Calendar Project

Group 9 - Elliot Hughes, Haseebuddin Syed, Katlyn Becvar, and Stanislav TurovskyCPSC-49200-002

# Proposal

We proposed creating a Zoom meeting scheduling website, called the Calendar Project, that somewhat mimics the Calendly.com [1] version, used by at least one Administrator and at least one User. Both a User and an Administrator are able to register and reset their logins for the website.

An Administrator acts as a secretary/clerk for a User who would be a manager or a professor. This person manages User’s daily availability, manages appointment type selections and their time ranges, and suggests actual appointments (including when they recur and who will attend them) for the User.

A User is able to ask an Administrator to plan new appointments (without specifics, such as “find a time I can meet with representatives of such and such company when I am free next week”), change scheduling settings (such as daily availability) [implied part], verify appointment type details, personally setup/change/delete connected Zoom account, view/approve/deny appointment requests sent by an Administrator, and send approved appointments to the connected Zoom account, which will then respond with an email confirmation along with a zoom link.

The features that have been requested for the project are creating specific logins for the Administrator(s) and the User(s) and doing the already described actions for each role.

As was planned, at the end of the project, the team implemented a basic GUI representation (but not always fully functional when too complex to implement) of each of the given requests, as well as implemented what was implied as necessary functions as well.

The team’s approach was to split the tasks, research the task(s) given, try various ways to achieve the goal, seek team’s approval on the selected solution, and then implement it into the project we have working.

[1] Free Online Appointment Scheduling Software – Calendly. <https://calendly.com/> (Accessed March 21, 2021)

# Test Planning

Our test plan was to simply try running the project as both a User (username = “User” and password = “userpwd”) and then as an Administrator (username = “admin” and password “pwd’), using their corresponding login/passwords. When you are logged in as the Admin, our login page will let you know that it recognizes it (for additional Administrator-only features). With a User login, our website won't tell you anything extra, and it will just take you to the default User menu page. We then expanded more on the results of the testing, and used scenarios such as: a misspelled password for an Administrator, creating appointments in conflict on the same date, etc. To test “if one cancels an appointment how can I verify if that time block updates?” we simply went through the logic of performing the action to remove an appointment (cancel it) and then noted that it would be removed from the list of the appointments, and that we would then be able to add another meeting at the same exact time (an action that would normally be blocked by our collision detection logic).

If Zoom API was fully implemented, we would also check that the Zoom meeting is cancelled. Login (and most of the project data that is not temporarily added) is currently hardcoded, so database functionality can’t be tested.

# Features Table

Features table is shown below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Description** | **Software Used** | **Implemented** |
| Admin Part | Functions for the host(s), who will assist the User(s). Mostly static example with temporarily modifiable parts. | React.js | Yes |
| Appointment List | Allows user to view, edit, and delete appointments, as well as adjust availability. Temporarily modifiable. | React.js | Yes |
| Appointment Types | Sets up names and time difference between appointment types. Static Example. | React.js | Yes |
| Date/time picker | Allows the user/admin to select a time/date for each appointment. Static example. | React.js | Yes |
| Login System | Login for the user(s) and Administrator(s). Hardcoded credentials. | React.js | Yes |
| Main Page | Allows viewing public page or log in to see personalized menus. | React.js | Yes |
| Password Reset | Allows changing user's password. Static example. | React.js | Yes |
| Registration | Account creation so that a user can interact with the app. Static example. | React.js | Yes |
| User Part | User related functions, such as viewing appointments and approving meetings for Zoom transfer. Mostly static example with temporarily modifiable parts. | React.js | Yes |
| Zoom API | Connects, changes, or deletes associated Zoom account. Static example. | React.js | Yes |

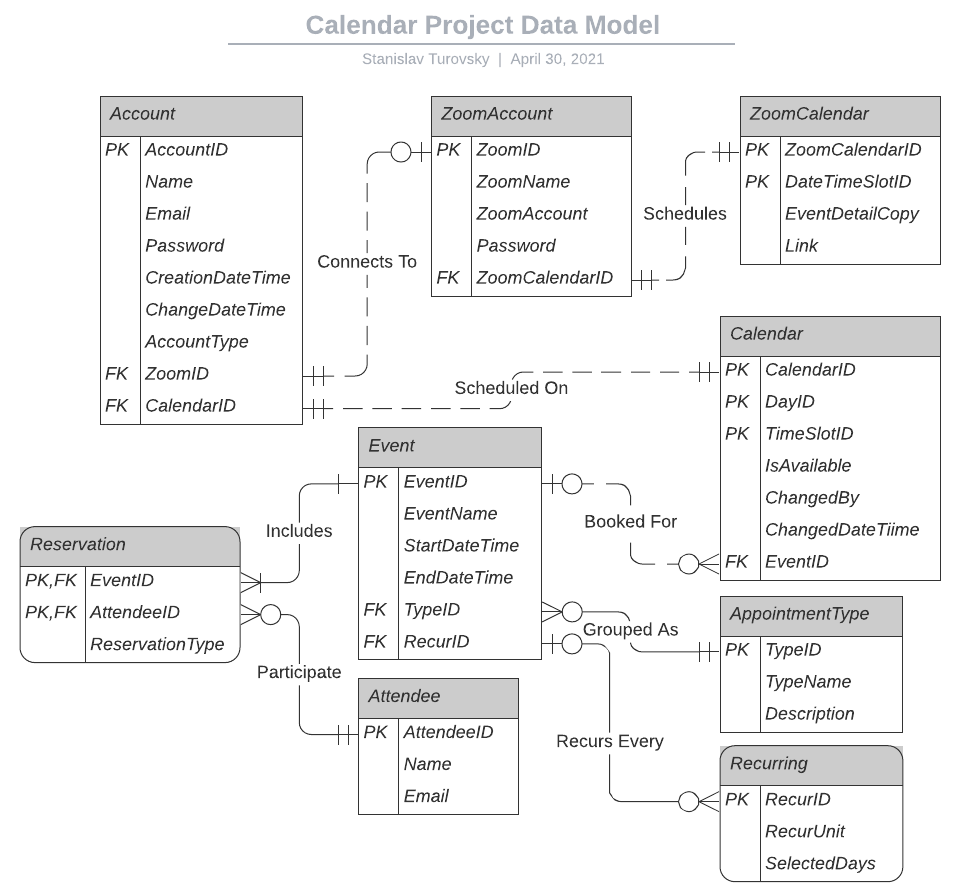
# Requirements

To use the project, both User(s) and Administrator(s) need a device with internet access and a web browser (notably, the latest non-ESR Firefox browser causes issues and should be avoided). To set up the project, we used multiple collaboration and development tools (both local and web based).

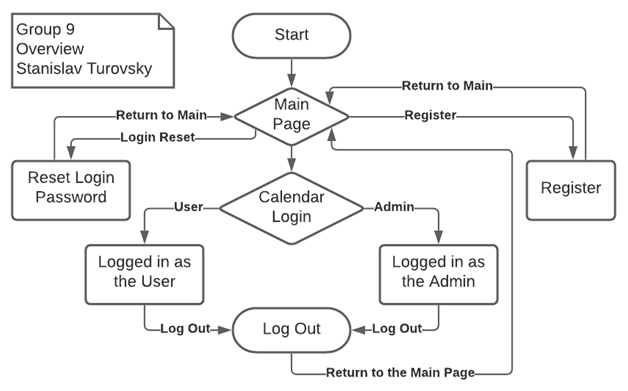
We used Discord for conversations about how we would implement and design the project. To use it, simply create an account at <https://discord.com/>. Discord hosted several different channels for our project, which were broken up into *general*, *links*, *alternate*, *requirements*, *assignments*, *availability*, *commands*, *zoom*, *node-diagram*, *manual*, and *presentation*. Each channel served a vital purpose to our project. *General* channel is where we discussed all our general ideas, and troubleshooting takes place. *Links* channel focused on keeping quick access to links of all the documents we’ve found or created ourselves, such as links to code examples, our Repit.com (formerly Repl.it) project code and related tests, GitHub repository, and Google docs. *Alternate* channel allowed discussions while someone was trying to read something in *General* and chat at the same time, without losing the exact spot being read. *Requirements* channel was specific to our project requirement checklist and served to check in once each task is completed. *Assignments* channel was where we focused on assigning tasks to each team member. *Availability* channel provided our individual availability if we were unable to participate in work or discussion for specific days and/or times. *Commands* channel was our chat designated for commands we’ve used to run our project, like *npm/yarn start/run/install*. *Zoom* channel hosted a chat specific to our conference calls with Dr. Lewis. *Node-diagram* channel was specific to our node diagrams. *Manual* channel was used to discuss changes to the manual. Lastly, *presentation* channel was used to discuss our project presentation.

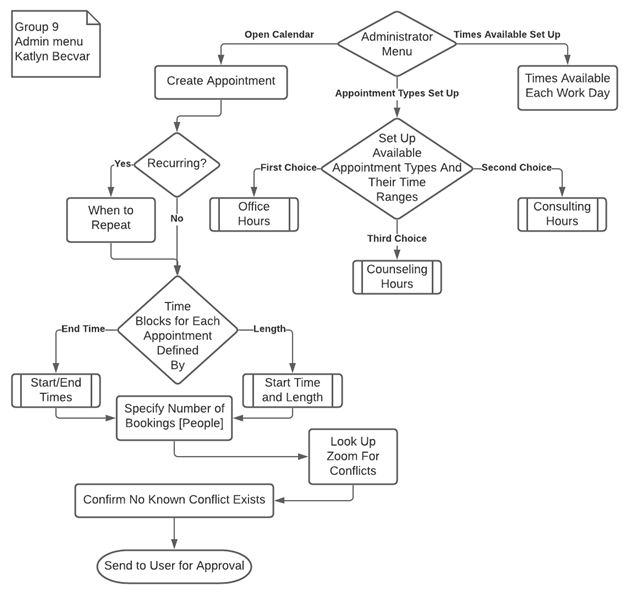
Replit.com (formerly Repl.it) was used for live time editing of code and multi-user editing abilities. You can create an account at its website, <https://replit.com/>. Along with being able to edit live with our group, Replit.com was also used to hold our current working project. To create our application workflow, project overview, and project’s data model, we created node and ERD graphs in Lucid Charts (sign up is at <https://www.lucidchart.com/pages/>). To run replit.com properly with some examples of React and Node.js, we used to install packages by running “install npm” and additional commands from replit.com. However, it was found that simply mentioning a package name was enough to trigger its install. Besides the web-based applications we used, we also installed the Node.js framework (available at <https://nodejs.org/en/>, React.js on top of Node.js, MS Visual Studio Code (available at <https://code.visualstudio.com/>), and several NPM packages (available at <https://www.npmjs.com/>).

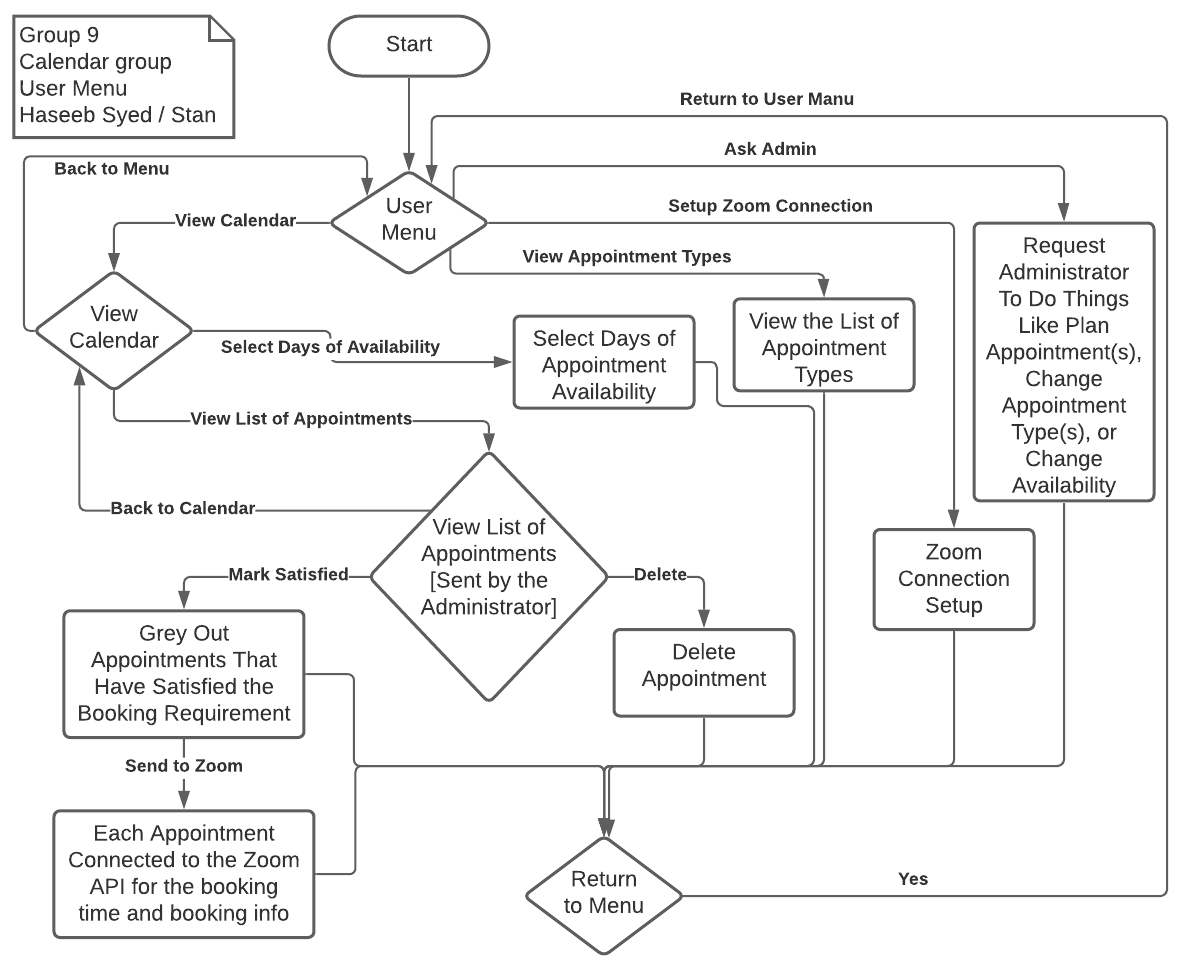
# Data Model



# Node Graphs







# 

# Conclusion

It was a very challenging project, not only from the technological difficulty perspective, but also from overcoming various team conflicts/issues as well. We completed the diagrams (both node and data model ones), project presentation, and the manual itself without issues. We struggled with implementing any kind of a database including things like JSON using React (even though we could get it working in Node.js), so it was skipped in favor of hard-coding and using temporary data changes instead. We also struggled with finalizing calendar related functionalities, but we still managed to implement a good static example with some temporarily changeable parts. Finally, we decided to skip fully implementing Zoom API connection in favor of yet another static example with temporarily changeable data parts as well. Future features would implement everything that wasn’t implemented yet (as described above).