

CS-TUTORING

User Manual

Technical Documentation

Author: Serdar Orazmamedov

Project: Master's Project

Northeastern Illinois University
Spring 2022

Preface

The purpose of this document is to describe the application (software) to assist users and for those who might be interested in expanding upon or adding new features to this project. Throughout this document words such as **application, software, web application, web app, system** will be used interchangeably. All refer to a program that was developed by Serdar Orazmamedov as a Master's Project. This document also assumes that the end user of this software has the basic understanding of how to use a computer, navigating in a web browser environment.

This document will be broken into four parts.

1. Introduction
2. How To's
3. Technical
4. Conclusion

As the name implies, the second part is intended to guide the end users of this web application. Although it is not limited to the following listed items, these are important parts of the web app in order to be able to use it.

- How to sign in (authenticate)
- User types / roles

After setting up the ground knowledge about how to sign in, user types / roles and who is authorized to do what, we will dive into use-cases such as:

- Navigation
- Updating profile information
- Checking available tutor schedules
- Creating appointment slots
- Booking / Canceling appointment

Technical part of the document is intended for developers either are going to expand on the project, bug fix or simply add new features to suit new needs that may arise later on.

This part will start with explaining the overall structure of the system as a whole such as how things are glued together, tooling etc. that were used and then goes into the steps of how to install the application.

TABLE OF CONTENTS

Preface	1
Acknowledgement	3
Introduction	4
How To's	5
1. Public Page	5
Google One Tap Sign-In	6
Schedules Table	7
Announcements Table	7
2. User Types	8
Admin	8
Tutor	9
Student	9
3. Home Page	9
Appointments	10
Schedules	11
Tutors	12
Timeslots	14
Announcements	15
Courses	16
My Calendar	17
Semesters	19
Technical	20
1. Front-End	21
Folder Structure	21
Sign In	22
Project Structure	23
Home Component	24
Calendar View	25
Profile	25
Semesters	25
Tutors	26
2. Back-End	28
Folder Structure	28
Clean Code Architecture	30
Demo Scenario Steps	31
3. Installation	32
Front-End (cs-tutoring-webapp)	32
Back-End (cs-tutoring-api)	32
Conclusion	33

Acknowledgement

I would like to thank Dr. Trana for everything she has done for us, students. Also, personally I would like to thank her for accepting to be my advisor during the Master's Project. It is such an honor for me.

Professor Peter Kimmel was and is a big help when it comes to providing feedback about the project, guiding and knows exactly what he wants. So, thank you to Dr. Kimmel.

Also, I would like to express my gratitude to Assistant Professor Ahmed Khaled for accepting to be my new co-advisor even if it was the middle of the semester.

And of course, thanks to Assistant Professor Manar Mohaisen for accepting to be a 3rd committee member on my Master's defense presentation.

The assistance provided by my dear friend Nazar Narmamedov ([email](#)) in designing the user interface of the project means a lot to me. Without him I would not be able to come up with such a simple, yet elegant design.

I wish to show my appreciation to Dave Gray ([github link](#)) for sharing his knowledge. I learned a lot of advanced ReactJS topics from Dave in a very short amount of time.

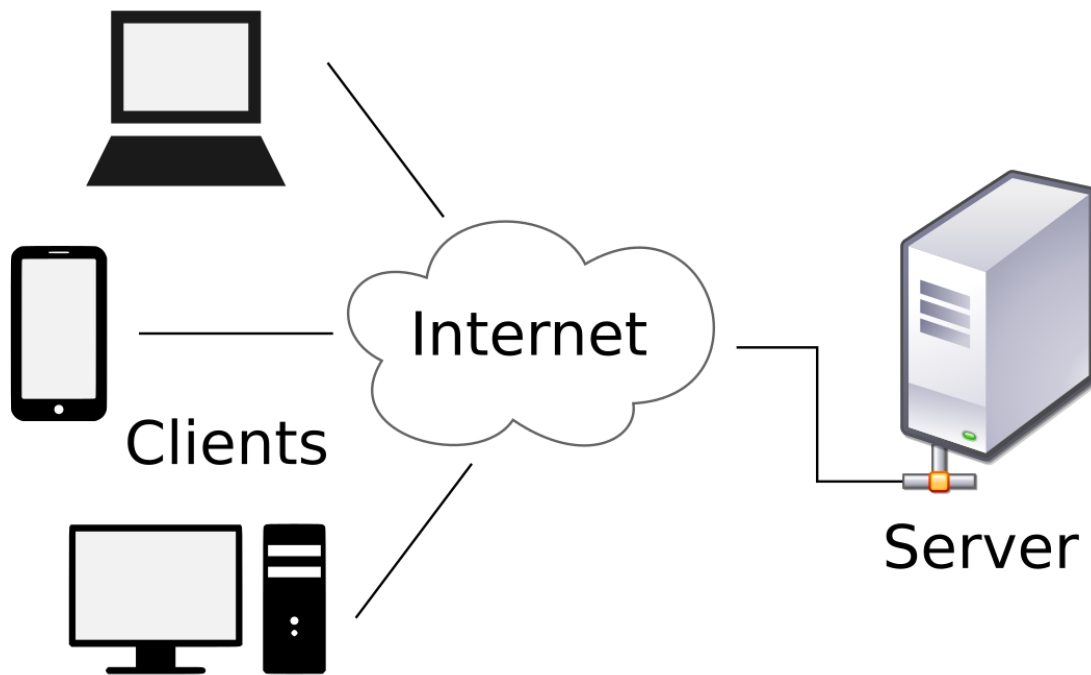
Last but not least, I would like to thank Bill Sourour ([github link](#)) for sharing invaluable knowledge about Clean Code Architecture with a demo and how to improve as a developer.

Introduction

CS-Tutoring is a client-server model, or client-server architecture, application. Meaning it consists of two parts: server and client. One of the advantages of building a software in client-server approach is that the back-end (server) is not tied to the front-end (client).

In this scenario a client could be an app that is running on a mobile device or a desktop application.

Figure 1: Client-server model



source: [wikipedia](https://en.wikipedia.org/wiki/Client-server_model)

The server is the back-end of the CS-Tutoring that is responsible for delivering information requested from the front-end, the client. In our case the front-end happens to be a web based application which runs in a browser. From the beginning the idea was to develop one back-end so that in the future if the need arises for mobile application any potential developer could build the app without a need to develop the server part from scratch or to rewrite it.

How To's

In order to follow this tutorial things you will need:

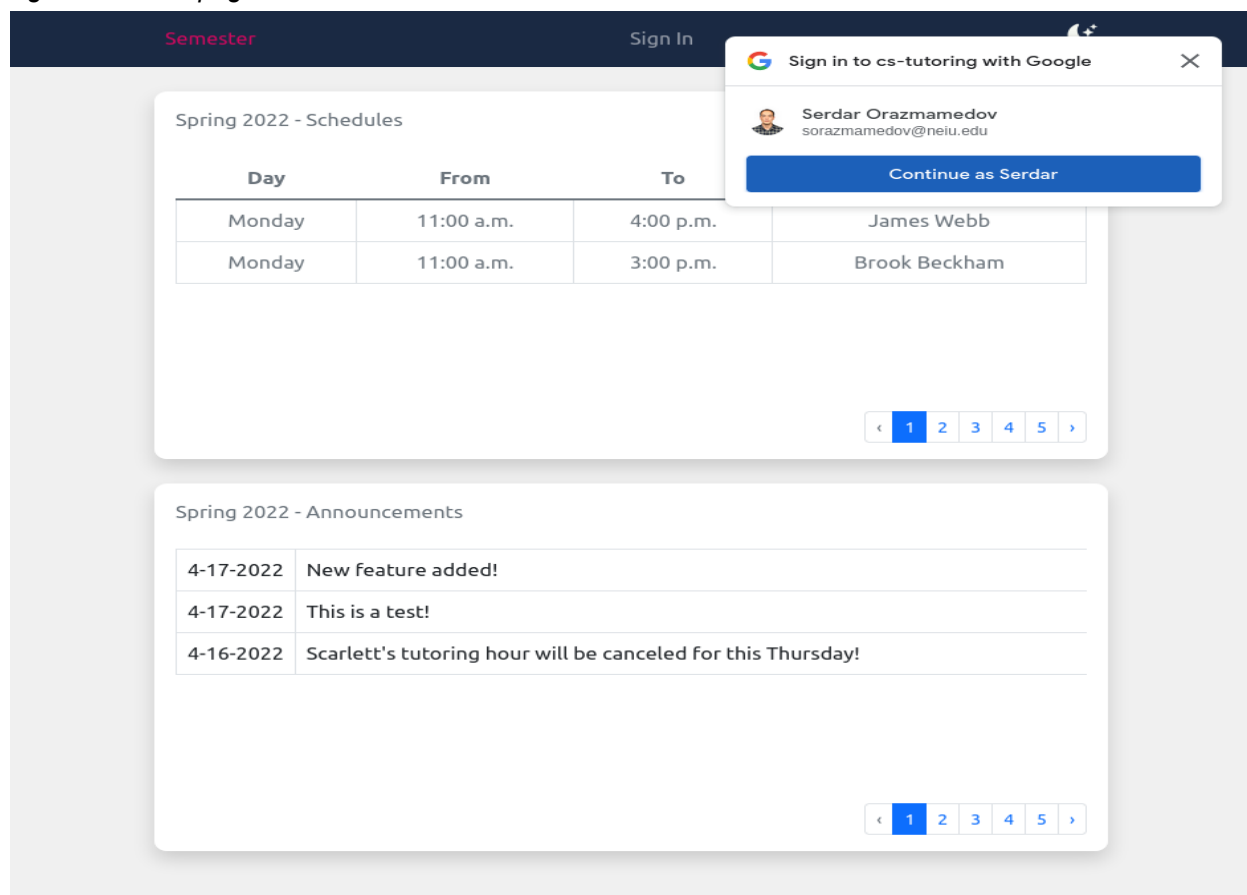
- Computer
- Internet connection
- Web browser of your choice (Google Chrome is recommended)
- Email address that ends with neiu.edu: **example@neiu.edu**

If you have everything listed above you can move onto the next step.

1. Public Page

Whenever you visit the web page of the application (ask the admin for the website's link) you will be greeted with what is referred to as a "Public page" which is shown below.

Figure 2: Public page



Without signing in, one could get a glance of currently available tutor schedules and latest announcements that were posted by the administrator.

Let's break it down by elements that we see in the image above:

- **Google One Tap Sign-In**

Upon page load a pop-up will appear in the right top corner or at the bottom of the screen depending on size of the display. The pop-up is known as Google One Tap Sign-In. In order to sign in, the user has to authenticate with Google using NEIU email address. Once you are authenticated the system will let you in. Next time you visit the site if the account you chose to sign in happens to be the only session in that specific browser it will automatically sign you in unless you click the cancel button which will show up in the same pop-up.

If you choose to not sign in and click the "X" button next time you visit the page or if you reload it, a pop-up may or may not appear depending on how much time has passed since you have clicked the "X" button. Google names it as a "cool of period" which will not show that pop-up until some time passes (controlled by Google).

Regardless of if the Google One Tap appears or not, or if you close it, you can always sign in to the system by clicking on "Sign In" on the navigation bar. Which will in order show you a pop-up where you can choose your Google account.

When the user chooses to sign in, Google will ask for your consent to share information with the "CS-Tutoring" app. Application needs only basic information from Google and below what Google share upon consent:

- First and last name
- Email address
- User's profile picture url
- Verified email or not etc.

The system only keeps the first 3 items which are used throughout the application. One example is the profile page. (See profile)

- **Schedules Table**

The table shows information about the currently available schedules. From the image above the title of the table reads "Spring 2022 - Tutoring Schedule" which tells that the currently active semester is Spring 2022. There could be a situation where upon visiting the page just to check the schedules table will have a text something along the lines of "No schedule available at this time!". This could mean 2 things:

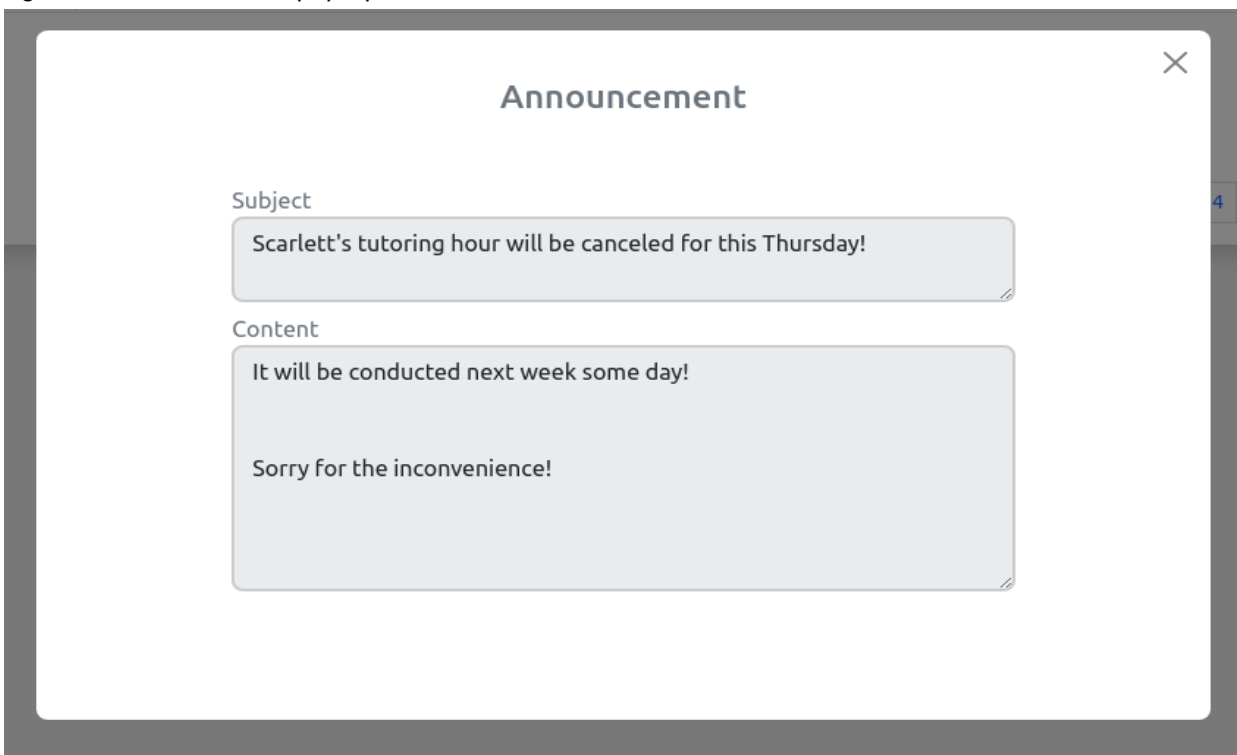
1. If the title does not have semester name then there is no active semester yet, hence no schedule
2. Title has a semester name but the table empty then it simply could be interpreted as no schedules were entered yet by the administrator

- **Announcements Table**

This is where one could get the latest announcements that were posted by the administrator. It could be anything from informing about the possible cancellation due to some holiday or tutor will not be able to make it or anything that needs to be announced.

Announcements will be sorted by date in descending order, latest will be on top. Unlike in the schedules table, rows in the announcements table are clickable. On click one should see a pop-up with subject and content fields.

Figure 3: Announcement pop-up



That is it for the public page where there is no signed in user. Header and Footer sections are self explanatory. The only thing left to mention is the icon that you can find in the navigation bar (header) which is used to toggle between “dark” and “light” themes and is available throughout the application.

2. User Types

Before continuing any further we need to cover user types, what each user will have access to on the home page. There are 3 types of users:

- Admin

Home Page:

- Semesters - can add/edit
- Schedules - can add/edit
- Tutors - can add/edit
- Timeslots - only cancel booked slot
- Announcements - can add/edit
- Courses - can add/edit/delete

- Tutor

Home Page:

- Appointments - can edit
- Schedules - view only
- Timeslots - can cancel own slots
- Announcements - view only
- Calendar - can add/delete calendar

- Student

Home Page:

- Appointments - can cancel booked appointments
- Schedules - view only
- Timeslots - can book a slot for appointment
- Announcements - view only


Profile Page:

Every user has their own profile pages regardless of a role. It is only allowed to change “Pronouns”, “NEIU ID” fields. A user with a “Tutor” role will have an extra field called “Bio” which they can use to share useful information about themselves. Such as courses taken, hobbies etc. this information will be available to a student to see. For instance, if the tutor has shared the courses taken it might be helpful for a student while booking an appointment.

3. Home Page

Once you sign in, you will be redirected to the home page where you can see more tables and information. The information, tables and actions that could be performed in the system will depend on the type of a user as mentioned in the [User Types](#) section.

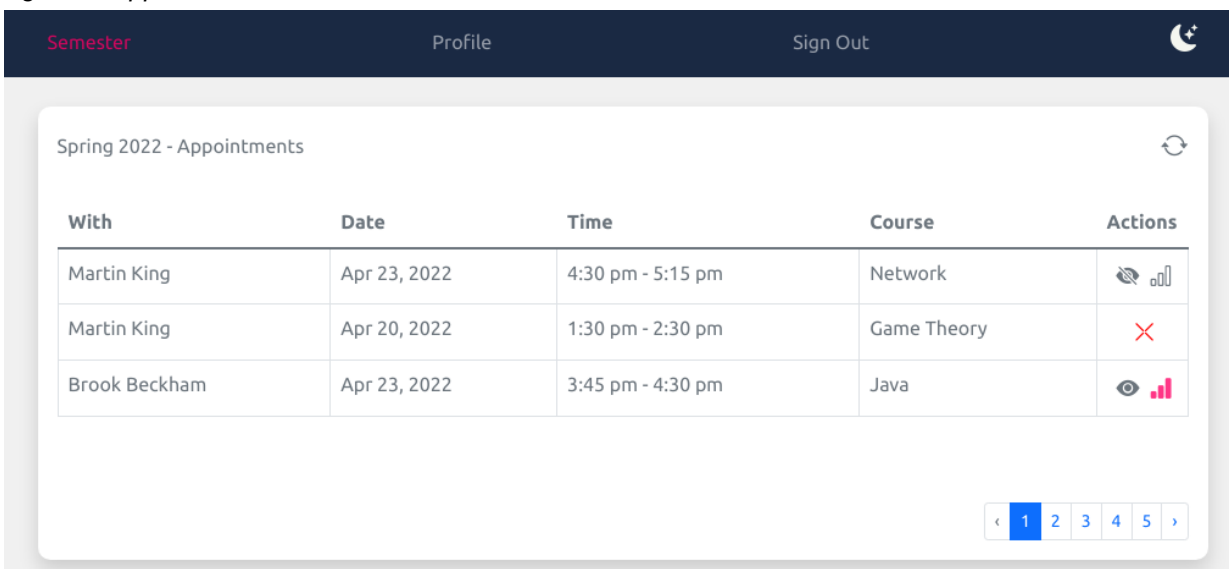
Next, we will take a look at each table one at a time. Since we briefly mentioned who is authorized to perform what set of actions on a specific table in the [User Types](#) section we will not be covering each table for each user type. Instead, it will be a guidance of important things to know about the table (with a screenshot) and actions.




NOTE: User interface was designed to be simple and intuitive. Therefore, if something is not mentioned or explained in detail something similar in functionality must have been mentioned in the documentation. For instance, icons have the same role unless it is specified. Take  icon for example, it has the same functionality in every table, on click it will either reveal a pop-up where you can enter information or add some row where you can add schedule etc.

- Appointments


Find everything about appointments in this table. Booked, canceled and appointments where a student did not show up. In this web app tutors are allowed to book appointments from other tutors if they need help with a class where other tutors might be better at.

Figure 4: Appointments Table





With	Date	Time	Course	Actions
Martin King	Apr 23, 2022	4:30 pm - 5:15 pm	Network	
Martin King	Apr 20, 2022	1:30 pm - 2:30 pm	Game Theory	
Brook Beckham	Apr 23, 2022	3:45 pm - 4:30 pm	Java	



Besides showing information about the appointment, there are some actions that could be performed on this table.


Student will only have  icon in the table which serves a purpose to cancel the appointment if they can't make it or for some other reasons.

Tutor will have all the icons that are shown in the table. As we mentioned before tutors are also welcome to book an appointment in case they need help with some classes.

Tutor

 an icon that looks like an eye shows that the student did show up for the booked appointment. In the case of no show, the tutor can mark the appointment as one. On clicking the icon it will turn into this  icon.

 an icon that shows if a report was written about the appointment. Upon clicking a pop-up will appear in the middle of the screen where contents of a report could be typed in. The written reports are sent, *once a week*, to a related faculty member that the student came in for tutoring. Tutor can mention topics they discussed with the student, areas the tutor thinks that the student could improve etc. Whenever the report is written the icon should turn into . Reports could be edited until the sending date, which is typically on Saturday evening of every week.

 the purpose of this icon to refresh the table. Instead of reloading the whole page to fetch the latest appointments, one could just click on the icon. If fetching the data takes longer than usual, the user might see the icon turning into pink color and rotating.

Moreover, upon hovering over the icons a tooltip with a short text should appear which hopefully will be useful in recalling the purpose of the icon.

- Schedules

The only difference of the table compared to one in the [Public Page](#) => [Schedules Table](#) subsection is that once there is a signed user, who does not have an admin role, it will show an extra column where you can find a “Zoom Link” if one was provided.

Figure 5: Schedules-Admin

Spring 2022 - Schedules



Day	From	To	Tutor	Zoom Link	Actions
Select ▼	<input type="text"/>	<input type="text"/>	Select ▼ Select Martin King Brook Beckham James Webb	<input type="text"/>	
Monday	11:00 a.m.	4:00 p.m.		James' Zoom Link	
Monday	11:00 a.m.	3:00 p.m.		tba	

click to add a new schedule. To be able to add a new schedule there has to be at least one active tutor in the given semester. As you can see the row has a populated drop down list of tutors where you can select from. Other fields that are not populated are required. If you do not provide them it will show a pop-up message complaining about it. Format for *From* and *To* fields is “11:45 a.m.” or “2:15 p.m.”. If at the time of creating a new schedule a zoom link is not available you have to provide some text of length at least 2.

click to edit the schedule. Upon a click table again will reveal an editable row with populated information. But this time instead of this pencil and status icons row will have green checkmark and cancel icons, one for saving and the other for canceling the changes, respectively.

shows the status of a given schedule. This is an active state, meaning other users, who don't have admin roles, will have it on their schedules table. Clicking will put into inactive status . As you might guess if the row has this icon, that schedule will neither be available to the public page nor to the “Tutoring Schedule” table..

• Tutors

Table for managing tutors for the chosen semester. Once a tutor is activated in this table, the schedules table will have access to that tutor information. To add a new tutor simply click on the plus icon which will trigger a pop-up where you can search for a user in the database with an email address. If the user exists in the database you will see a screen similar to the below screenshot. You can add by clicking the add button which will in turn add the user to the current tutors list of the given semester. By default, the newly added tutor is inactive.

Figure 6: Tutors-Admin-Add-Tutor

The screenshot shows a web application interface with a modal window titled "Add New Tutor". The modal is centered over a background that appears to be a calendar or scheduling interface. The modal contains a search bar with the text "jenmonroe" and "@neiu.edu" entered, followed by a blue "Search" button. Below the search bar is a large, light gray placeholder for a profile picture. Underneath the placeholder, the name "Jennifer Monroe" is displayed, followed by the ID "000676877" and the email address "jenmonroe@neiu.edu". At the bottom of the modal is a blue button labeled "Add as Tutor". The background interface includes a header with "Monday", "11:00 a.m.", "3:00 p.m.", "Brook Beckham", and "tba". On the left, there is a list of IDs: "000123", "000777", "000789", "000789", and "000546". On the right, there is a "Status" section with toggle switches and a "Refresh" button. At the bottom, there is a calendar grid with days of the week and dates.

Monday 11:00 a.m. 3:00 p.m. Brook Beckham tba

Add New Tutor

jenmonroe @neiu.edu Search

Jennifer Monroe
000676877
jenmonroe@neiu.edu

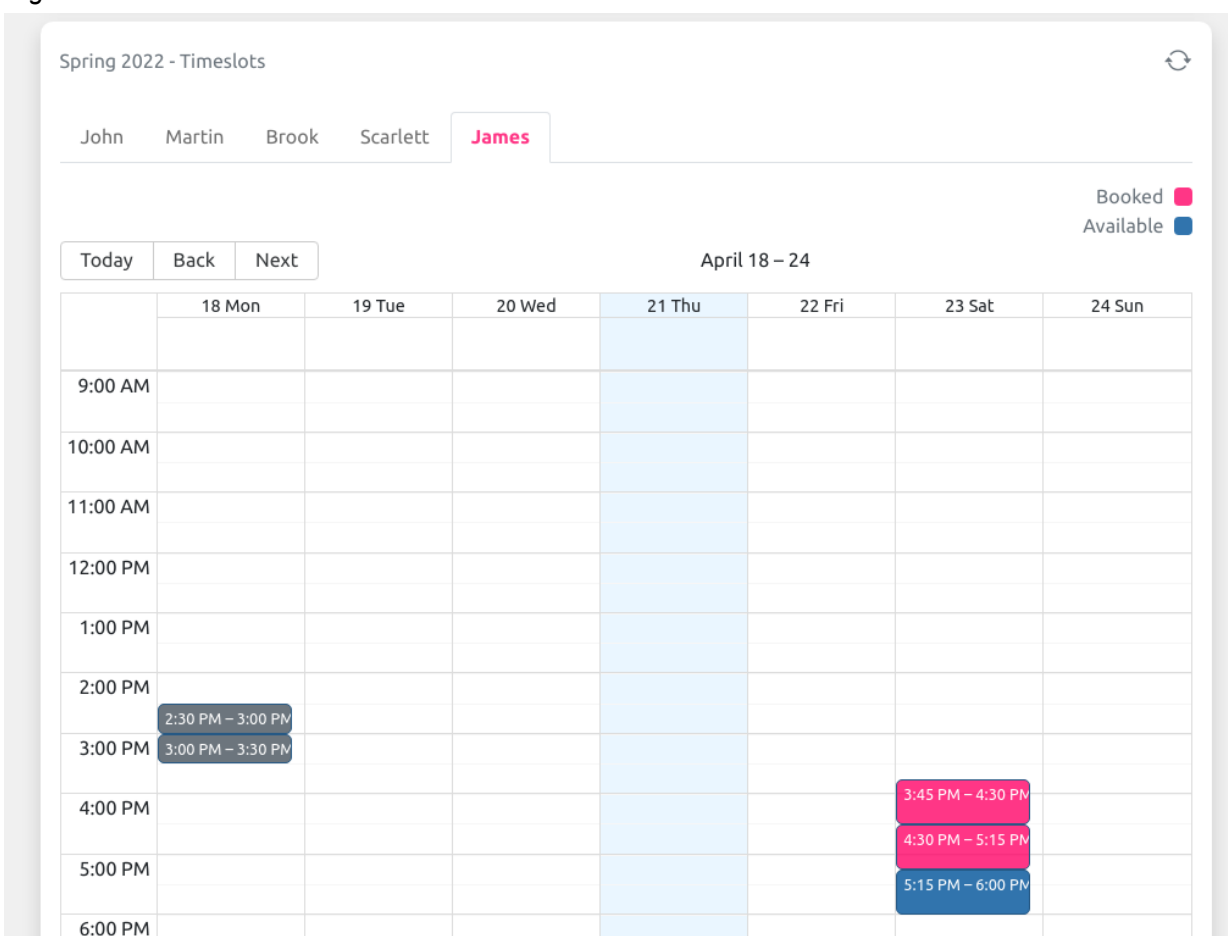
Add as Tutor

Only type in the username portion (before the @ sign) of the email address in the search box.

• Timeslots

This is where the booking, canceling of appointments happen. Regular users, meaning students, can only cancel the appointment they booked in their appointments table. Tutor and Admin on the other hand are allowed to cancel an appointment from the Timeslot calendar view. As you can imagine, the tutor is only allowed to cancel slots that belong to him/her. Admin should be able to cancel any booked slot. Upon cancellation both tutor and student will receive email notification about the cancellation of the appointment.

Figure 7: Timeslots



To book an appointment:

1. Select tutor from the top tabbed view
2. Selected tutor's available time slots will be show on the calendar
3. Pick the date and time (Back and Next) that you like
4. Pop-up will be triggered upon selecting the slot
5. Search the class you are coming for tutoring
6. Select the correct class (section and instructor)

7. After you select the class, a brief summary will be shown regarding the booking. Make sure it is correct
8. Confirm booking
9. If the booking is confirmed
10. Refresh the appointments table to see the booked appointment

- **Announcements**


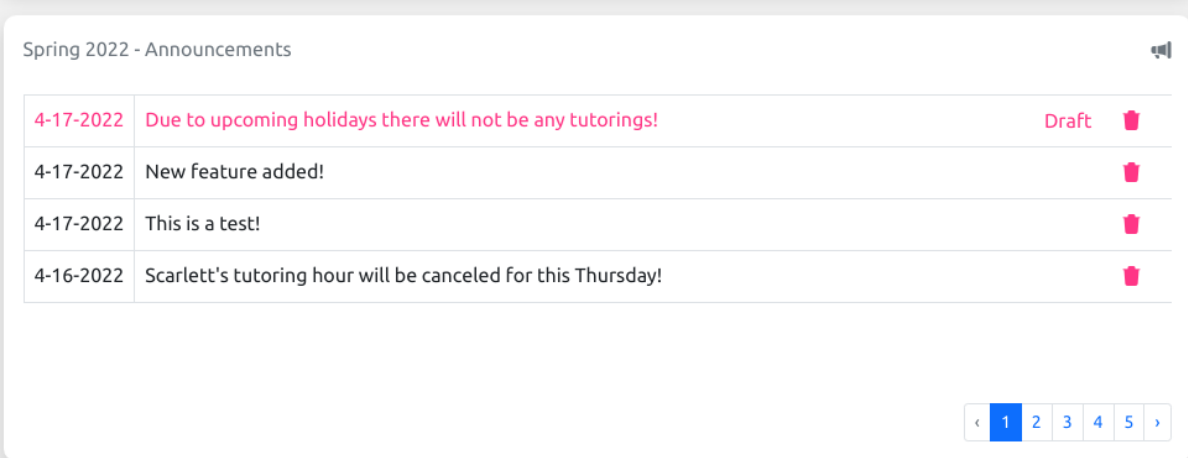




Same table as in the public page. Admin can post a new announcement by clicking the  icon on top right corner. Which in turn, again, triggers a pop-up where you can enter the subject and content of the announcement. Once you are done entering all the details you can choose to save it as a draft or publish it. If you choose to publish the announcement will be available for the rest of the users. On the contrary, if you go with draft, the announcement will be saved as a draft and won't be accessible by others. Announcements that were saved as a draft will be colored as pink with the text at the end of the row saying "draft" as in screenshot below:

Figure 8: Announcements-Admin



4-17-2022	Due to upcoming holidays there will not be any tutorings!	Draft	
4-17-2022	New feature added!		
4-17-2022	This is a test!		
4-16-2022	Scarlett's tutoring hour will be canceled for this Thursday!		

Page 1 of 5

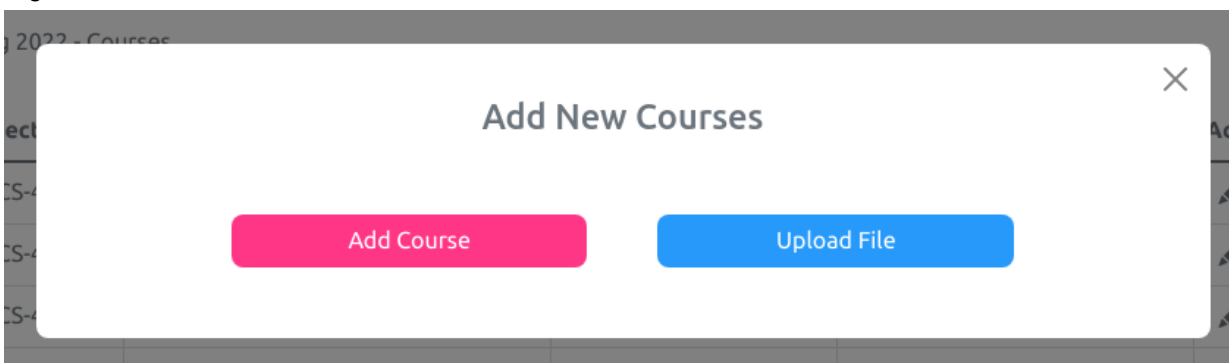
At any time admin can click any of the announcements to publish or to save as a draft or delete it (after confirming) by clicking the trash icon. As mentioned before, keeping the consistency of the colors, actions etc. makes things intuitive and easy to use and requires less explanations.

- Courses

The purpose of this table is to add courses that are being offered in the given semester so that when booking an appointment a student can choose which particular course he/she is coming for tutoring. As a result, the system will have the information of the faculty member so it knows where to send the report that was written by the tutor at the end of the week.

To add a new course or an excel file that contains a list of courses click the plus icon. Select the action you want to perform on the pop-up. For a single course select the “Add Course” button, for a file upload click the “Upload File” button.

Figure 9: Courses-Admin-Add-New



Once you select, a form will appear where you can fill in the information about the course and submit. Same goes for the upload option. Only “.xlsx” formatted files are allowed.

The rest is the same as were talked about in other sections of this document. Click the pencil icon to edit, trash icon to delete etc.

- My Calendar

This is where tutors create their time slots that you saw in [Timeslots](#) calendar. To create a calendar simply with a mouse or touchpad start dragging from the time you want the slots to begin and release when you want it to end. For instance, say you would like to create a calendar from 11:00 am through 3:00 pm. You start dragging from 11 am and release it at 3 pm. Which show a pop-up like below:

Figure 10: My Calendar-Add-Slots

Appointment Slots

Choose:

Apr 20, 2022

11:00 am - 3:00 pm

Slots with duration

30 minutes

Repeat until:

Does not repeat

Preview:

Slots:

1) 11:00 am - 11:30 am

2) 11:30 am - 12:00 noon

3) 12:00 noon - 12:30 noon

4) 12:30 noon - 1:00 pm

5) 1:00 pm - 1:30 pm

6) 1:30 pm - 2:00 pm

7) 2:00 pm - 2:30 pm

8) 2:30 pm - 3:00 pm

CANCEL

SUBMIT

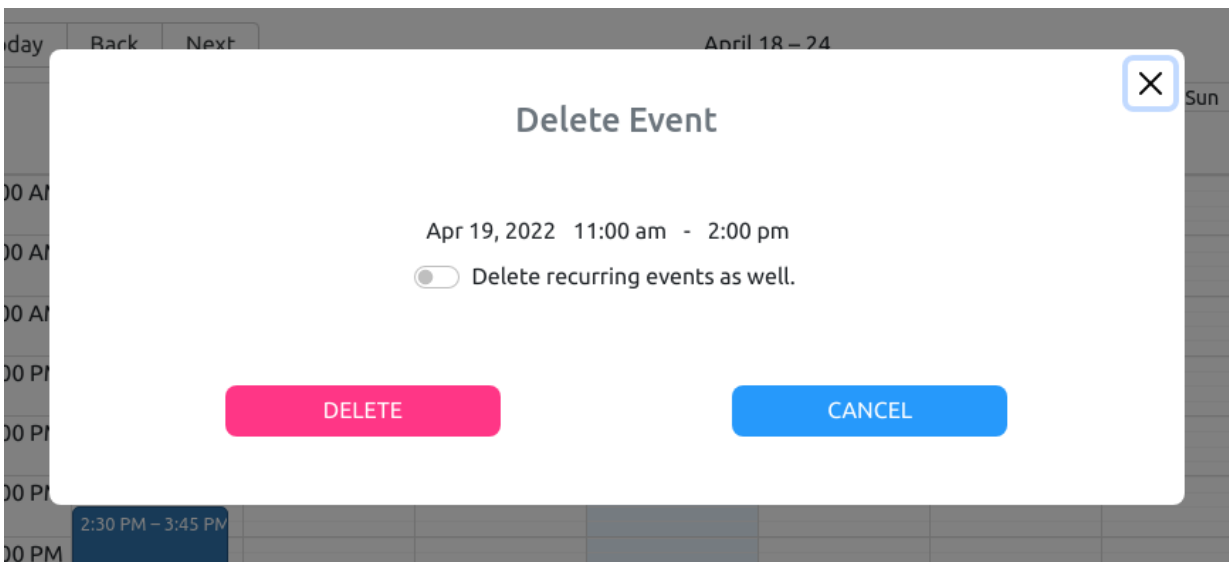
Left side is where you configure, fine tune duration of a slot, if it's a repeating event or not. Very much like a Google calendar if you have used one to create appointment slots. The right hand side is there for your convenience. As you play with the left hand side the right part will react to changes. In this case if you were to leave the left side as is, slot duration being 30 minutes, it would create 8 appointment slots. On the calendar it will show up as a one block but in Timeslots table there will be 8 slots.

16

Another important point here is that whenever you try to create a slot outside of the dates of the given semester it will not allow you to.

If at any point you need to make changes, the only way to achieve this is to delete the created calendar. Simply on the calendar view select the one you want to delete. If at the time of creating the calendar you selected the event to repeat, say until the end of semester then you will have the option to delete all the recurring events as well. Take a look at image below:

Figure 11: My Calendar-Delete-Slots

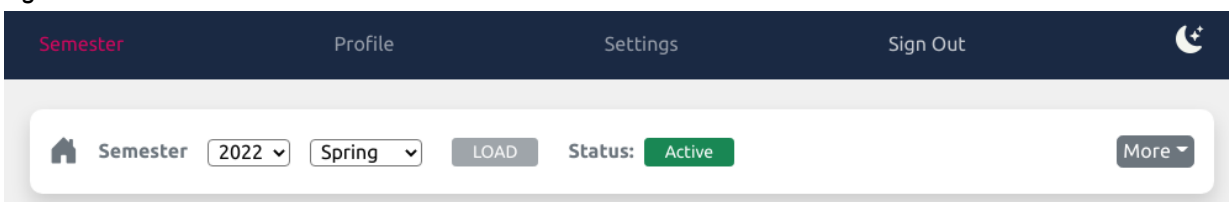


Toggling the switch will delete all the recurring events. It is recommended that you delete individual slots from the [Timeslots](#) table instead of here. As this will create the block of slots not the individual. In the case where a slot is booked, the user who booked the slot will be notified through email.

- Semesters

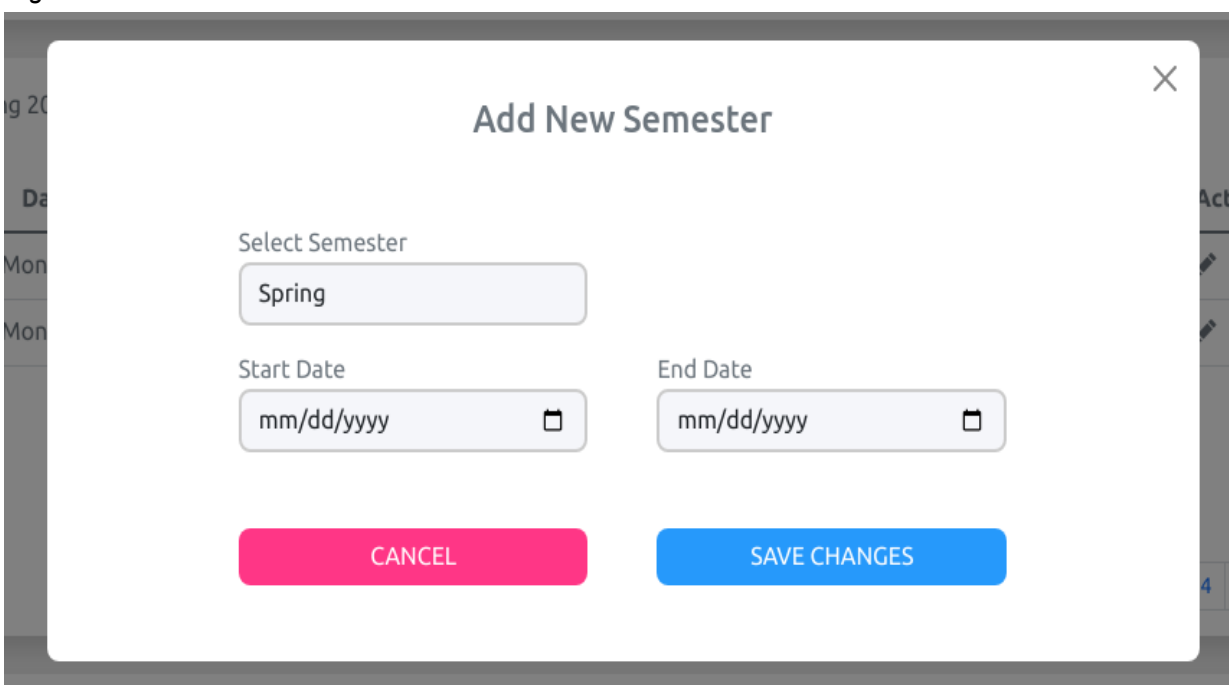
This is the most important part of the application. As everything depends on the configuration that is applied in this section.

Figure 12: Semesters-Admin



At any given time this bar will show the currently selected semester. Unless the load button is grayed out it does not mean that the data on the tables that you see below belong to this semester. To see any particular semester data you have to load the semester first which will trigger all the tables to fetch according data. Also, you should see the title on each table should reflect the name of the loaded semester. Status button as the name suggests shows the state of the semester. If it is active like the one in the picture only then other users will be able to see data if any is available. You can toggle the status button which will put it in “Inactive” or “Active” state. It is allowed to have no active semester but only 1 semester could be active. On the right there is a drop down named “More” from where you can add a new semester or edit currently selected.

Figure 13: Semesters-Admin-Add



Technical

This part will be completely technical. As a result you might expect to see lots of technical jargon. Intention of this section of the documentation is to provide as much information as possible to future students who might consider adding new features, fixing bugs or do some refactoring, clean-up. Hopefully, it will shed some light on overall structure and ease the process of getting started.

As it was already mentioned in the introduction part, this project has a front-end and back-end. We will start off with a front-end first, then dive into the back-end side. Before diving into explanations though, it is important to note the prerequisites to be able to understand easily:

1. Front-end:
 - a. Basic knowledge of web development
 - b. HTML, CSS
 - c. [Bootstrap](#) and [React-Bootstrap](#)
 - d. JavaScript - especially ES6 syntax
 - e. [Axios](#) - for making API requests
 - f. [ReactJS](#) - UI Library
 - g. [React-Router](#) v6
 - h. [date-fns](#)
 - i. [YupJS](#) - for validating data
2. Back-end:
 - a. What is REST API
 - b. [NodeJS](#)
 - c. [ExpressJS](#) - web framework
 - d. [MongoDB](#)
 - e. [Joi](#) - for data validation

At a first glance it might seem a lot but you might already be familiar with some or you might know some other technologies. For instance, if you know Ruby on Rails or Django or Flask, then ExpressJS might make you feel at home. So, do not be discouraged.

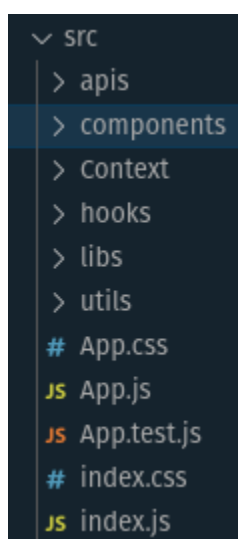
In an unusual way, the process of installation will be kept for the end.

1. Front-End

Interface for the users of this software happens to be a web based application where they can interact with the system. Assuming you have the knowledge or familiarity of the aforementioned topics, next will be the structure of the front-end to show how things are glued together, which parts share the same information among each other and much more.

● Folder Structure

The front-end was created with a tool called “create-react-app” which was developed by Facebook. In a nutshell, it saves time to start building a project instead of spending time on configuration such as webpack, linters etc. If you want to know more about it, please refer to their documentation. To start, the image on the left shows the project folder structure, src folder in particular. Everything regarding the front-end code lives in this folder. Besides configuration files and some tooling specific files or folders that create-react-app creates at the time of initiating a project.



- apis/ - place where you can find axios related configurations. For instance, if the post number or the url of the server has changed this is the place where you update that information.
- Context/ - as you might know to share information between components in react you have couple of choices:
 - Pass it as a prop
 - State management tools like Redux, Flux etc.
 - Or Context API - as will be explained later on, in this project I chose to use the Context API. Looking back I can tell that it might have been a good use case for more advanced tools such as Redux as it gets convoluted with the Context. I hope that you, the reader, will take action on this and reimplement it with such a tool.
- hooks/ - this folder contains all the hooks. Any function that starts with “use” word in front of it will be in here (except reactjs hooks). Things like useLogin, useLogout, useAuth etc.
- libs/ - this one only contains one js file which loads Google’s library that is used for sign in.
- utils/ - as the name implies it contains utility functions or tools. YupJS (for validating data) that was mentioned before lives here. So does the “nanoid” - helps to create unique identifiers.
- App.css - stylings regarding icon colors, text and so on could be found here
- App.js - main component that is rendered under root element which could be found in index.html file under public/ folder.
- index.css - main cleanup of default browser styles, background colors

- index.js - main entry point to the application.
- Sign In

To authenticate users, the application utilizes Sign in with Google. Please refer to its documentation here: [Sign in with Google docs](#). As previously mentioned all the configurations regarding this you can find in:

1. App.js file - inside useEffect hook using the function that resides in libs/ folder and sets up the stage
2. Clicking Sign In on the navigation bar will trigger Login component (pop-up) this where you can change the style of the sign in with google button that is rendered
3. useLogin hook - this is the function that processes the response from google. Also this is the place where it makes a GET request to “/auth/login” endpoint with the token set as Authorization header.
4. Both One Tap and Sign in with Google button use the same *handleResponse* function from useLogin hook.
5. client_id - that is acquired from Google Console will be kept in an .env file. Which will not be pushed to GitHub. So check with the admin (currently that would be Dr. Peter Kimmel) for the information.

tokenId - the token that Google sends back when the user gives consent. This token contains some information that we use:

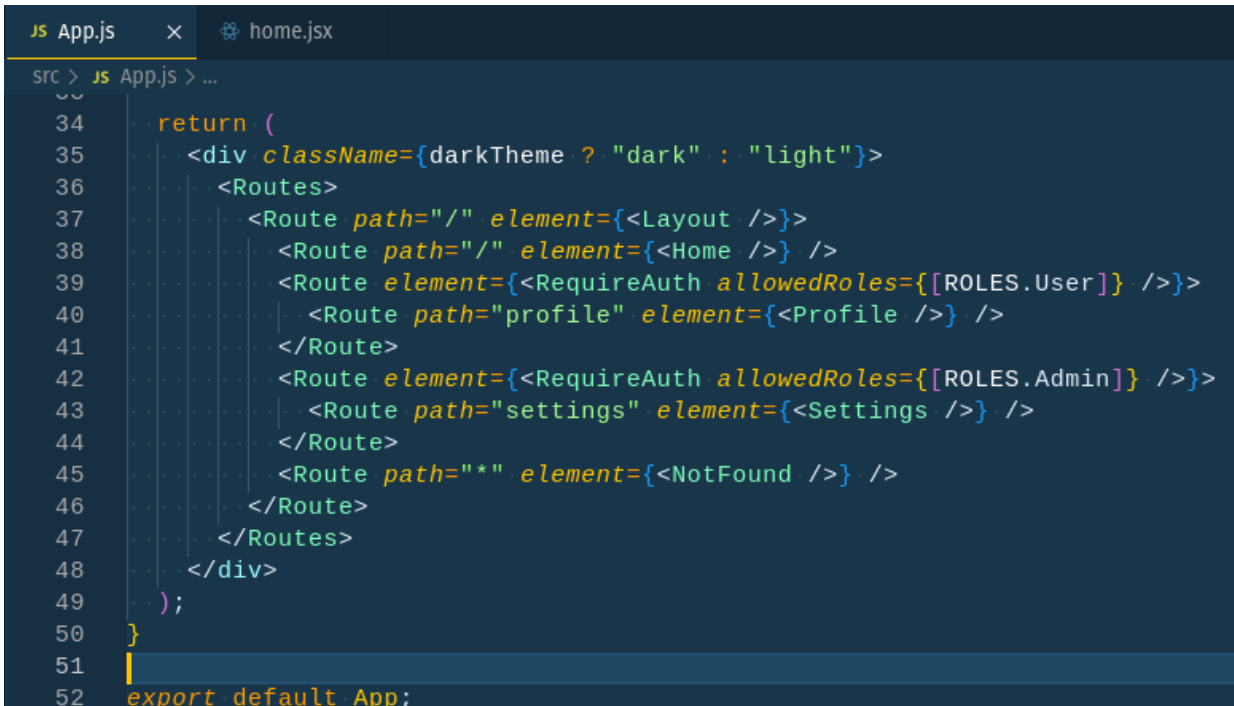
- picture url
- firstName
- lastName
- Email address

and many more fields but we use these fields to save a user into a database if we already don't have one. Refer to above link if you need more information.

- Project Structure

As you can see from the image below it is a screenshot of the App.js file located in src/ folder.

Figure 14: App.js



```
34   return (
35     <div className={darkTheme ? "dark" : "light"}>
36       <Routes>
37         <Route path="/" element={<Layout />} />
38         <Route path="/" element={<Home />} />
39         <Route element={<RequireAuth allowedRoles={[ROLES.User]} />} />
40         <Route path="profile" element={<Profile />} />
41       </Route>
42       <Route element={<RequireAuth allowedRoles={[ROLES.Admin]} />} />
43       <Route path="settings" element={<Settings />} />
44     </Route>
45     <Route path="*" element={<NotFound />} />
46   </Route>
47 </Routes>
48 </div>
49 );
50 }
51
52 export default App;
```

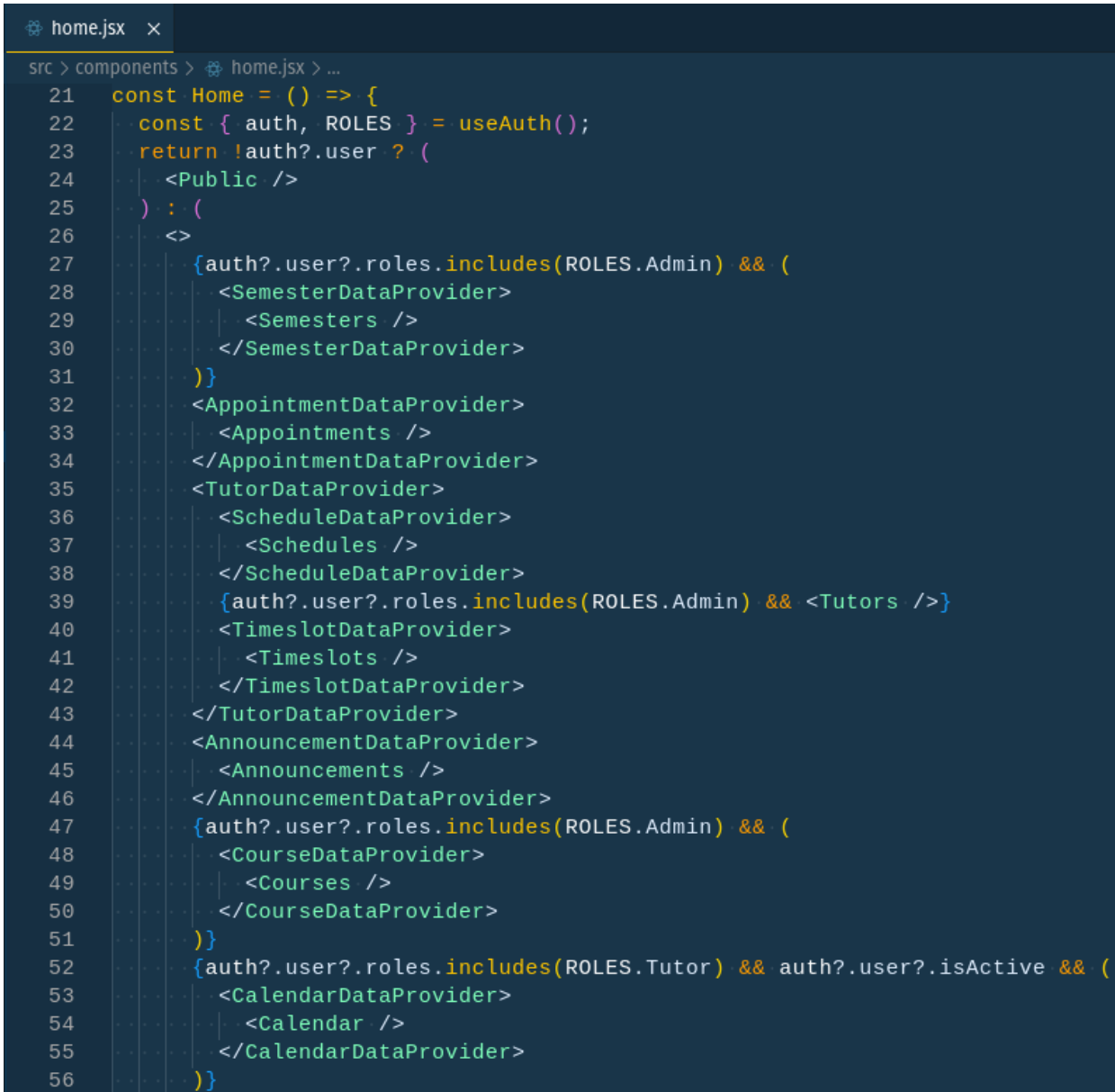
This is where the initial routings are defined depending on the user type and the url that is being visited. As you can see `<Profile />` component is only rendered when there is an authenticated user. By the same token, `<Settings />` component makes its way to the DOM if the user happens to have an *Admin* role.

Furthermore, if the user manually types in the url and if nothing matching is found then it will be caught by the last route where it says `path="*">``<NotFound />` component will be rendered to inform that there is no matching route.

- Home Component

Upon checking the contents of `<Home />` component you will find most of the high level components being rendered underneath it.

Figure 15: Home.jsx



```

src > components > home.jsx > ...
21  const Home = () => {
22    const { auth, ROLES } = useAuth();
23    return !auth?.user ? (
24      <Public />
25    ) : (
26      <>
27        {auth?.user?.roles.includes(ROLES.Admin) && (
28          <SemesterDataProvider>
29            <Semesters />
30          </SemesterDataProvider>
31        )}
32        <AppointmentDataProvider>
33          <Appointments />
34        </AppointmentDataProvider>
35        <TutorDataProvider>
36          <ScheduleDataProvider>
37            <Schedules />
38          </ScheduleDataProvider>
39          {auth?.user?.roles.includes(ROLES.Admin) && <Tutors />}
40          <TimeslotDataProvider>
41            <Timeslots />
42          </TimeslotDataProvider>
43        </TutorDataProvider>
44        <AnnouncementDataProvider>
45          <Announcements />
46        </AnnouncementDataProvider>
47        {auth?.user?.roles.includes(ROLES.Admin) && (
48          <CourseDataProvider>
49            <Courses />
50          </CourseDataProvider>
51        )}
52        {auth?.user?.roles.includes(ROLES.Tutor) && auth?.user?.isActive && (
53          <CalendarDataProvider>
54            <Calendar />
55          </CalendarDataProvider>
56        )}

```

Whenever you see a component with the name ending “...Provider” it means that it’s a Context providing data to child components. For instance, `<SemesterDataProvider>` would fetch the semesters from the server and pass it down the tree. In this case `<Semesters />` component and the components within it will have access to semesters data. So, instead of passing data as a prop we lift up the state to the context and access it.

The most important thing here to note is that `<TutorDataProvider>` provides tutors information to 3 different sub components: Schedules, Tutors and Timeslots. In the code you will see that actually DataProviders pass the tutors data down the tree but the idea is the same. So whenever something changes in TutorsDataProvider that affects all below components.

Another important note is that if there is no active user session only the `<Public />` component will be rendered. As a result, only 2 public tables will be shown. One is the tutoring schedule, no zoom links here. Second is the announcements table for checking the latest announcements. **Most importantly, no tutor ids or zoom links are sent by the server, only minimum information.** In the case when it is requested just to show the first name of the tutor but not the last name, this is the place you should be making changes. Also make the appropriate changes on the server side, so it only sends first names.

- **Calendar View**

There are 2 calendar views in the project. One for Timeslots which is visible by everyone and the other for tutors (My Calendar table) to create their calendars. In the future if something needs to be added or changed about the calendar itself, please refer to the docs ([Big Calendar](#)) where they provide lots of demos.

- **Profile**

Picture is fetched from Google with a link that we save when the user signs in. Not much here to talk about, the only thing to note is that the *Bio* field only becomes visible if the user is an active tutor. We will be talking about this later in the documentation but in order to sign in as a tutor and be able to create calendars etc. the person first **must be activated** by the admin as such in the given semester.

- **Semesters**

Important thing to remember here is that **only 1 active semester** is allowed at any given time. If an admin marks the semester as “Inactive” then he/she has to remember to activate another one when the time is right or it could be automated.

- Tutors

<Tutors /> component will be used as an example here to show how things are rendered. Everything explained in this section applies to other high level components in one way or another. Let's take a look at the image below:

Figure 16: Tutors.jsx



```

41  ...};
42
43  ...return (
44    ...<MainContainer>
45    ...{!loading && !error && (
46    ...  ...<TitleBar
47    ...  ...title="Tutors"
48    ...  ...icon={
49    ...  ...auth?.user?.roles.includes(ROLES.Admin) && (
50    ...  ...  <PlusIcon onClick={handleAddTutor} />
51    ...  ...  )
52    ...  ...}
53    ...  .../>
54    ...  ...)}
55    ...  ...{loading && <LoadingPlaceholder />}
56    ...  ...{!loading && error && <ErrorPlaceholder />}
57    ...  ...{!loading && !error && tutors && tutors.length === 0 && (
58    ...  ...  <NoDataPlaceholder />
59    ...  ...)}
60    ...  ...{!loading && !error && tutors && tutors.length !== 0 && (
61    ...  ...  <>
62    ...  ...    <Table hover responsive>
63    ...  ...    <TableHeader
64    ...  ...    ...headers={
65    ...  ...    ...auth?.user?.roles.includes(ROLES.Admin) ? adminHeader : header
66    ...  ...    ...}
67    ...  ...    ...darkTheme={darkTheme}
68    ...  ...    .../>
69    ...  ...    <tbody className={darkTheme ? "" : "text-muted"}>
70    ...  ...    ...<TutorRows {...{ setShow, setTitle, setModalBody }} />
71    ...  ...    ...</tbody>
72    ...  ...    </Table>
73    ...  ...    <CustomPagination />
74    ...  ...    </>
75    ...  ...  )}
76    ...  ...<TemplateModal {...{ show, title, ModalBody, reset }} />
77    ...</MainContainer>
78  ...);
79  ...};

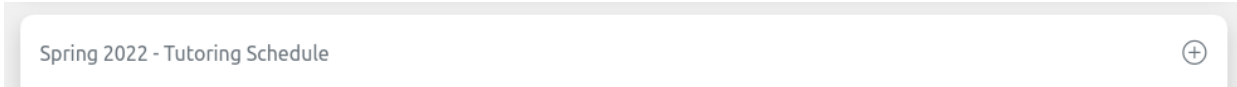
```

From top to bottom:

<MainContainer> - is the commonly used component in every high level component. It is the white or silver colored container depending on the theme that is set which has some depth.

`<TitleBar>` - is also another commonly used component which you can find under the `src/Components/common/` folder. It is used for displaying the name of the table, the semester that is loaded and at the far right corner an action icon.

Figure 17: Titlebar



`loading` - variable comes from the Context which in turn comes from `useAxios` hook. It indicates whether a data fetching is in progress or not. Depending on the state of loading `<LoadingPlaceholder />` is shown or not.

`error` - another variable that comes from the Context. Indicates if any error occurred during request to server.

`<Table>` - is a React-Bootstrap provided component. You can check the documentation if something is not clear. Links to all used tools and libraries can be found in this [section](#).

`<TableHeader>` - is one of those commonly used components. Also could be found under `common/` folder.

`<tbody>` - is just a plain html tag.

`<TutorRows />` - is another sub component which takes care of rendering rows and the functionality for toggle the active state of the tutor.

`<CustomPagination />` - as the name suggests it is for cycling between pages. Also under `common/` folder.

`<TemplateModal />` - is just a wrapper around React-Bootstrap's `<Modal />` component. It is all over the place. Wherever you see a pop-up screen it is this component that gets rendered (except Google's One Tap sign in pop-up).

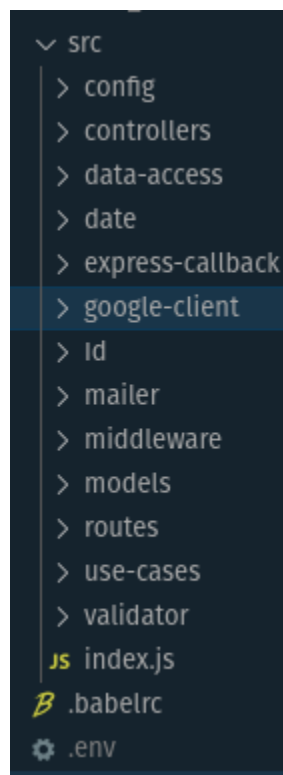
In conclusion, Schedule, Timeslots, My Calendar, Courses etc. all will have a similar layout. So there is no point going into each of those and explaining the same stuff.

2. Back-End

The server of the CS-Tutoring application was written in JS using NodeJS/Express web framework. It is quite simple and intuitive, especially if you are coming from Ruby on Rails, Flask etc. As in the Front-End section we will start off with a folder structure and move on to other parts that make the server.

- Folder Structure

The back-end also contains some configuration files which will not be covered here because it is purely for setting up the environment. The back-end and the



front-end both were built using functional programming only. So you will not see any declarations that start with “*class*” keyword as you might see in languages like Java, C# or JS’s syntactic sugar around the functions. At the end of the document there is a credits section where the links will be shared to the GitHub repo that inspired this whole project structure. Furthermore, the repo has a link to a youtube video where the author explains in much more detail about the pros of the “[Clean Architecture by Bob Martin](#)”, also known as “Uncle Bob” and also about functional programming. Although the folder names are self explanatory, I would like to give some brief descriptions:

- *config* - configuration files like Roles, allowedOrigins
- *controllers* - these are the functions that receive

httpRequest object from express-callback function

- *data-access* - only place where you define the database related configuration, and export an object to be used to write to the database. In our case, it happens to be a mongodb. But if you wanted to change it to some other alternatives all you have to do is to supply the exact same functions with the same signature and export it. You can think of it like *interfaces* like in OOP. You

implement a particular interface and no swapping needed in places where the old one is used. Given that the old one was provided as a *Dependency Injection*, meaning it was provided as a parameter instead of importing

- *date* - same principle applies here. We could swap the date manipulating library/module for another one. Here we are using *date-fns*
- *express-callback* - this folder contains only 1 function. Which acts as a middleware function. It receives (*req, res*) objects from the express and transforms needed information into a regular JS object, named httpRequest. Which in turn it passes the httpRequest down the circle to the Controller function
- *google-client* - exports OAuthClient that is used to verify the Google tokenId sent in the header from the front-end

- *Id* - exports `makeId()` function that is used to create unique ids. Could be easily swapped for an alternative module.
- *mailer* - exports `mailer()` for sending email notifications for canceled appointments or weekly reports. Swap for an alternative if you wish so. In this application `nodemailer` was the choice.
- *middleware* - all the middleware functions reside in this folder. `verifyUser`, `verifyRoles`, `errorHandler` and `credentials`.
- *models* - these are entities like appointment, schedule, user etc.
- *routes* - all route handlers are in here. `userRouter`, `appointmentRouter` etc.
- *use-cases* - this is very the logic of the application lives, also known as *Business Rules*. Each model will have its own corresponding use-case folder. For the user model there is a “user” folder in the use-cases folder. Which will contain super self explanatory files that export functions like *add-user* for adding new users, *edit-user* for editing existing ones, and so on and so forth. What’s more is that same idea for all models.
- *validator* - this is where we keep the validation schemas that are named after models. `user-schema`, `calendar-schema`, `event-schema` etc.
- *index.js* - is the entry point for our application where we have configurations like setting up the correct routers to specific routes. Setting up middlewares for CORS, parsing json from requests etc.
- *.env* - is the file where we keep our environment variables such as database url, password, username and password for an email account that is used to send emails. Moreover, we also keep the `baseUrl` to our API, `google client_id` and secret that is provided when you create a project in [Google Console](#). Secret that you get from google is needed when verifying the `tokenId`.

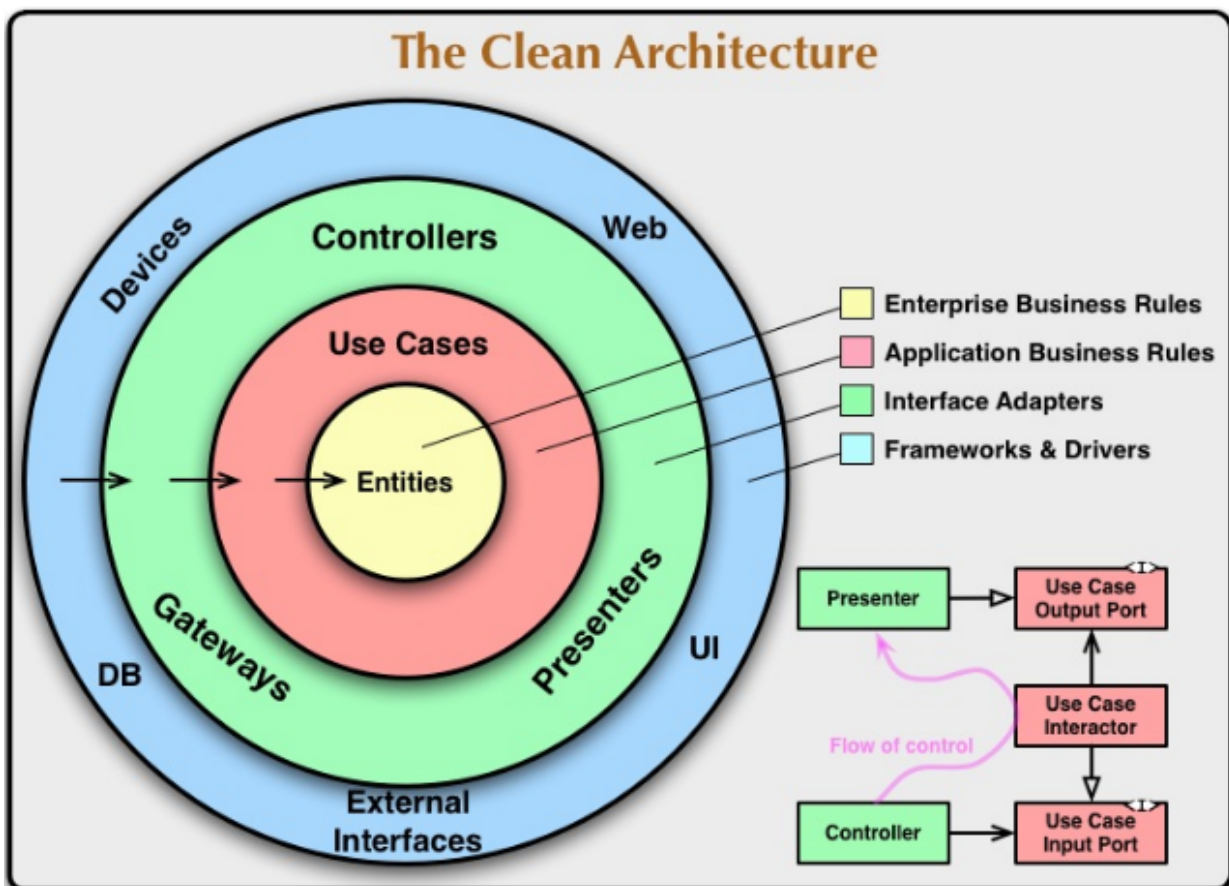
- Clean Code Architecture

Before we begin discussing the architecture there is one thing I need to say, I hope that I stayed within compliance of the “Clean Code Architecture” and I did not cross boundaries which would violate the principles. Now with that out of the way it is time to discuss the overall architecture and try to match the folders that we have to layers in the circle below.

- *Entities* models/
- *Use-Cases* use-case/
- *Controllers* controllers/, express-callback/
- *Outermost circle* mongoDb, express, nodemailer all other tools

The idea here is to decouple the inner circles from outer circle dependency. For example, Entities cannot have a dependency from Use-Case or any other layer. By the same token, Use-Cases cannot depend on the outer circle. In our application for instance, we use Joi as a data validation library which is an external library and it is used in models. But we are not importing the library in the model, instead we are passing it as a parameter, in other words injecting it.

Figure 18: Clean Code Architecture



source: blog.cleancoder.com

- Demo Scenario Steps

Next, we will go over the steps from the moment a request is received on the server and processed, to the point where the response is sent back to the requester.

1. Request comes in
2. Middlewares will be executed in order that is defined in index.js
3. verifyUser middleware will also run in step 2 to check the header for the cookie that was set when the user logged in. It will check who is the user and will set the req.user object. (***This part might change, instead of a cookie maybe it will only be Auth token which will expire in an hour***)
4. If the requested endpoint requires verified roles, verifyRole middleware will run
5. If there are no errors, express-callback middleware will run. Which will process the request and prepare the httpRequest object to pass down to the controller that was passed into it in route handler
6. Controller that received the httpRequest will get the body of the request if there should be one for the requested action, params like in query. Which will in turn pass them to the use-case.
7. Use-case processes the information, makes database calls if needed. Furthermore, it will either throw an error with appropriate code or return data to the controller.
8. Controller processes the response from the use-case and sends back an object containing {headers, statusCode, body: { e.message } }
9. Express-callback prepares res with the data it received, sets the statusCode that was passed by inner circles and sends response back to the requester

In conclusion, as you can see the process is pretty straightforward and it applies to all requests, no exceptions.

Next, we will cover the Installation process.

3. Installation

There are a couple things that we need to mention before we actually start installing the server. Make sure that following tools, libraries are installed on your machine:

1. NodeJS
2. NPM - which is usually installed alongside the NodeJS
3. MongoDB (or mongo in the cloud)
4. Browser - chrome recommended
5. Internet connection

First we need to download the source code for the front-end and back-end. At the time of writing this document I have it in my github account which you can find [here](#). From the link you need to download both [cs-tutoring-api](#) and [cs-tutoring-webapp](#).

I would encourage you to always contact the admin first for the most recent version of it as mine might not be the latest at the time of reading this doc.

Steps to install:

- Front-End (cs-tutoring-webapp)
 - a. Edit the name of the file `.env_sample` make sure it is just a `.env`
 - b. Provide the `client_id` in a `.env` file for Google Sign In
 - c. Open a terminal in the folder containing the source code
 - d. Run the command: `npm ci`
 - e. It should download the necessary files
 - f. **Run this line after the back-end is up and running:** `npm start`
 - g. Which will automatically start the development server of front-end
 - h. Uses port 3000 by default
 - i. Automatically launches the home page with default browser
- Back-End (cs-tutoring-api)
 - a. Repeat the step “a” from the Front-End Installation instructions.
 - b. Provide all the information needed in a `.env` file
 - c. Open a terminal in the folder containing the source code
 - d. Run the command: `npm ci`
 - e. It should download the necessary files
 - f. Run the command: `npm ci`
 - g. Which should start the development server of back-end
 - h. Uses port 4000 by default
 - i. You should see in the terminal which reads something like “Server is listening on port 4000”
 - j. Now is the time to run the step “f” of the front-end installation: `npm start`

Conclusion

From the time I began this project it has been quite a wild experience for me both personally and professionally. For one, I feel more comfortable again doing the projects from scratch on my own, as a result, it helped me to develop a more mature mind to defy the urge to quit when I face difficulties. Moreover, I feel like I'm back in shape in terms of learning something new. Although I did not know ReactJS when I started the project I was able to pull it off in 2-3 month and that feels rewarding. It may not be the most elegant React code on the planet but I will keep on improving.