Ministry of Education of Republic of Moldova

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Software Engineering Department

Web Programming

Laboratory work #4

CONSUMING AN API

Author:
Anastasia Gavrilita
std. gr. FAF-191

Supervisor:
Alexei Sersun

Chisinău 2022

1 Task

1. Pick a frontend framework;
2. Create a web app that has the following functions:
  - it shows a landing page with different quizzes;
  - the user can pick a quiz and play it;
  - after the game has ended, the user can see their score.

3. The app should have attractive UI;
4. Consume [Quiz API](https://pure-caverns-82881.herokuapp.com) to fetch data from backend server.

2 Results

First, as a non-Coursera student, I tried to use Vue.js. However, the work environment set up has been failing for 4 days, so I switched to React, even though I took a mini-course on Vue on LinkedIn Learning, because the readable syntax attracted me. Starting with React, the very first thing I did after setting up a template project was to play with colors and dimmensions, using mui components, as well:

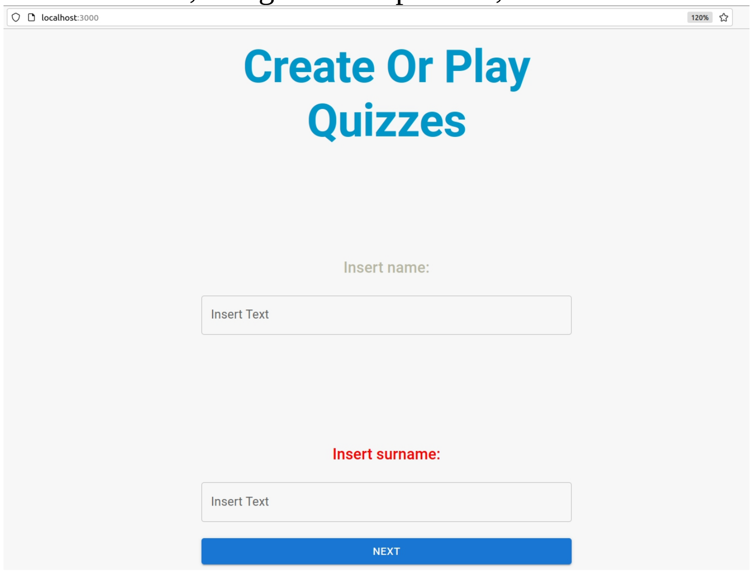


Figure 2.0. Create user page - experimenting with colors

After this, I started researching about best practices in React. Partly. The other part was pure React beginner-level explanations researching. I added the links I went through in the last section of this report.

After setting a new user, the quiz configurations page appears.

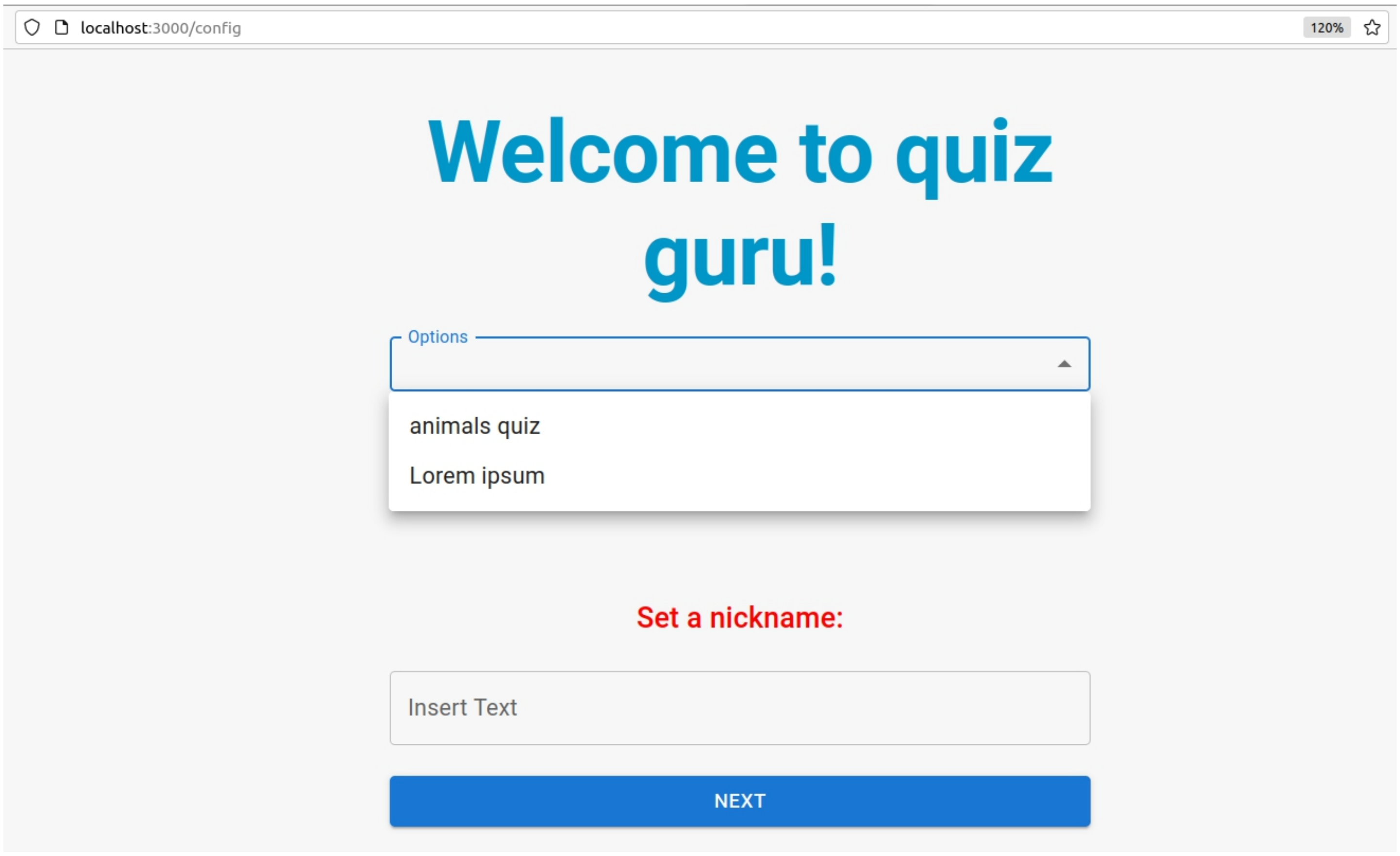


Figure 2.1.Choose quiz and set a nickname for the game

The source code of the project is available on Github.

Anyone would notice the red text prompting the player to set a nickname. In code, the red color is set like this:

```
color='red'
```

Listing 1: Setting color using actual name for it

I decided to get more flexible about colors, so I used hex codes, to get more accurate:

```
color='#0096c7'
```

Listing 2: Hex color code for blue

And this is the result:

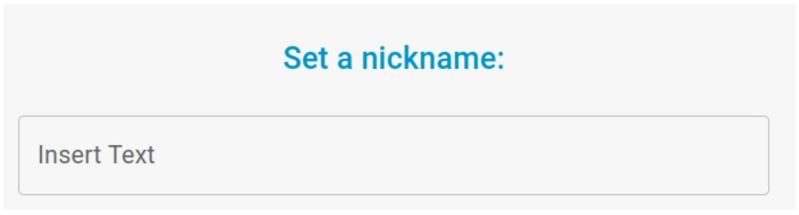
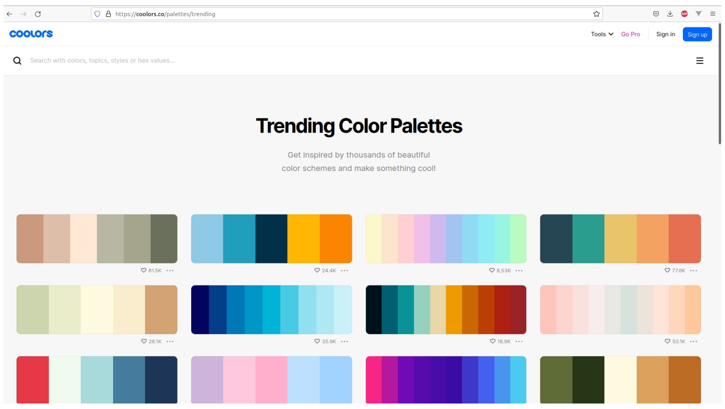


Figure 2.2. Nice blue obtained from hex code

I used a website that holds color palettes that can help any beginner combine and choose nice colors.

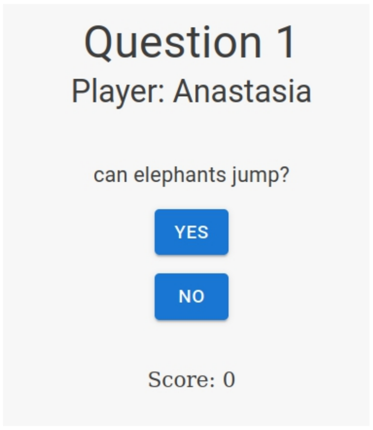


Trending Color Palettes

Get inspired by thousands of beautiful color schemes and make something cool!

Figure 2.3. Online color palette

This is what a question looks like:



Question 1

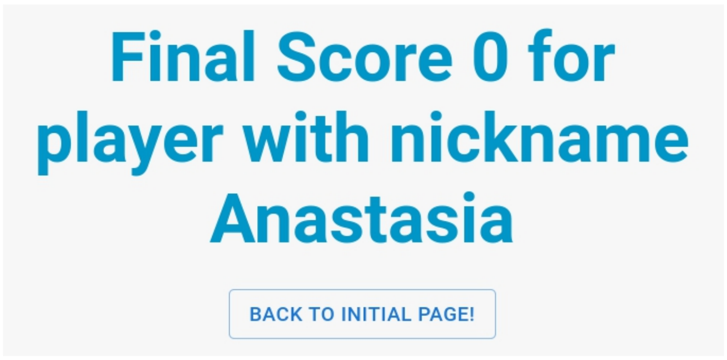
Player: Anastasia

can elephants jump?

Score: 0

Figure 2.4. Quiz question

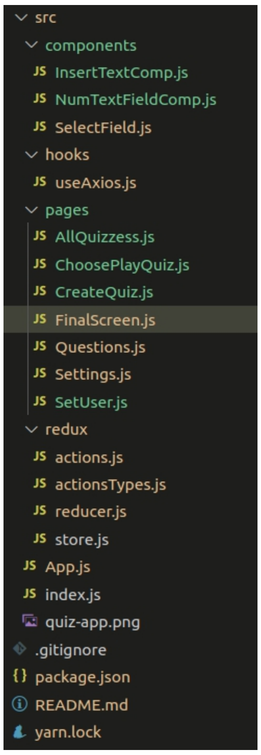
And the results page:



Final Score 0 for player with nickname Anastasia

Figure 2.5. Quiz final score

I created separate pages for each type of activity, and added them to the *pages* directory in the project. The project may seem to have a lot of files. This is how most of the tutorial recommended, and I felt like it was easier to sort out and isolate matters.



```
src
  components
    InsertTextComp.js
    NumTextFieldComp.js
    SelectField.js
  hooks
    useAxios.js
  pages
    AllQuizzess.js
    ChoosePlayQuiz.js
    CreateQuiz.js
    FinalScreen.js
    Questions.js
    Settings.js
    SetUser.js
  redux
    actions.js
    actionsTypes.js
    reducer.js
    store.js
  App.js
  index.js
  quiz-app.png
.gitignore
package.json
README.md
yarn.lock
```

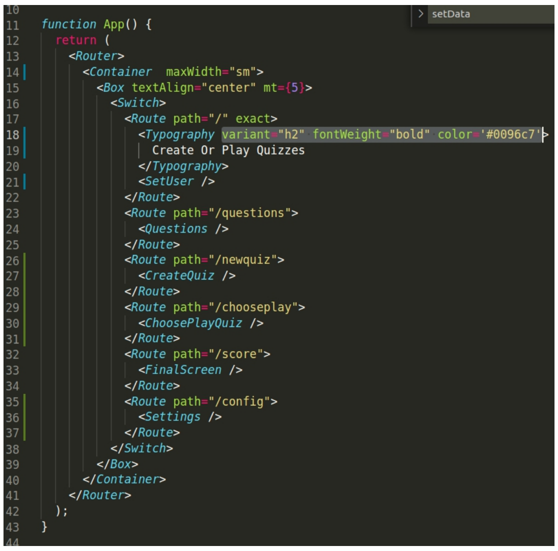
Figure 2.6. Project structure

The components directory contains actual parts of components in the screen - anything from a text box to a button we interact with. This has enabled me to reuse these components in several pages. The *pages* directory contains endpoint pages that hold some logic behind the user's interaction with the components and returns the actual webpage that should be displayed by the application. I used mui elements instead of pure html/css/js tags, for quicker development.

The *redux* directory contains actions (I see them as function definition) for events during the user's interaction with the application. An example of such would be the SET_NICKNAME action triggered by the user when they insert a nickname in the dedicated box, rendered in a page, using a component called *InsertTextComp* from the *components* directory.

The *hooks* directory is for sorting React hooks in it. React hooks do some magic configurations for us, web developers. An example is the actual GET/POST methods combination that use an url for this purpose. I let the "useEffect" -> axios do the trick for me and send the requests, then return the response.

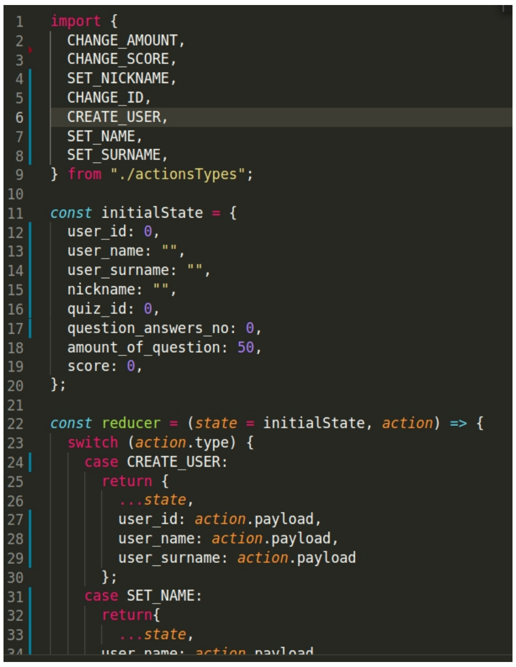
The rest of the files have been automatically created during the loading of the project and its build. I used the index page to let the webpage know I'm using React app, the the App.js file to enable a switch for my own customized routes:



```
function App() {
  return (
    <Router>
      <Container  maxWidth="sm">
        <Box textAlign="center" mt={5}>
          <Switch>
            <Route path="/" exact>
              <Typography variant="h2" fontWeight="bold" color='#0096c7'>
                Create Or Play Quizzes
              </Typography>
              <SetUser />
            </Route>
            <Route path="/questions">
              <Questions />
            </Route>
            <Route path="/newquiz">
              <CreateQuiz />
            </Route>
            <Route path="/chooseplay">
              <ChoosePlayQuiz />
            </Route>
            <Route path="/score">
              <FinalScreen />
            </Route>
            <Route path="/config">
              <Settings />
            </Route>
          </Switch>
        </Box>
      </Container>
    </Router>
  );
}
```

Figure 2.7. Switch that uses a router for custom endpoints that map to pages

In order to communicate information between pages and components, I used global states, initiated in the *reducer.js* file:



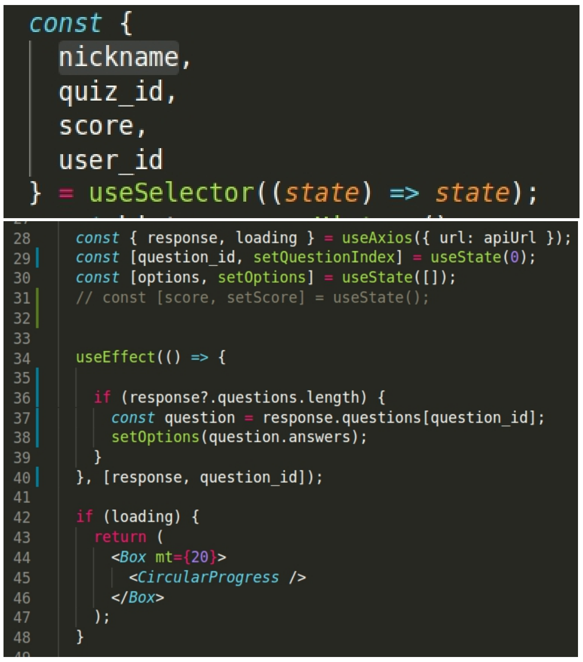
```
import {
  CHANGE_AMOUNT,
  CHANGE_SCORE,
  SET_NICKNAME,
  CHANGE_ID,
  CREATE_USER,
  SET_NAME,
  SET_SURNAME,
} from "./actionsTypes";

const initialState = {
  user_id: 0,
  user_name: "",
  user_surname: "",
  nickname: "",
  quiz_id: 0,
  question_answers_no: 0,
  amount_of_question: 50,
  score: 0,
};

const reducer = (state = initialState, action) => {
  switch (action.type) {
    case CREATE_USER:
      return {
        ...state,
        user_id: action.payload,
        user_name: action.payload,
        user_surname: action.payload
      };
    case SET_NAME:
      return{
        ...state,
        user_name: action.payload
```

Figure 2.8. Initiating states, to pass information

This is an example of how a state is used:



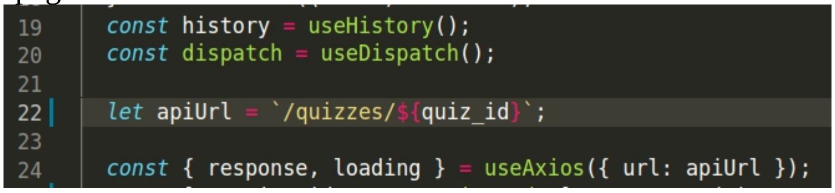
```
const {
  nickname,
  quiz_id,
  score,
  user_id
} = useSelector((state) => state);
```

```
28    const { response, loading } = useAxios({ url: apiUrl });
29    const [question_id, setQuestionIndex] = useState(0);
30    const [options, setOptions] = useState([]);
31    // const [score, setScore] = useState();
32
33
34    useEffect(() => {
35
36      if (response?.questions.length) {
37        const question = response.questions[question_id];
38        setOptions(question.answers);
39      }
40    }, [response, question_id]);
41
42    if (loading) {
43      return (
44        <Box mt={20}>
45          <CircularProgress />
46        </Box>
47      );
48    }
```

Figure 2.9. States

For example, line 29 sets the current state of the question to *question_id* variable, and sets it to 0 initially. The next state would be set returned by the setQuestionIndex function. In the same key, options are currently equal to the *options* variable, whilst the next state will be returned by the *setOptions* function. For this entire purpose, the useState() hook comes in handy.

Last, but not least, it is important to mention that the navigation from one page to another is possible with the help of the useHistory() hook. First, I created a variable *history* that uses this hook. Then, I called *useAxios()* to get access to the current page's URL.



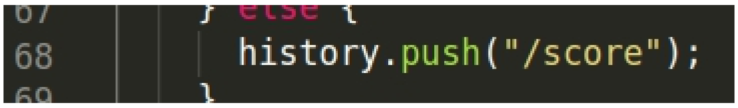
```
const history = useHistory();
const dispatch = useDispatch();

let apiUrl = `/quizzes/${quiz_id}`;

const { response, loading } = useAxios({ url: apiUrl });
```

Figure 2.10. useHistory() hook

After all the logic has taken place, I call history.push("/{next endpoint}) in order to tell this page where to direct the user:



```
    } else {
      history.push("/score");
    }
```

Figure 2.11. The score page will appear at the end of the quiz

3 Conclusion:

This laboratory work was, by far, the most challenging one for me. Not only did the environment set up take a lot, but also - it allowed me to learn a lot about the lock files, building process of a project and running an application in a web container. In the same context, I discovered the world of React and how friendly its logic can become when used with the best practices (like structuring separate components in separate files and directories). Also, the more magic functions, like hooks, I learned, the fastest I went through the process of creating and integrating new components. I still don't understand everything that happens under the hood and I'll look more into it, in order to solve some bugs in this laboratory work.

Working with an external API, bringing its information into my own custom endpoints was another challenge. Also, setting up a token made me reach out for help. In the end, I found out React has it all covered. This is why I rate this laboratory work to be one of the most fun one. At the same time, for now, I'll just submit this version, for some features and bugs will need more than a few minutes of attention in the future (parts of code are commented out).

Some of the websites I used, in order to research topics and get examples will be stated in the below section [1][2][3].

4 References

[1] React in 30 Minutes Tutorial, https://youtu.be/hQAHSlTtcmY, accessed on April 15, 2022
[2] UseState React Hook Tutorial, https://youtu.be/O6P86uwfdR0, accessed on April 16, 2022
[3] Custom Hooks Tutorial, https://youtu.be/6ThXsUwLWvc?list=PLZlA0Gpn_vH8EtggFGERCwMY5u5hOjf-h, accessed on April 20, 2022
[4] Use Axios Example for Post Method, https://dev.to/ms_yogii/useaxios-a-simple-custom-hook-for-calling-apis-using-axios-2dkj, accessed on April 21, 2022
[5] Use Axios Example For Post Method (with future and with callback), https://jasonwatmore.com/post/2020/07/17/react-axios-http-post-request-examples, accessed on April 21, 2022
[6] Source code: https://github.com/Anniegavr/Lab4React