

001196337

Maija.lehtosaari@student.lut.fi

AI STATEMENT

ChatGPT was used to:

1. Help with wording in some sentences in this document
2. Understand where to add bearer token in postman and how to get it
3. To assist in understanding and implementing complex state updates few times. This was done in tickets
4. To find typos in code when finishing the project

Table of contents

Project environment setup	2
Backend login & registration set up	2
Frontend login & registration set up	2
Database planning	3
Kanban routes	3
Basic frontend view for showcasing boards and columns	4
Styling boards and columns	5
Post & Get routes for tickets	6
Front-end fields for adding boards, columns & tickets	7
Dragging tickets	8
Styling	9
Security and redirection	9
Sources	14

Project environment setup

I used my week 7's homework as my backend base. Because of this I was able to install almost all my backend dependencies at once. Since my existing code had dependencies defined (in package.json), I had to just run "npm install" once and everything was installed (to my backend).

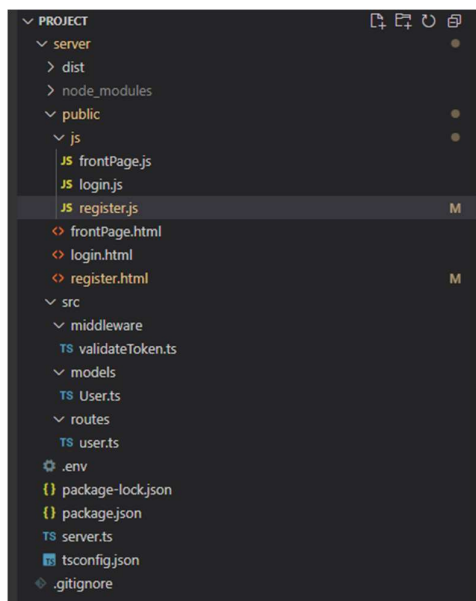
Front-end was initialized using Vite "npm vite@latest". I also used TypeScript as instructed and React for UI. MUI was used for styling & responsiveness.

I also initialized the project with Git and connected my local folder with remote GitHub repository. I tried to stay consistent with my commits and push my new files when I had a new feature added.

Lastly, I added the same "concurrently" script that was in week 12 to run front- and backend easily.

Backend login & registration set up

As stated previously I used my week 7's existing login & register routes and code. I re-read my code and modified a few things to make it better. At this point my folder structure was this:



Frontend login & registration set up

After setting up my backend, I started working on the frontend.

I wanted to create a similar structure as in week 11 homework, where I would use react-router and have app.tsx routing the pages. I used my week 11 header and throughout the project added necessary dependencies.

I now had to create the working login and register pages at client side. I referred to the course example code from week 12 and tried to do a similar structure but made modifications to better fit my project (e.g my project used email instead of username).

Once I had the basic login set up, I tested it but came across an issue with the connection. I couldn't send requests using my frontend or postman.

To troubleshoot, I reviewed the week 12 example code to understand how the client and server connection was set up there. I found that my issue was either that my front- and back- were using the same port or that I had CORS restriction. After changing my frontend port and configuring CORS options in the backend, I was able create a working connection and login from the front end.

I used my login page as a base for my registration but just modified the route address. I also modified my login page to redirect the user to the front page after login. I did this using `useNavigate` which is a function I found by searching [1] up how to redirect with react. I had to change the function structure a bit since I was getting a error that navigate can't be used inside the `fetchdata`, but it must be under the `login` function.

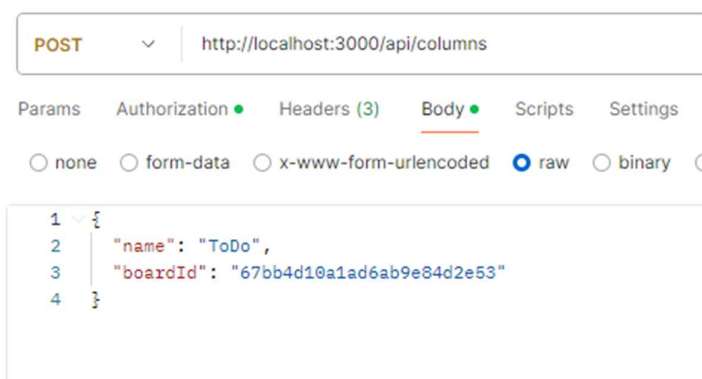
Database planning

After this I started thinking about how I want to do the kanban board itself. I've done a similar web application where I tied information to `userId`. I decided that I would tie board to `userId`, column to `boardId` and ticket to `columnId`. This way users can have multiple boards; columns belong to that board and tickets are tied to columns. I also thought this would make the code more expandable.

Kanban routes

I created the schemas for tickets, columns and boards as well as routes to create boards and columns in `kanban.ts`. The post routes for columns and boards were done by taking POST route from user and modifying this. I modified the URL, main code and return message. I created most of my new routes by copying an old one and using this. I did this to make less errors, typos and to have consistent error handling.

I searched up how to reference MongoDB's ID's and used this in my codes [2]. I would try my routes throughout the project with postman and in this case too, I got certain errors which made me modify my backend. I also had to figure out how to add my bearer token in postman authorization:



Params	Authorization	Headers (3)	Body	Scripts	Settings
Headers					
Key	Value	Description			
<input checked="" type="checkbox"/> Content-Type	application/json				
<input checked="" type="checkbox"/> x-version	1.0.0				
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFnbiI6InRlc3Qx...				
Key	Value	Description			

Data sent to MongoDB using Postman:

```

_id: ObjectId('67bb52acc1258c07fd474444')
name: "Test column 1"
boardId: ObjectId('67bb4d10a1ad6ab9e84d2e53')
__v: 0

```

```

_id: ObjectId('67bc9752fcd0737f92230a3a')
name: "Test Column"
boardId: ObjectId('67bb4d10a1ad6ab9e84d2e53')
__v: 0

```

```

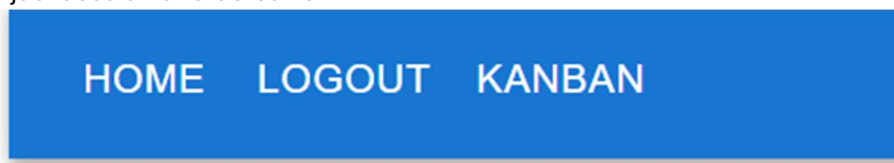
_id: ObjectId('67bc9758fcd0737f92230a3d')
name: "Test Column 2"
boardId: ObjectId('67bb4d10a1ad6ab9e84d2e53')
__v: 0

```

After this I created a GET route for both columns and boards. Boards would get the user._id from my validateToken which uses a customJwt to get the user_id. This makes it so that the user sees the correct boards. The columns route gets board._id and uses this to display right columns for the board.

Basic frontend view for showcasing boards and columns

After this, I created a simple way of displaying boards in home.tsx. I used map to display my boards and created a interface so that each boards _id would be identified and I could use it to redirect the user based on that. useEffect is used to fetch the boards from backend and to start I just used an unordered list:



Boards here:

- [test board3](#)

I then modified kanban.tsx to have a route that allows users to view a specific board by its ID. I had used ID's before in URLs, but I did look up some examples [3, 4]. I added useParams at the

start next to my useState. After this I was able to reference boardId and my front end looked like this:



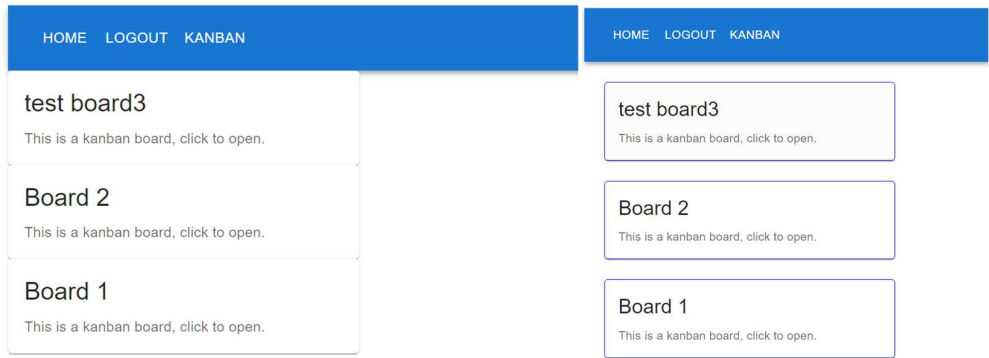
Kanban Board

Columns listed:

- test board3
- Test column 1
- Test Column
- Test Column 2
- ToDo

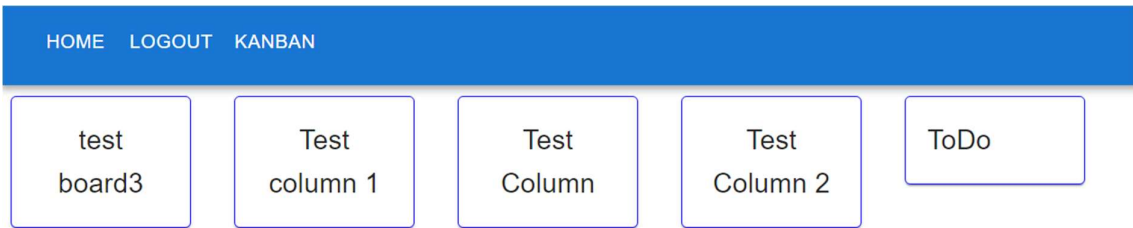
Styling boards and columns

After this I decided to style the boards and columns. I looked up example MUI cards [5] and used one of the examples as a base for my board cards:



I added extra margin and looked up how to create colored outlines [6]

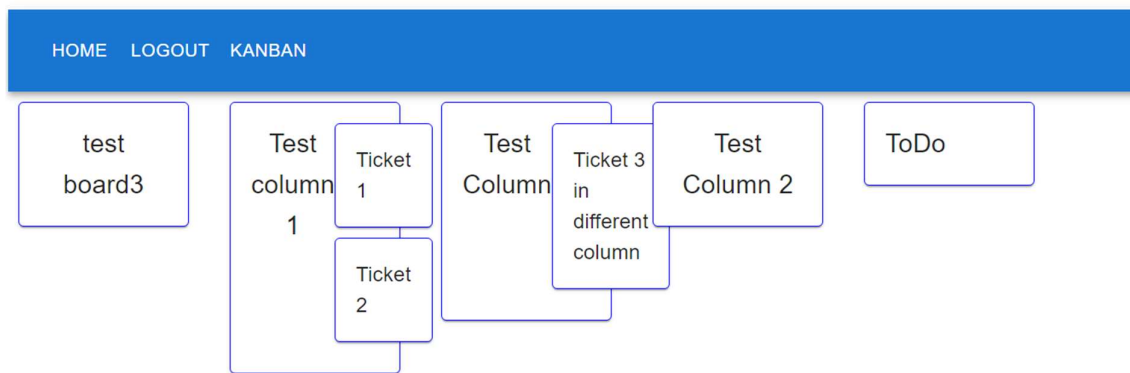
For the cards I used Grid [7]:



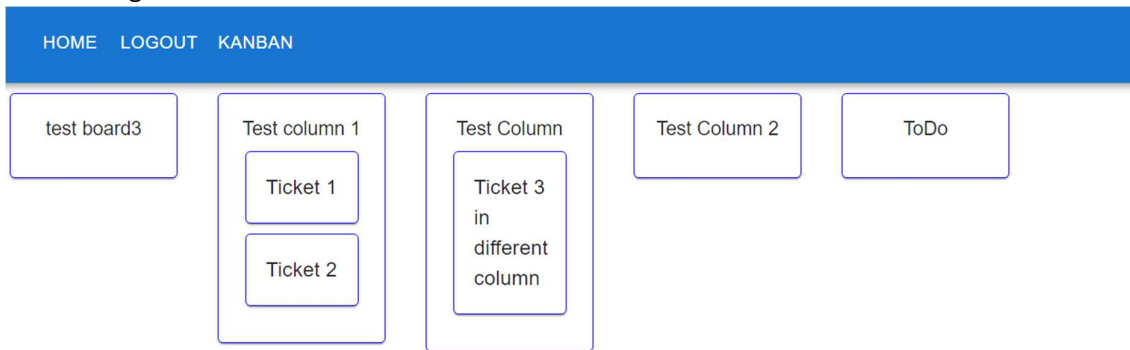
Post & Get routes for tickets

Next up I wanted to add my tickets. I created a POST and GET route for them. I created these by copying column routes and modifying them slightly. The code was very standard for these and like what I had done during the course.

I started modifying `kanban.tsx` which would get columns with `boardId` and map the tickets using `columnId`. The modified code first fetches the columns with previously created `fetchColumns` and iterates through the list to fetch tickets based on the `columnId`. I found good sources on how to do this when I googled “how to make multiple api calls react” [8-10]. I read some of these, tried my front end and changed the ticket styling to make the front end look good:
Initial look:



After changes:



Front-end fields for adding boards, columns & tickets

I then started creating fields to add boards, columns and tickets, since I had done this using Postman before this. I used source [11] to revise how to create react forms/textfields as well as course material. I added html buttons and front-end route logic to send POST message to my backend. This was easy for boards but when I tried to use the exact same code for columns I had some issues.

My column would be blank before I refreshed the page or the whole site would go blank when I clicked the add button. The column state wasn't updating after adding a new one. After trying different things, I decided to call `fetchColumns` immediately after adding a new one. I had to move `fetchColumns` to be outside of `useEffect` so my `createColumns` could use it. This fixed the state issue.

I also created HTML elements inside column that would allow the user to add ticket. When the field has text and button is clicked it calls `createTicket`. This function sends a request to the server to add that ticket within that column. All my routes retrieve the authentication taken from local storage and use it in the header. Then the ticket name and column ID are sent to the request body. This is what it looked like after:

The image displays a user interface for managing columns and tickets. At the top, there is a form to add a new column, consisting of a text input labeled "New column name" and a blue button labeled "ADD A COLUMN". Below this, there are two column cards. Each card has a title ("test" and "test2"), a text input labeled "new ticket" with a "text" placeholder, and a blue button labeled "ADD A TICKET".

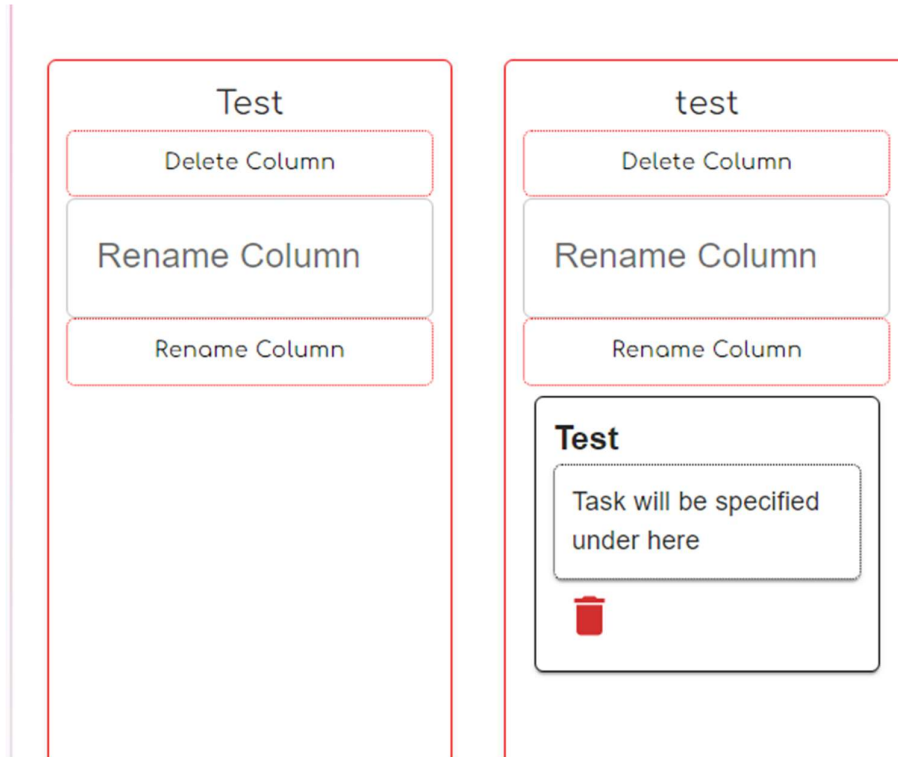
I also had an issue with all my ticket text boxes updating at the same time. The issue was fixed by managing ticket names individually and having a `newTicketName` -state object that used column ID as the key.

Delete columns and tickets

In the front-end, Columns and tickets are deleted by calling their API endpoints and updating the state after deletion. Delete columns sends a DELETE request to the backend with the column ID. Delete tickets sends a DELETE request to the backend using ticket ID. After successful deletion, the frontend state (view) is updated to reflect the change.

Rename column

I noticed I didn't have a rename column function at the very end of my project and didn't have a lot of time left. I added the rename column functionality by first creating a PUT route to update the column name. This was based on the POST column route, with a few variable names and functions adjusted. Then, I set up a front-end call to this PUT route and a setState function to update the column name in the frontend. Finally, I made it so that users can trigger the rename by typing in a text field and clicking a button.:



I would have liked to make it look better but didn't have much time.

Dragging tickets

I now had most of the basic functionality down. I decided to create the drag and drop functionality for tickets. I looked up different React drag and drop guides which I didn't add to sources. The one I ended up using is [12] a YouTube video and GitHub source code that shows how to create a drag and drop using React-beautiful-dnd.

I watched the video and tried to write my code alongside it. I first installed the react-beautiful-dnd and added the imports. I then wrote the HTML and handleDragAndDrop-function.

The handleDragAndDrop includes error handling for wrong use cases (e.g. ticket dropped outside drop zone). After this it defines the basic function where a ticket is dragged into another column.

The HTML defines what can be dragged (draggable) and where it can be dropped (droppable). It also includes the index of the draggable item within its container which defines the position

during the drag and drop. After the drag and drop is done, the source and destination data are passed to the function to use and update the state.

Styling & Responsiveness

I styled my application by adding button icons, changing fonts, colors and layouts of things. Most of these were done by using MUI and its own official tutorial page. For example, I would look for fitting icon from MUI's page [13], copy the import and paste it in my code. Then I would just modify my old button to use the icon.

I tested the responsiveness of my app by stretching the screen to different sizes. I read through my code to ensure that no hard values (e.g. 15px) were used. This guarantees that the app will stretch and work well across different devices like PC, mobile and tablet.

Additionally, by setting some of my boxes to have “flex” display instead of static position, I was

Code was formatted using prettier VsCode extension.

Security and redirection

I had been securing my routes throughout the project with `validateToken` but I wanted to check that everything aligns with the instructions.

First, I changed the app to launch at login page where the user can also go to register. I did this by changing the default route in `app.tsx` to `/login`. After this I tested my home page. Non-authenticated users can't add boards, but they can see the empty page. I was okay with this. I then checked that all my routes were secured with `validateToken` and tried different breaches and workarounds. I made sure that users can't POST, GET or DELETE without a token.

User manual + installation:

To use the application, follow these steps:

Prerequisites:

- Have node.js and npm installed
 - Have MongoDB installed locally and open
-
1. Clone the repository
 2. Run “npm install” to install the dependencies (in server and client)
 3. Create a db in your MongoDB that the project can use

4. Add your MongoDB URI here (project/server/server.ts:

```
9
10  dotenv.config()
11
12  const port = 3000
13  const app: Express = express()
14  const mongoDB: string = "mongodb://localhost:27017/testdb"
15
```

5. Run “npm run dev” in root folder
6. Front end runs on <http://localhost:3001>

User Guide:

1. App starts in login, click register to redirect there.
2. Register with email and password that you remember:

Register

Email *

test1@gmail.com|


Password *

.....

REGISTER

LOGIN

3. When at home page, type a board name in the text box and click “add board”:

Web task manager application  LOGOUT


Board name

Add a board

test

This is a kanban board, click to open.

4. Click on board, type a column name in text box and click “add column”:

Web task manager application  LOGOUT

New column name

Add a column

To Do

Delete Column

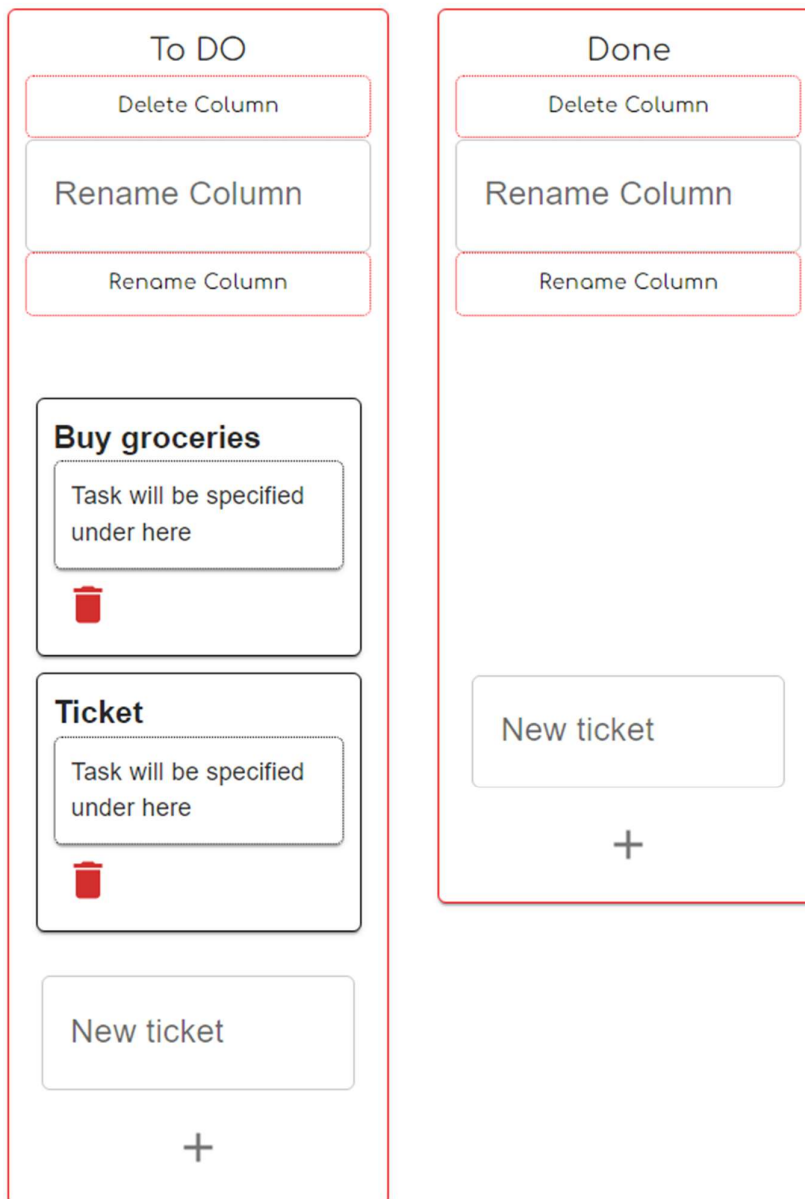
Rename Column

Rename Column

New ticket

+

5. New ticket field and “+” Icon creates new tickets, “trash”-icon deletes a ticket. Rename column renames the column after user types in the “rename column” field. Tickets can be dragged up/down or to a new column.



6. Home icon in header can be clicked to see user's boards, "Logout" redirects user to login page.

Implemented features + points:

Features:

- Mandatory requirements such as use of db, node.js
- Add/remove/rename columns
- Authentication through validateToken
- Cards can be moved between columns and in them
- Cards can be deleted
- Non-authenticated users can register and login

Feature	Points
Basic features (as stated in the previous chapter) with well written documentation	25
Uses React	3
Cards can be dragged and dropped	2
Extra styling (MUI icons, colors, google fonts)	1-2
Total	31-32

Sources:

- [1] useNavigate; <https://www.dhiwise.com/post/enhancing-navigation-with-redirects-in-react-route>
- [2] <https://stackoverflow.com/questions/63673652/is-there-a-way-to-pass-user-id-to-schema>
- [3] <https://stackoverflow.com/questions/75522048/react-how-to-access-urls-parameters>
- [4] <https://refine.dev/blog/react-router-useparams/#conclusion>
- [5] Card style (Lizard clickable) <https://mui.com/material-ui/react-card/?srsltid=AfmBOopSJYUSQLPa3rWf7HwPTpXdBez8XxKltJq1GZLbG6hAukK52onC>
- [6] Card outline / color <https://stackoverflow.com/questions/68830099/how-to-set-the-border-color-to-card-component-in-react-material-ui>
- [7] Grid and other MUI used in columns: https://mui.com/material-ui/react-grid/?srsltid=AfmBOorxuGvv--Kj7ZnInLwHKBkWoqyBa6oERo2ehVPmtqFTn_qnAL9v
- [8] Multiple api calls for tickets: <https://stackoverflow.com/questions/73092249/multiple-api-calls-with-promise-all>
- [9] Promise.all use: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/all
- [10] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function
- [11] React forms quick reading: https://www.w3schools.com/react/react_forms.asp
- [12] Drag and drop video + source code: <https://www.youtube.com/watch?v=YJ5EMzyimfc>
<https://github.com/harblait7/React-Beautiful-DND/blob/main/src/App.js>
- [13] MUI source: https://mui.com/material-ui/material-icons/?srsltid=AfmBOopLca1GG1INQZ1-3nI-bS2geHg5QMgsZW_-J5q5nTgVtZNmmEGt&query=home
- [14] NOT USED: <https://www.geeksforgeeks.org/how-to-use-google-fonts-in-reactjs/>