Sprint # 02 - Team 33

23 March 2021

Ishwor Giri, Hamza Bouhelal

FRONTEND INSTALLATION

#Install Dependencies
npm install
or
yarn install

Further steps are only shown using yarn but you can use npm
# Starting Frontend

yarn start
Test cases
Only checks for .tests.js files
yarn test

#Most of the frontend was not implemented by the members from sprint 1.
#Documentation was not present. We had to understand the code ourselves and create this documentation.

FRONTEND INTRODUCTION

All the source code related to frontend can be found inside se-02-team-33 folder

Entry point is **index.js** and all components are put together using imports from various js files or modules.

All web pages feature a navigation bar with items “Login,” “Signup,” “About,” and “logout“ It also has a placeholder for a logo, which is temporarily occupied by an alternate text reading “Beer Game.”

AXIOS:
We have customized axios to add features of including login tokens/ refreshing tokens to the request header for authentication and saving JWtokens to the localstorage.
https://jwt.io/

Before working with frontend please make sure to edit axios.js file if you have a backend server that doesn't run on the already defined url and port localhost:8000
axios.js

```
const baseURL = 'http://localhost:8000/';
```

https://github.com/axios/axios Please read the documentation before working with axios.

Using Customized Axios

```
import axiosInstance from './axios';

axiosInstance.get("game/").then( (response) => {
// do whatever with response}).catch((error)=>{
//catch error and update your state to display error
}
)
```

Above function will request using get method from baseurl + game/ . for this case http://localhost:8000/game/

Also, make sure to whitelist your frontend website url in the backend for removing CORS

https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

Index

The *Index.js* imports *App.js* from "./App" component.

App.js centralises activity.

It imports all other components through their respective directories as well as BrowserRouter as Router, Switch, Route, and Link from React.

It makes use of <Switch> protocol, which looks through its children <Route> and renders the first result that matches the current URL it holds. All the routing to the app is handled using this and other components can refer to these endpoints using <Link to=""> for quick render without reloading pages instead of using anchor tags from html.

Login Sign Up and Logout

Login.js , Signup.js, logout.js

Login and Sign Up pages make use of simple Material CSS forms and labels/names to extract user information: email and password.
It uses states to manage state and information related to forms like email, password, errors.

Login:
requests : backendurl/api/token
User login is facilitated by use of jwt token which is saved on localstorage and sent on every request using a custom axiosInstance function found on axios.js .

Signup:
requests: backendurl/user/create/
The user can input their information and the information will be validated and stored in a database.

Both login, signup use state management like setstate and usestate for making changes to states and function handlechange handles changes to form and updating the object containing all details like email, password. Once the user clicks on the button another function gets triggered (handleformsubmit) which sends requests to the backend to their respective urls and gets the response and is handled accordingly (to update state for error or redirect if successful).

Logout:
requests: backendurl/user/logout/blacklist/
Logout sends an axios request to this url to the backend for blacklisting this current token so that no one can use this token and all tokens are also removed from localstorage.

Homepage
landingpage.js

The Homepage simply gives the user an option to choose between acting as a **Player** or an **Instructor**. uses Link to route to respective pages /player/ and /instructor/.

INSTRUCTOR AND PLAYER VIEWS
player.js, Instructor.js

Join as: Instructor

Once the user has chosen to join as an instructor, the user will be redirected to *Instructor.js* directory (./instructor/).

Instructor.js gives the user the following options:

1. **Create Game**
   requests:backendurl /game/ POST
   CreateGame.js

In order to create a new game, the instructor has to fill in the following information: Enable Wholesaler (type: checkbox,) enable Retailer (type: checkbox,) whether to Share Activity with Other Players (type: checkbox,) Week Length, Backorder Cost, Password for the game, and Demand Pattern through a dropdown with hardcoded values (can be changed later.)
NOTE: NOT THE FINAL FORM STRUCTURE, needs more properties to be added such as shareactivity, password for game, demand pattern.

2. **View Created Games List**
   requests: backend /game/ GET

Once the instructor logs in to the system. he/she can see the games created by himself/herself not others . The UI has not been implemented yet.
Once the successful response is received from the backend in json array format, state will be updated to save the gamesdata. The function loadgame will then go through each game and create a new component ViewGame and will be attached to an array component which will be then returned by the loadgame function .

If response is 401 (unauthorized/ missing credentials will be displayed on the screen)

```
<ViewGame key ='{gamedata.id}' gamedata={element} />
```
is one functional component corresponding to each game .

There are still missing attributes like demand patterns , to be handled and displayed. But the main thing to get data from the backend has been handled and one can simply add things on that.

Join as: Player

Once the user has chosen to play as a player, the user will be redirected to *Player.js* file (./player/).

The Component will request /backend/user/info to get email details and display the currently logged in user email if logged in otherwise it will display None and error.

There is an option to retrieve game details .
Users can enter GAME ID and submit . After submission if the game id is valid the game details will be displayed in json format (view not implemented yet) . As the backend does not have the functionality to relate players with games , it will only display the game id for now.

users can also see the ongoing game(only one) and can continue the game without entering any game id, password , role. (not implemented yet . but one can create a backend api to return the current game and display it here) .

THE GAME SCREEN

Game.js

The game screen makes use of `Tab` and `Tabs` components from the 'react-bootstrap' library to separate information into four views: Orders, History, Plot, and More Info.

Orders: view1()

Players can see all the information for the current week such as demand from DOWNSTREAM, back Order, Total Requirements, Beginning Inventory, Incoming Shipment and Total Availability, Units Shipped to Downstream, Ending inventor. There is also a form and button to submit the request to purchase from upstream. Right now, the button only calls a pop alert window and prints the number. But, one should implement the function to send the request to backend to handle the further process

downstream,upstream players must be replaced with respective names once the backend works.

History: View2()

This view shows all the information for the last 10 week data. It uses another functional component DisplayTable taking data as week

<DisplayTable week="1" /> should render week 1 details

Plot: View3()

Not implemented yet but it should contain all the graphing related options.

More Info: View4()
not implemented yet but it should contain the status of the game if the week is over and the status of other players if enabled by the instructor .

That is all we have implemented for now. Please add more as you take on this project and make this the best one of all

Testing
----
all the testcases are inside the test folder. We have used a react testing library and jest for testing. Also, msw https://mswjs.io/ is being used to fake a server and send responses. Please read the documentation from the website to understand how it works before modifying test cases.

one can run all the tests by running this command **yarn test**
or
npm test