window.location.reload()
  // }.bind(this), 10)
}

friendsList() {
  return this.state.friends.map(currentfriend => {
    return <Friend user={currentfriend} deleteFriends={this.deleteFriends} key={currentfriend._id}/>;
  })
}

eventsList() {
  return this.state.events.map(currentEvent => {
    return <Event name={currentEvent} deleteEvents={this.deleteEvents} key={currentEvent._id}/>;
  })
}

render(){
  return (
    <div>
      <div class="container-fluid">
        <div class="row">
          <div class="col-2">
            <div>
              <Paper variant = "outlined" >
                <Typography variant = 'h6' align = "center">Friends L
ist</Typography>

```

```

        <div class="list-group">
            { this.friendsList() }
        </div>

    </Paper>
</div>
<br><br>
<div>
    <Paper variant = "outlined" >
        <Typography variant = 'h6' align = "center">Events Added</Typography>
        <div class="list-group">
            { this.eventsList() }
        </div>

    </Paper>
</div>
</div>
<div class="col-10 ">
    <Calendar/>
</div>

</div>
</div>
</div>
)
}
}

```

## Components/Calendar.jsx

```

import React, { useRef, useState, useEffect } from "react";
import FullCalendar from "@fullcalendar/react"
import dayGridPlugin from "@fullcalendar/daygrid"
import timeGridPlugin from "@fullcalendar/timegrid"
import listPlugin from '@fullcalendar/list'
import interactionPlugin from '@fullcalendar/interaction';
import googleCalendarPlugin from '@fullcalendar/google-calendar';
import {Paper,createMuiTheme} from '@material-ui/core'
import '../..../node_modules/@fortawesome/fontawesome-free/css/all.css'

import '@fullcalendar/bootstrap/main.css'

```

```

import '../..node_modules/bootstrap/dist/css/bootstrap.css'

import '@fortawesome/fontawesome-free/css/all.css';
import bootstrapPlugin from '@fullcalendar/bootstrap';

import '../..node_modules/@fullcalendar/bootstrap/main.css'; // our app's CSS
import useStyles from './styles'
import axios from 'axios';

import AddEventModal from './AddEvent";

export default function(){

  const classes = useStyles();
  const theme = createMuiTheme({
    spacing: 2,
  })

  theme.spacing(3)

  const calendarRef = useRef(null);

  const onEventAdded = event=>{
    //save to database
    axios.post('http://localhost:5000/event/add', event)
      .then(res => console.log(res.data));

    let calendarApi = calendarRef.current.getApi()
    calendarApi.addEvent(event);
  }
  const [events, setEvents] = useState([]);

  useEffect(() => {
    axios.get('http://localhost:5000/event/').then(response => {

      const requiredDataFromResponse = response.data;
      const data = requiredDataFromResponse.map(event => ({
        start: event.start,
        end: event.end,
        title: event.title

      }));
    });
  });

```

```

        setEvents(data);
    })
    .catch(error => { setEvents([]); })
}, []);

return (
    <div>
        <div class="container-fluid">
            <div class="row">
                { /* calendar */ }
                <div class="col-10">
                    <Paper variant = "outlined" className = {classes.paper}>
                        <div style={{position:"relative", zIndex:0}}>
                            <FullCalendar className = {classes.FullCalendar}
                                ref={calendarRef}
                                plugins = {[dayGridPlugin, timeGridPlugin, interactionPlugin, googleCalendarPlugin,bootstrapPlugin, listPlugin]}
                                themeSystem = "bootstrap"
                                aspectRatio = "1.50"
                                headerToolbar={{
                                    left: "dayGridMonth timeGridWeek listDay",
                                    center: "title",
                                    right: "prev,next today",
                                }}

                                windowResizeDelay = "200"
                                eventBackgroundColor = "#cc3300"
                                // #cc3300
                                // #006699

                                eventBorderColor = "#cc3300"
                                background-color = "#cc3300"

                                selectable = {true}
                                editable = {true}
                                unselectAuto = {true}
                                weekends = {true}
                                navLinks = {true}

                                googleCalendarApiKey= 'AIzaSyA7ILoMzctHVI16y1LWaTPUKlMp1sWcT_Q'
                                eventSources= [{
                                    {
                                        googleCalendarId: 'giovanniarevalo246@gmail.com',

```

```

        },
        events
    ]}

    />
  </div>
</Paper>
</div>
{/* add event */}
<div class="container-fluid col-2">
  <AddEventModal
    onEventAdded={event => onEventAdded(event)}
  />
</div>
</div>
</div>
)
}

```

## Components/navbar.js

```

import React, { Component } from 'react';
import { Link } from 'react-router-dom';
import "bootstrap/dist/css/bootstrap.min.css";
import { Icon, Typography } from '@material-ui/core'
import pic from "../images/whitecal.png"
import HomeRoundedIcon from '@material-ui/icons/HomeRounded';

export default class Navbar extends Component {

  render() {
    return (
      <div class = "container-fluid" position= "stickytop">
        <nav className="navbar navbar-dark bg-dark navbar-expand-lg">
          {/* leftcol */}
          <div class="col-2">

```



```

table,
th,
td {
  border: 1px solid rgb(180, 179, 179);
}
fc-toolbar-chunk {
  background-color: blanchedalmond;
  border-color: blueviolet;
}

```

## Components/styles.js

```

import { makeStyles } from '@material-ui/core/styles';
import './Calendar.css'

export default makeStyles((theme) => ({
  root: {
    '& .MuiTextField-root': {
      margin: theme.spacing(1),
    },
  },
  paper: {
    padding: theme.spacing(2),
    backgroundColor: "white"
  },
  form: {
    display: 'flex',
    flexWrap: 'wrap',
    justifyContent: 'center',
    marginBottom: -5
  },
  buttonSubmit: {
    marginBottom: 5,
    marginTop: 5
  },
  FullCalendar: {
    margin: '0 auto',
    maxWidth: '900px',
  },
}));

```



## App.js

```
import "bootstrap/dist/css/bootstrap.min.css";
import { BrowserRouter as Router, Route, } from "react-router-dom";

import Navbar from "../components/navbar"
import EventList from "../components/HomePage";
import CreateEvent from "../components/FindTime";
import CreateUser from "../components/AddFriend";
import Modal from "react-modal"

Modal.setAppElement('#root')
function App() {
  return (
    <Router>
      { /* navbar */ }
      <div className="container-fluid">
        <Navbar />
        <br/>
        <Route path="/" exact component={EventList} />
        { /* <Route path="/edit/:id" component={EditEvent} /> */ }
        <Route path="/create" component={CreateEvent} />
        <Route path="/user" component={CreateUser} />
      </div>
    </Router>
  );
}

export default App;
```

## App.css

```
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}
```

```

    }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

```

## Index.js

```

import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import './index.css';

ReactDOM.render(<App />, document.getElementById('root'));

```

## Index.css

```

body {
  background-color: #295278;
}

```

```

background-
image: url("data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg' viewBox='0 0 2000 1500'%3E%3Cdefs
%3E%3CradialGradient id='a' gradientUnits='objectBoundingBox'%3E%3Cstop offset='0' stop-
color='%23000000'/%3E%3Cstop offset='1' stop-
color='%23295278'/%3E%3C/radialGradient%3E%3ClinearGradient id='b' gradientUnits='userSpaceOnUse' x1='0' y
1='750' x2='1550' y2='750'%3E%3Cstop offset='0' stop-color='%2315293c'/%3E%3Cstop offset='1' stop-
color='%23295278'/%3E%3C/linearGradient%3E%3Cpath id='s' fill='url(%23b)' d='M1549.2 51.6c-5.4 99.1-
20.2 197.6-44.2 293.6c-24.1 96-57.4 189.4-99.3 278.6c-41.9 89.2-92.4 174.1-150.3 253.3c-58 79.2-
123.4 152.6-195.1 219c-71.7 66.4-149.6 125.8-232.2 177.2c-82.7 51.4-170.1 94.7-260.7 129.1c-90.6 34.4-
184.4 60-279.5 76.3C192.6 149.5 96.1 150.2 0 150.0c96.1-2.1 191.8-13.3 285.4-33.6c93.6-20.2 185-49.5 272.5-
87.2c87.6-37.7 171.3-83.8 249.6-137.3c78.4-53.5 151.5-114.5 217.9-181.7c66.5-67.2 126.4-140.7 178.6-
218.9c52.3-78.3 96.9-161.4 133-247.9c36.1-86.5 63.8-176.2 82.6-267.6c18.8-91.4 28.6-184.4 29.6-277.4c0.3-
27.6 23.2-48.7 50.8-48.4s49.5 21.8 49.2 49.5c0 0.7 0 1.3-
0.1 2L1549.2 51.6z'/%3E%3Cg id='g'%3E%3Cuse href='%23s' transform='scale(0.12) rotate(60)'/%3E%3Cuse href=
'%23s' transform='scale(0.2) rotate(10)'/%3E%3Cuse href='%23s' transform='scale(0.25) rotate(40)'/%3E%3Cus
e href='%23s' transform='scale(0.3) rotate(-20)'/%3E%3Cuse href='%23s' transform='scale(0.4) rotate(-
30)'/%3E%3Cuse href='%23s' transform='scale(0.5) rotate(20)'/%3E%3Cuse href='%23s' transform='scale(0.6) r
otate(60)'/%3E%3Cuse href='%23s' transform='scale(0.7) rotate(10)'/%3E%3Cuse href='%23s' transform='scale(
0.835) rotate(-
40)'/%3E%3Cuse href='%23s' transform='scale(0.9) rotate(40)'/%3E%3Cuse href='%23s' transform='scale(1.05)
rotate(25)'/%3E%3Cuse href='%23s' transform='scale(1.2) rotate(8)'/%3E%3Cuse href='%23s' transform='scale(
1.333) rotate(-60)'/%3E%3Cuse href='%23s' transform='scale(1.45) rotate(-
30)'/%3E%3Cuse href='%23s' transform='scale(1.6) rotate(10)'/%3E%3C/g%3E%3C/defs%3E%3Cg transform='rotate(
0 0 0)'%3E%3Cg transform='rotate(0 0 0)'%3E%3Ccircle fill='url(%23a)' r='3000'/%3E%3Cg opacity='0.5'%3E%3C
circle fill='url(%23a)' r='2000'/%3E%3Ccircle fill='url(%23a)' r='1800'/%3E%3Ccircle fill='url(%23a)' r='1
700'/%3E%3Ccircle fill='url(%23a)' r='1651'/%3E%3Ccircle fill='url(%23a)' r='1450'/%3E%3Ccircle fill='url(
%23a)' r='1250'/%3E%3Ccircle fill='url(%23a)' r='1175'/%3E%3Ccircle fill='url(%23a)' r='900'/%3E%3Ccircle
fill='url(%23a)' r='750'/%3E%3Ccircle fill='url(%23a)' r='500'/%3E%3Ccircle fill='url(%23a)' r='380'/%3E%3
Ccircle fill='url(%23a)' r='250'/%3E%3C/g%3E%3Cg transform='rotate(0 0 0)'%3E%3Cuse href='%23g' transform=
'rotate(10)'/%3E%3Cuse href='%23g' transform='rotate(120)'/%3E%3Cuse href='%23g' transform='rotate(240)'/%
3E%3C/g%3E%3Ccircle fill-opacity='0.1' fill='url(%23a)' r='3000'/%3E%3C/g%3E%3C/g%3E%3C/svg%3E");
background-attachment: fixed;
background-size: cover;
}

code {
font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
monospace;
}

```

## Server/models/event.model.js

```
const mongoose = require('mongoose');
```

```

const Schema = mongoose.Schema;

const eventSchema = new Schema({
  start: {type: Date, required: true},
  end: {type: Date},
  title: {type: String, required: true},
});

const Event = mongoose.model('Event', eventSchema);

module.exports = Event;

```

### Server/models/user.model.js

```

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const userSchema = new Schema({
  username: {
    type: String,
    required: true,
    unique: true,
    trim: true,
    minlength: 3
  },
}, {
  timestamps: true,
});

const User = mongoose.model('User', userSchema);

module.exports = User;

```

### server/routes/event.js

```

const router = require('express').Router();
let Event = require('../models/event.model');
const moment = require('moment');

router.route('/').get((req, res) => {
  Event.find()
    .then(events => res.json(events))
    .catch(err => res.status(400).json('Error: ' + err));
});

```

```

router.route('/add').post((req, res) => {
  const start = req.body.start;
  const end = req.body.end;
  const title = req.body.title;

  const newEvent = new Event({
    start,
    end,
    title
  });

  newEvent.save()
    .then(() => res.json('Event added!'))
    .catch(err => res.status(400).json('Error: ' + err));
});

router.route('/:id').get((req, res) => {
  Event.findById(req.params.id)
    .then(event => res.json(event))
    .catch(err => res.status(400).json('Error: ' + err));
});

router.route('/:id').delete((req, res) => {
  Event.findByIdAndDelete(req.params.id)
    .then(() => res.json('Event deleted.'))
    .catch(err => res.status(400).json('Error: ' + err));
});

router.route('/update/:id').post((req, res) => {
  Event.findById(req.params.id)
    .then(event => {
      event.start = req.body.start;
      event.end = req.body.end;
      event.title = req.body.title;

      event.save()
        .then(() => res.json('Event updated!'))
        .catch(err => res.status(400).json('Error: ' + err));
    })
    .catch(err => res.status(400).json('Error: ' + err));
});

module.exports = router;

```

## server/routes/users.js

```
const router = require('express').Router();
let User = require('../models/user.model');

router.route('/').get((req, res) => {
  User.find()
    .then(users => res.json(users))
    .catch(err => res.status(400).json('Error: ' + err));
});

router.route('/add').post((req, res) => {
  const username = req.body.username;

  const newUser = new User({username});

  newUser.save()
    .then(() => res.json('User added!'))
    .catch(err => res.status(400).json('Error: ' + err));
});

router.route('/:id').delete((req, res) => {
  User.findByIdAndDelete(req.params.id)
    .then(() => res.json('User deleted.'))
    .catch(err => res.status(400).json('Error: ' + err));
});

module.exports = router;
```

## server/sever.js

```
const express = require('express');
const cors = require('cors');
const mongoose = require('mongoose');

require('dotenv').config();

const app = express();
const port = process.env.PORT || 5000;

app.use(cors());
app.use(express.json());

const uri = process.env.ATLAS_URI;
mongoose.connect(uri, { useNewUrlParser: true, useCreateIndex: true })
```

```

);
const connection = mongoose.connection;
connection.once('open', () => {
  console.log("MongoDB database connection established successfully");
})

const eventsRouter = require('./routes/event');
const usersRouter = require('./routes/users');

app.use('/event', eventsRouter);
app.use('/users', usersRouter);

app.listen(port, () => {
  console.log(`Server is running on port: ${port}`);
});

```

## Works-for-me/package.json

```

{
  "name": "works-for-me",
  "version": "0.1.0",
  "private": true,
  "proxy": "http://localhost:3000",
  "dependencies": {
    "@fortawesome/fontawesome-free": "^5.15.3",
    "@fullcalendar/bootstrap": "^5.6.0",
    "@fullcalendar/core": "^5.6.0",
    "@fullcalendar/daygrid": "^5.6.0",
    "@fullcalendar/google-calendar": "^5.6.0",
    "@fullcalendar/interaction": "^5.6.0",
    "@fullcalendar/list": "^5.6.0",
    "@fullcalendar/react": "^5.6.0",
    "@fullcalendar/timegrid": "^5.6.0",
    "@material-ui/core": "^4.11.4",
    "@material-ui/icons": "^4.11.2",
    "@testing-library/jest-dom": "^5.12.0",
    "@testing-library/react": "^11.2.6",
    "@testing-library/user-event": "^12.8.3",
    "axios": "^0.21.1",
    "bootstrap": "^5.0.0",
    "fullcalendar": "^5.6.0",
    "react": "^17.0.2",
    "react-datepicker": "^3.8.0",
    "react-dom": "^17.0.2",
    "react-modal": "^3.13.1",

```

```
    "react-router-dom": "^5.2.0",
    "react-scripts": "4.0.3",
    "web-vitals": "^1.1.2"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```