# Proposal

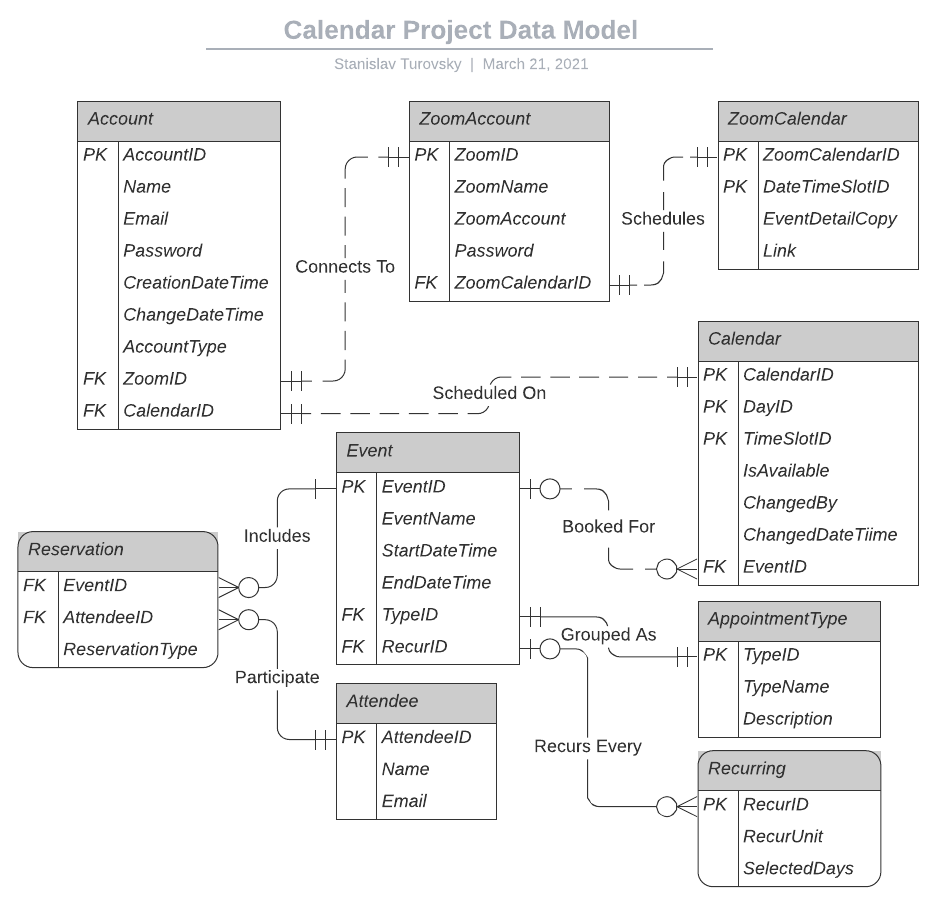
We propose creating a Zoom meeting scheduling website, called the Calendar Project, that somewhat mimics the Calendly.com [1] version, and is to be used by at least one Administrator and at least one User. Both the User and the Administrator will be able to create and modify their logins for the website. The Administrator will act as a secretary/clerk for the User who would be a manager or a professor. The Administrator will manage User’s daily availability, manage appointment type selections and their time ranges, and suggest actual appointments (including when they recur and who will attend them) for the User. The User will be able to ask 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”) or change scheduling settings [implied part], personally setup/change Zoom account, view/approve/deny the appointment requests sent by the Administrator, and send approved appointments to a 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. By the end of the project, the team will implement at least a basic GUI representation (not necessarily functional if too complex to implement) of each of the given requests, as well as implement what was implied as necessary functions as well. The team’s approach is 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)

# Requirements

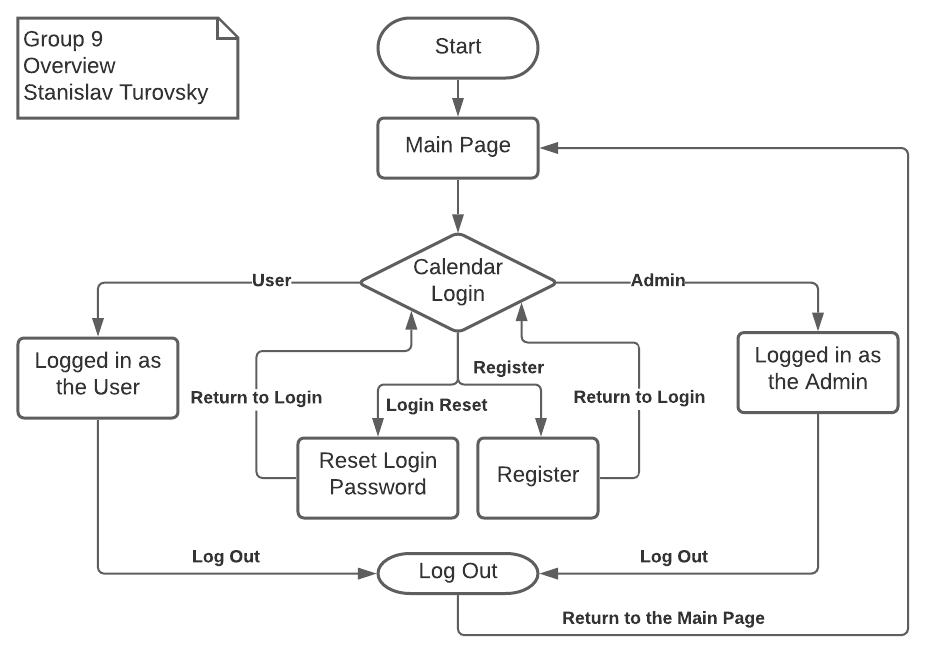
To use the project, both User(s) and Administrator(s) will need a device with internet access and a web browser. To set up the project, we used multiple collaboration and development tools (both local and web based). We used Discord for conversations about how we would implement and design the project. Discord hosts several different channels for our project, which are broken up into *general*, *links*, *alternate*, *requirements*, *assignments*, *availability*, *commands*, *zoom*, and *node-diagram*. Each channel serves a vital purpose to our project. *General* channel is where all of our general ideas are discussed and troubleshooting takes place. *Links* channel focuses on keeping quick access to links of all of the documents we’ve found or created ourselves, such as links to code examples, our Repl.it project code and related tests, Github repository, and Google docs. *Alternate* channel allows discussions while someone is trying to read something in *General* and chat at the same without losing the exact spot being read. *Requirements* channel is specific to our project requirement checklist and serves as a way to check in once each task is completed. *Assignments* channel is where we focus on assigning tasks to each team member. *Availability* channel provides our individual availability should we be unable to participate in discussion for a specific day/s. *Commands* channel is our chat designated for commands we’ve used to run our project like npm/yarn start/run/install. *Zoom* channel hosts a chat specific to our conference calls with Dr. Lewis. Lastly, our *node* channel is specific to our node diagrams and work on those. Repl.it was used for live-time editing of code and multi-user editing abilities. Along with being able to edit live with our group, Repl.it is 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 LucidCharts. To run repl.it properly with some examples of React and Node.js, we used to install packages by running “install npm” and additional commands from repl.it. 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, React.js on top of Node.js, MS Visual Studio Code, and several NPM packages.

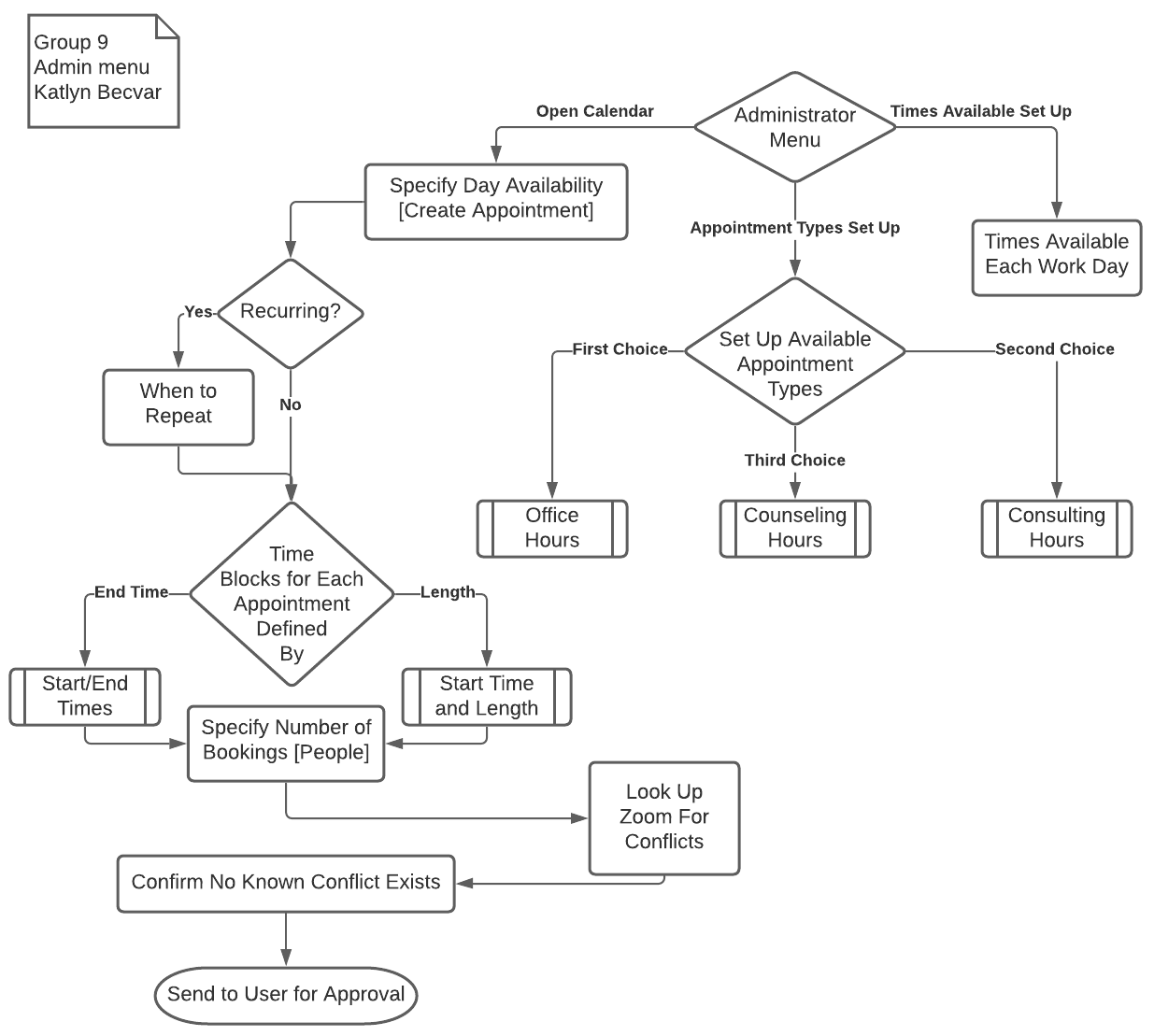
# Database

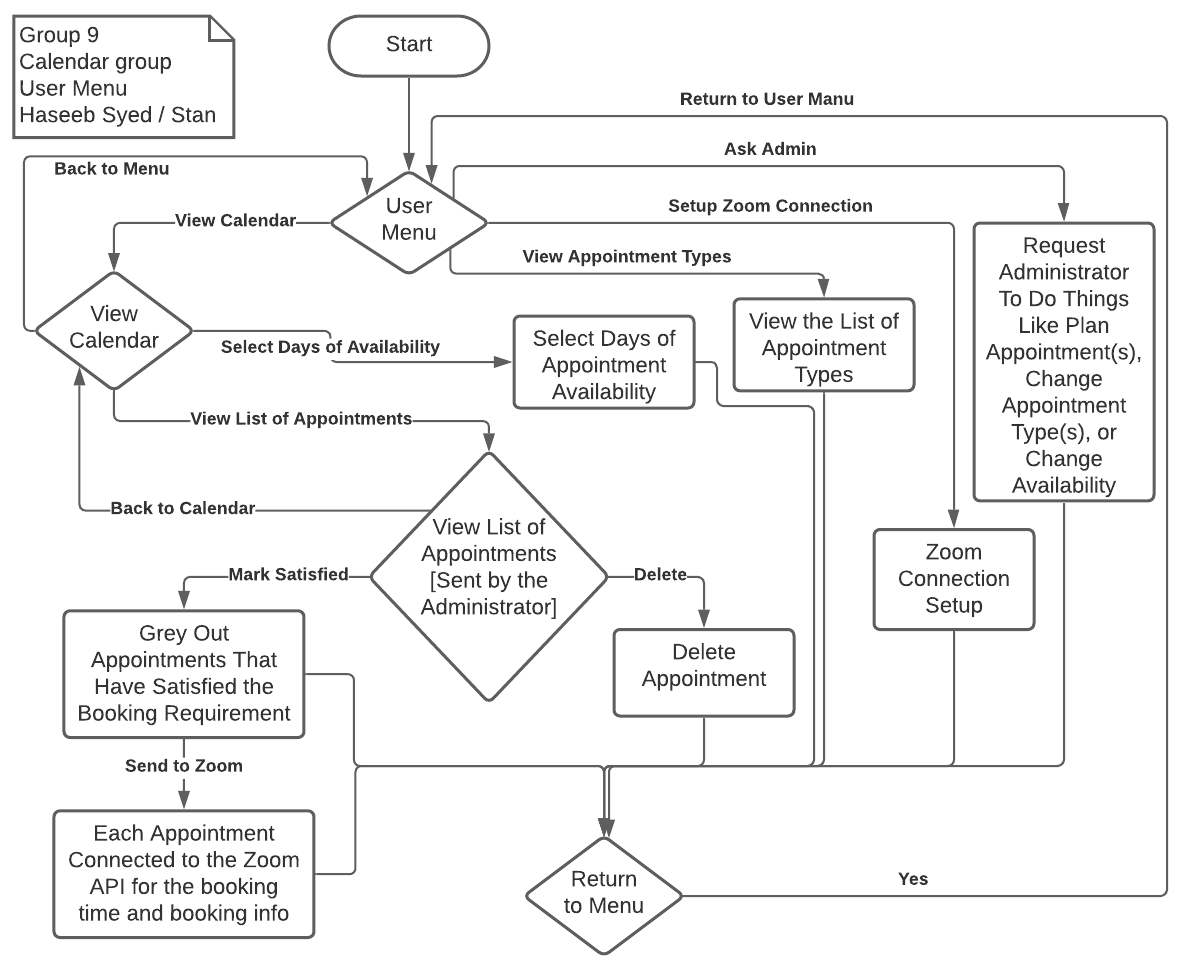
There is no database being used in our project as of right now, but in theory you could use an actual database like MySQL, and its functions are emulated in code. We considered using a JSON file as a data source, but there is no easy implementation of that in React.js vs. Node.js. We thought we could use a JSON file to write usernames and passwords via the “Register” section, and then read that file for authentication in the “Login” section. This could’ve been practical on a small scale, say for a classroom size of 30 students, but if we were to expand to 500+ students, we would need to build and connect to a database. If there was an actual database, the way this would work is detailed in the ER Diagram data model below. The first entity is *Account,* that holdswebsite’s Account, and besides its primary key AccountID, it holds account details such as user’s Name, Email, Password, Entry Creation and Change date/times, Account Type (such as User or Administrator), and 2 foreign keys to *Zoom Account* and *Calendar* entities. *Zoom Account* entity’s, with its self-explanatory name, primary key is ZoomID, and it holds attributes of Zoom’s Name, Account number, Password, and one foreign key to Zoom Calendar entity. There should be none (such as for Administrator) or one (such as for a User) Zoom Account per a website’s Account. Zoom Calendar entity’s, self-explanatorily named, primary key is merged from ZoomCalendarID and DateTimeSlotID, with Event Detail Copy containing virtually all details later emailed as a confirmation, and Link containing a separate copy of the meeting’s link for quick access purposes. Each Zoom Calendar has exactly one Zoom Account. Calendar entity holds the user’s calendar, and its primary key is merged from CalendarID, DayID, and TimeSlotID, with attributes being IsAvailable, ChangedBy, ChangeDateTime, and one foreign key to the Event entity. Calendar can have zero or many Events. Event entity holds event details, and its primary key is EventID, and it holds attributes EventName, event Start and End date/times, and 2 foreign keys to AppointmentType and Recurring entities. Event can have exactly one appointment type and from zero to many recurrences, as well as from one to many attendee reservations. *Reservation* entity is used to combine all possible combinations of events and attendees, and its primary key is merged from foreign keys to Event and Attendee entities. A reservation entry would have exactly one event and one attendee. *Attendee* entity holds all possible attendees, and its primary key is AttendeeID, with attributes of each attendee’s Name and notification Email address. Yes, in theory there could be multiple email addresses, but we are keeping it simple. Each attendee can be linked to from zero or many reservations. *AppointmentType* entity holds all possible appointment types, and its primary key is TypeID, with attributes being appointment type’s name (TypeName) and Description. Each appointment type can be used in from zero or many events. *Recurring* entity holds event recurrings, its primary key is RecurID, and its attributes are RecurUnit (such as a month or a week) and SelectedDays (such as 2 for “every 2nd day of the month” if the recurring unit is “month”).  


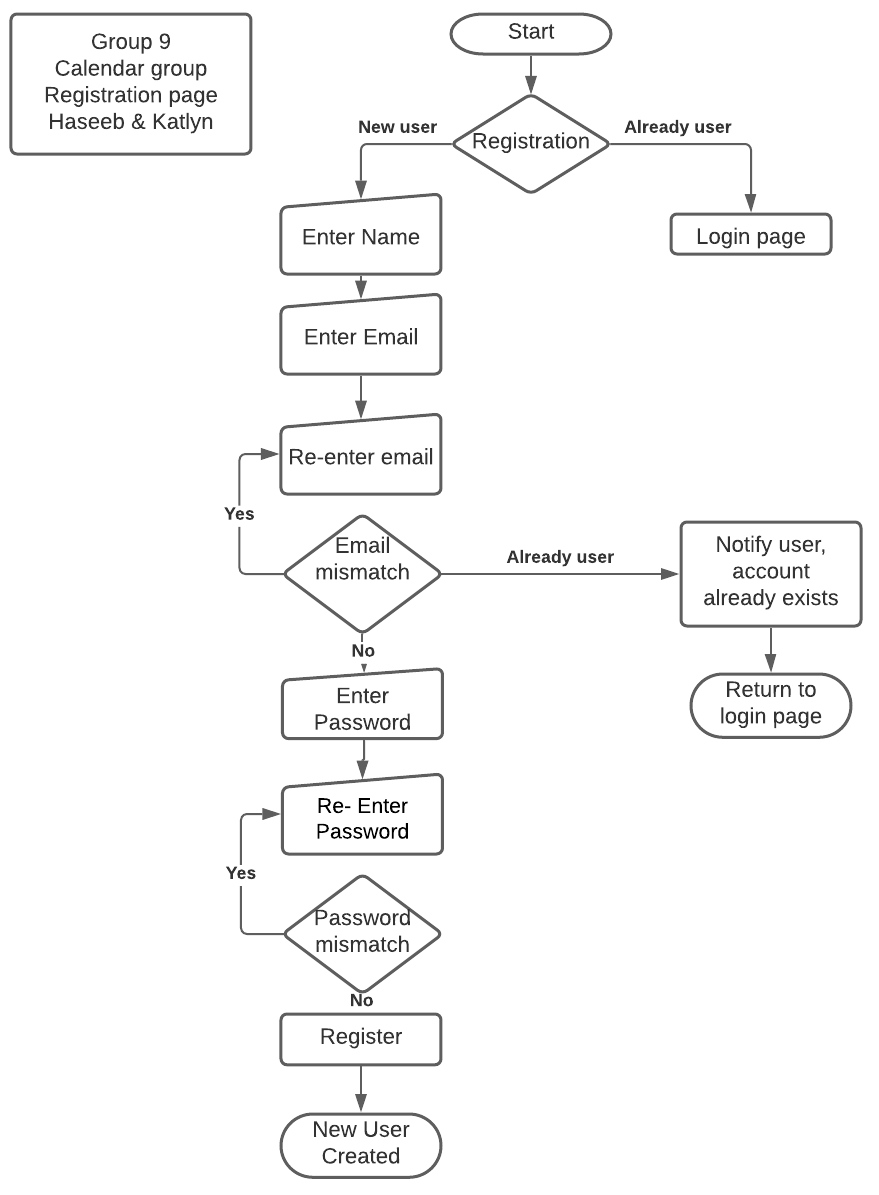
# Application Workflow

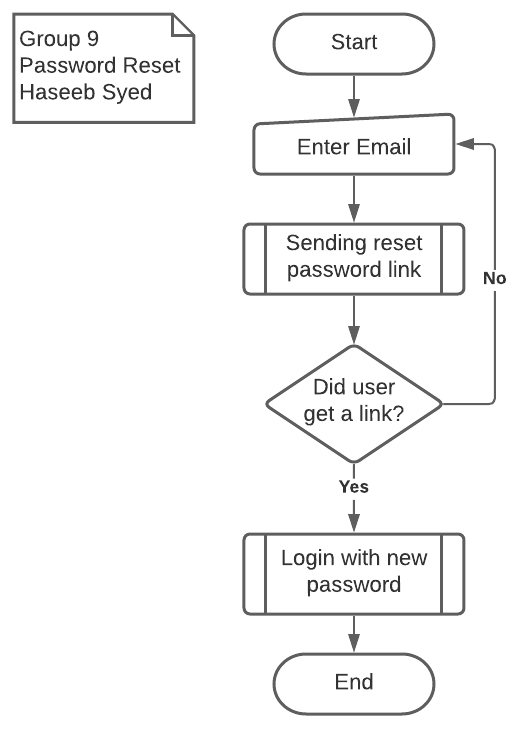
Application Workflow diagrams were originally designed in LucidChart format, and are embedded here as images. They will also be included separately, as PNG image files, as part of the project manual’s submission. Diagrams will start with Overview, then continue with Admin/User parts, Registration, Password Reset, and they finally end with Zoom API Setup.

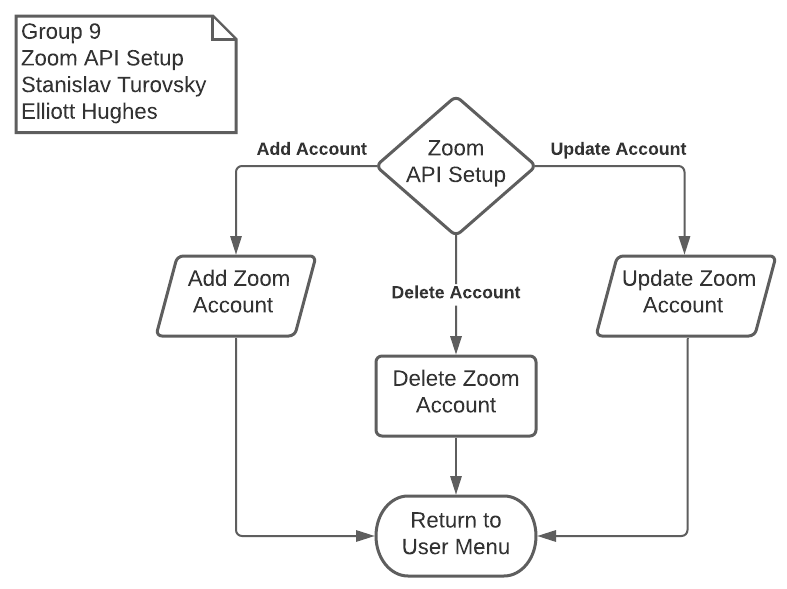












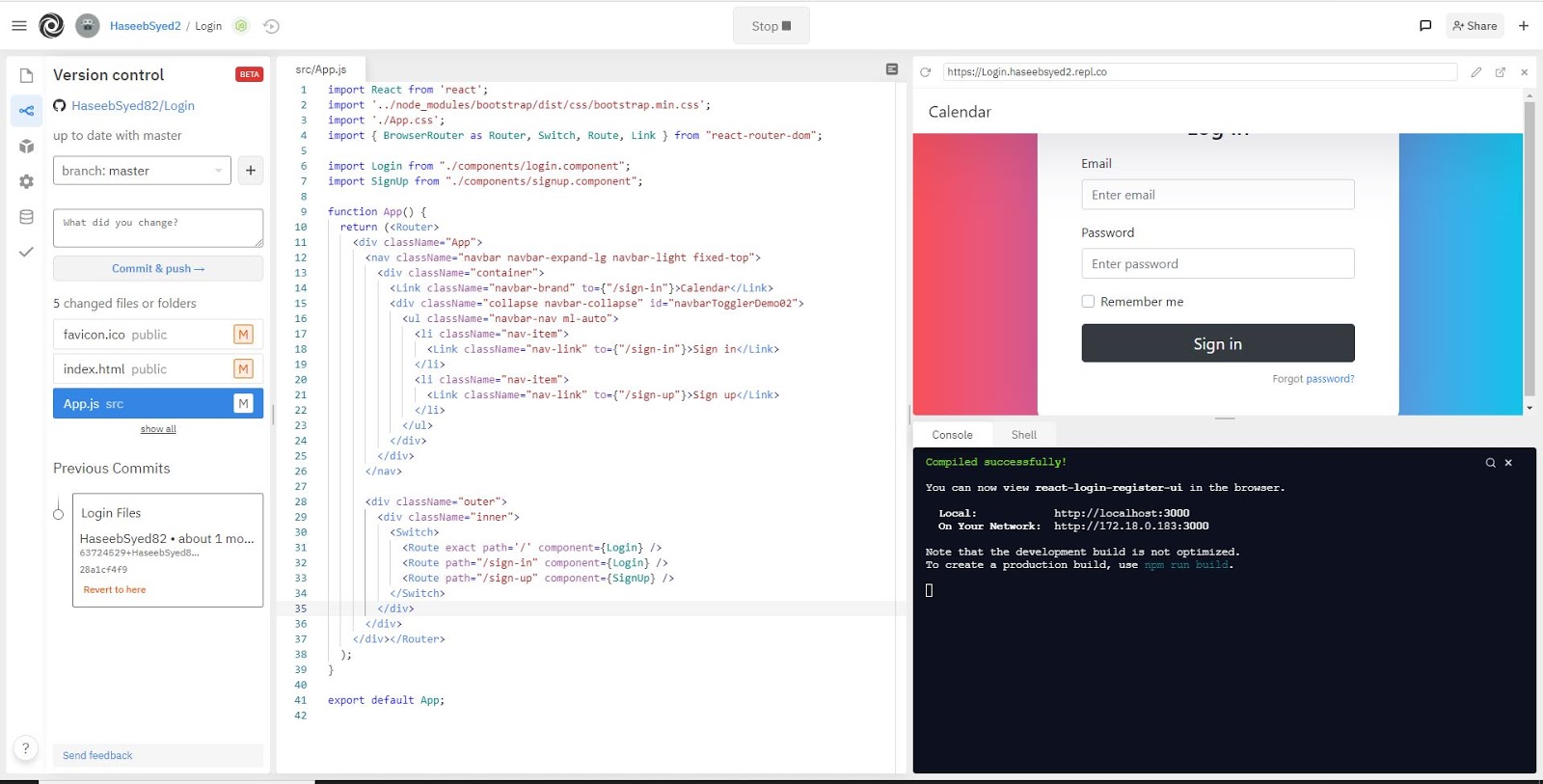
# Features Table

Features table is shown below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Description** | **Software Used** | **Implemented** |
| Admin Part | Functions for the host, who will assist the User | React.js | Partially |
| Appointment List | Allows user to view, edit, and delete appointments, as well as adjust availability | React.js | No |
| Appointment Types | Sets up names and time difference between appointment types | React.js | No |
| Date/time picker | Allows the user/admin to select a time/date for each appointment | React.js | No |
| Login System | Login for the user(s) and Administrator(s) | 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 | React.js | No |
| Registration | Account creation so that a user can interact with the app. Unless we use a database, it will only be shown as a static example. | React.js | No |
| User Part | User related functions, such as viewing appointments and approving meetings for Zoom transfer | React.js | Partially |
| Zoom API | Connects, changes, or deletes associated Zoom account | React.js | No |

# Project Planning

We did create a GitHub account. Actual development was done using Repl.it online IDE. We used GitHub repository to store our project’s code and all related files, such as this manual and its diagrams. The private link for our project’s repository is [github.com/StanTsky/ReactTest](https://github.com/StanTsky/ReactTest). In the beginning of the project, for the first few weeks, we were using VS Code for editing our code, but due to issues like working at the same time, compiling, and having to separately upload to our GitHub Repo, we switched to the online compiler Repl it. VS Code was good for working individually and it also had an option for VS Code Live, where we could work as a team, but it had code compiling issues when used with multiple machines. Due to these issues we switched to Repl.it, which is an online compiler that allows you to see/work in real time with your teammates. The link to our Repl.it code is<https://replit.com/@StanT/ReactTest#src/App.js>. Repl.it allows you to see changes you make as you edit/add your code. You can also link your code to a GitHub repo, so that you can push, commit and go back to the last change within Repl.it workspace, without having to go to a different app for pushing your work to  GitHub repository. Here is an example of Repl.it workspace:



Some of the tasks that were carried out in the past few weeks were creating a login page, role-based menu system, and elements of a calendar function page. Team member Stan and Elliot were responsible for creating a login page, which was completed around week 7. Other members Kate and Haseeb were responsible for creating a calendar page which was created and was able to run on their local pc, but due to package installation issues we were not able to run calendar it on our Repl.it online IDE. We will continue trying to figure out a way to run calendar pages on our Repl.it page at a later time.

# Test Planning

Our test plan is to simply try running the project as both a User and then as an Administrator. We will keep testing our final website, particularly focusing on it during week 14 and 15. As of now, we have done testing on our existing login page, where we tested logins as the Administrator and logins as a User. 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 if you anything extra, and it will just take you to the default User menu page. Login data is currently hardcoded, so database functionality can’t be tested.

# Conclusion

We accomplished what was reasonably possible for our abilities. Lately we had to focus on the diagrams (both node and data model ones) and the manual itself. We are struggling with implementing any kind of a database including things like JSON, and we are struggling with finalizing calendar related functionalities. These items will need to be worked on after the manual.

# Contributions

Stan - Acting team lead. Worked on editing virtually every chart, particularly the Data Model, Overview, User Part, and the Zoom API Setup. Created most of the development environment including Discord server room and manages everything related to it, created project’s primary GitHub repo and linked it to his Repl.it account, and invited all other team members. Wrote the current flexible Menu System (changing based on a user type) and implemented project’s hardcoded logins. Organizes Zoom mettings and attempts to manage team conflicts. Started initial manual version (short starts), suggested overall manual format, wrote Conclusion and edited everyone else's manual parts.

Kate - Created the initial Discord server room between the two calendar project groups. Researched interactive calendars and created a version of the Calendar module for the project (which works in a local IDE but not in Repl.it). Made the Administrator node diagram. Helped with checking everyone else's graphs and code. Worked on the project manual's Proposal and Requirements sections and approved the final project manual edit. Assisted with clarifying work for the fellow team members.

Haseeb - Worked on the manual’s Application Workflow, Features Table, Project Planning, and the Test Planning sections.

Elliot - Worked on node graphs, editing others’ graphs and making suggestions on better shape usage. Worked on manual in database and features section.