Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Department of Software Engineering and Automation

Web Programming

Laboratory work #4

Quiz app

Author:
Liviu Mocanu
std. gr. FAF-203

Supervisor:
Alexei Șerșun

Chișinău 2023

1 Task

Our task for this laboratory work was to:

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 to fetch data from backend server.

We had no restrictions on using third-party packages.

2 Results

I chose `React` as the frontend framework for this laboratory work and `Chakra UI` for the desing management. I also used `Formik` and `Yup` for the forms present in the app as well as `Fetch API` for the requests.

Besides the main Authentication/Quiz functionality I added some features I liked to see in my Quiz App and that is:

1. A **song**, namely, Debussy's **Claire de Lune**, meets the user when loading the page for the first time. The user has a set of buttons in the header to help them manage it. Since I am using React's Link and Route elements, the music doesn't stop on URL change:
   - Play/pause depending on the state of the song
   - Volume down
   - Volume up
2. When scrolling down, the Header has a smooth animation of going up and disappearing, leaving more space to the user. Contrary, if the user scrolls up, the Header comes back down smoothly.
3. The User's avatar is a picture with their credentials. This is achieved with the help of the **https://ui-avatars.com/api** API.

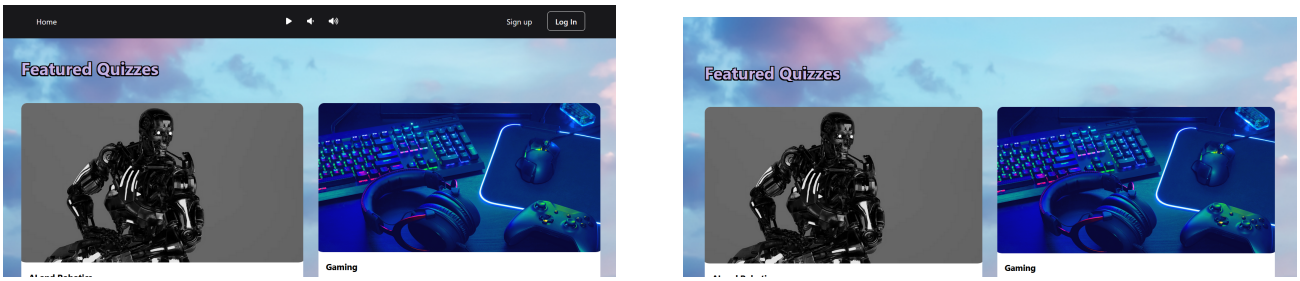
(a) Song is playing (b) Song is paused

Figure 1: Song control functionality in the header

```
function App() {
    ...

    const audio = useMemo(() => new Audio(backgroundMusic), []);
    audio.loop = true;

```

```
    useEffect(() => {
        audio.play();
    }, [audio]);

    ...
}

const Header = () => {
    ...

    const togglePlay = () => {
        if (isPlaying) {
            audio.pause();
        } else {
            audio.play();
        }
        setIsPlaying(!isPlaying);
    };

    ...
}
```

Listing 1: Mounting and allowing control for the song



(a) Header is present

(b) User scrolled down

Figure 2: Header disappearance on scroll down

```

const Header = () => {
    ...

    useEffect(() => {
        const handleScroll = () => {
            prevScroll.current > window.scrollY ? boxAppear.current = "0" : boxAppear
    .current = "-200px"
            prevScroll.current = window.scrollY;
            boxRef.current.style.transform = `translateY(${boxAppear.current})`;
        }

        window.addEventListener('scroll', handleScroll);

        return () => {
            window.removeEventListener('scroll', handleScroll);
        };
    });

    return (
        <Box
            position="fixed"
            top={0}
```

```
            left={0}
            right={0}
            translateY={0}
            transitionProperty="transform"
            transitionDuration=".3s"
            transitionTimingFunction="ease-in-out"
            backgroundColor="#18181b"
            ref={boxRef}
            zIndex={1000}
        >

        ...
    );
}
```

Listing 2: Making the Header disappear after scrolling down and reappear on scroll up

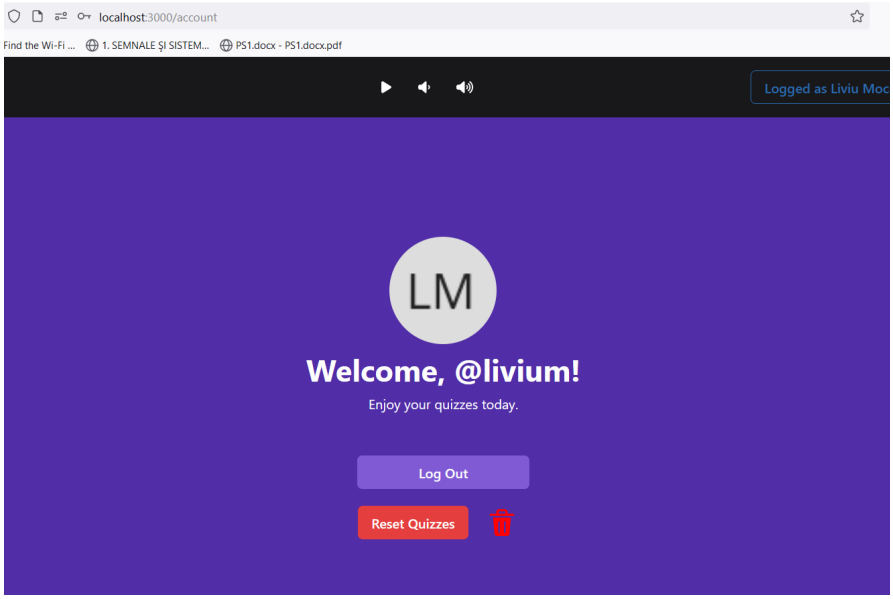


Figure 3: Generated User Credentials Avatar.

```

const imageUrl = "https://ui-avatars.com/api/?name=";

const LoggedIn = () => {
    ...

    const fetchImage = async () => {
        const userAvatarUrl = `${imageUrl}${name.replace(/ /g, "+")}`
        const res = await fetch(userAvatarUrl);
        const imageBlob = await res.blob();
        const imageObjectURL = URL.createObjectURL(imageBlob);
        setImg(imageObjectURL);
    };

    ...
}
```

Listing 3: Fetching the avatar as the user's name credentials

2.1 Authentication

The main page is the QuizzesSection page which displays all the featured quizzes.

A user is required to be logged in their account in order to attempt a quiz, otherwise, if they still proceed with trying out a quiz, they will be redirected to the signUp page with the prompt that they require an account first. Right after, they will be redirected to the previously chosen quiz.

All users are stored in the `localStorage` as well as the current user which is `loggedInUser` in order to avoid re-logging on refresh. A logged in User is allowed to review their cabinet by clicking the `Logged In as name` where they can log out, delete or reset their account.

The validation in Figure 4 is displayed after focusing and unfocusing the input or clicking submit. Moreover, notice how the header changes in Figure 5.

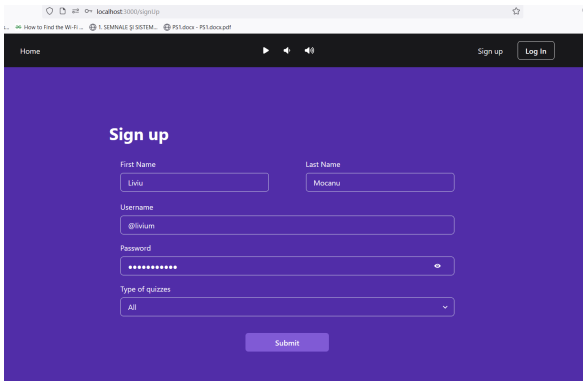
```
function App() {
     return (
         ...

         <main>
             <Box minHeight="100vh" display="flex" flexDirection="column">
                 <Header/>
                 <Box flex="1">
                     <Routes>
                         <Route path="/signUp" element={<Authenticator page={"sign"
    }/>}/>
                         <Route path="/logIn" element={<Authenticator page={"log"
    }/>}/>
                         <Route path="/account" element={<Authenticator page={"account
    "}/>}/>
                         <Route path="/quiz" element={<Authenticator page={"quiz"
    }/>}/>
                         <Route path="/" element={<Authenticator page={"home"}/>}/>
                     </Routes>
                 </Box>
                 <Footer/>
             </Box>
             <Alert/>
         </main>

         ...
     );
}
```

Listing 4: Showcase of the App.js structure, where Authenticator decides what page to render

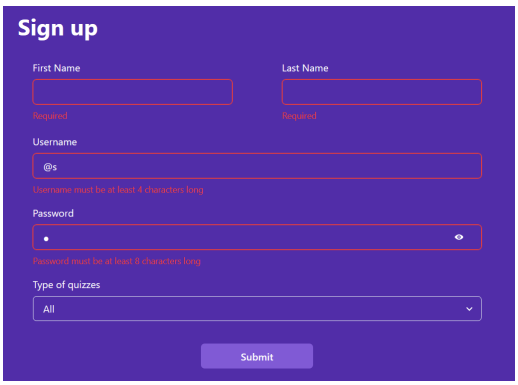
```
const Authenticator = (props) => {
    const {isLoggedIn} = useUserContext();

    const notLoggedInPage = () => {
        switch (props.page) {
            case 'sign':
                return <SignUp/>;
            case 'log':
                return <LogIn/>;
            case 'quiz':
                return <SignUp redirect={true}/>;
            default:
                return <QuizzesSection/>;
        }
    };
```

```
17
18      const loggedInPage = () => {
19          switch (props.page) {
20              case 'account':
21                  return <LoggedIn/>;
22              case 'quiz':
23                  return localStorage.getItem("quizId") ? <Quiz/> : <QuizzesSection/>;
24              default:
25                  return <QuizzesSection/>;
26          }
27      };
28
29      return isLoggedIn ? loggedInPage() : notLoggedInPage();
30  };
31
32  export default Authenticator;
```

Listing 5: Authenticator.js deciding on what page to render



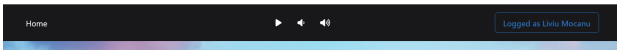
(a) Sign Up page



(b) Sign Up page validation

Figure 4: The Sign Up page which is also being Validated

(a) Header while not logged in



(b) Header after logging

Figure 5: The Header buttons after logging in.

```
1
2   const LogIn = () => {
3       ...
4
5       return (
6           ...
7
8           <FormControl isInvalid={formik.touched.password && formik.errors.password} pb
    ={5}>
9               <FormLabel htmlFor="password">Password</FormLabel>
10              <InputGroup>
11                  <Input
12                      id="password"
13                      name="password"
14                      type={showPassword ? "text" : "password"}
15                      {...formik.getFieldProps('password')}
16                  />
```

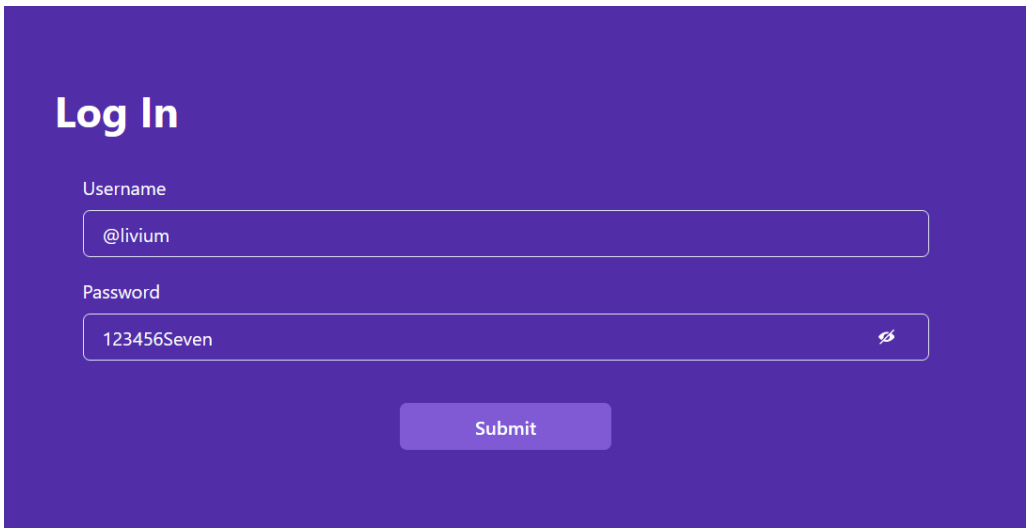


Figure 6: Log In page

(a) Hidden Password (default)

(b) Revealed Password

Figure 7: Password can be revealed or stay hidden with the click of the icon

```
17                  <InputRightElement width="4.5rem">
18                      <Button h="1.75rem" size="sm" onClick={togglePasswordVisibility}
    variant={"none"}>
19                          {showPassword ? <ViewOffIcon /> : <ViewIcon />}
20                      </Button>
21                  </InputRightElement>
22              </InputGroup>
23              <FormErrorMessage>{formik.errors.password}</FormErrorMessage>
24          </FormControl>
25          <Button type="submit" colorScheme="purple" width="25%">
26              {isLoading ? <Spinner/> : "Submit"}
27          </Button>
28
29          ...
30      );
31  }
```

Listing 6: Revealing and hiding the password

2.2 Playing a quiz

If the user is logged in, they can attempt a quiz out of the featured ones.

There exists some validation so that all the questions have an answer before submitting the quiz, after which the submit button disappears and is replaced by the User's score on the quiz, as well as being shown which answers were wrong and what was their correct counterpart. The User is also being presented with a new Home button for easier navigation.

The score is shown on the main page afterwards as well as being seen even after logging out and back in, since they are stored in the localStorage per User, as well as their answers, in order to be able to see their last try at all times.

```
1
2   const QuizzesSection = () => {
```

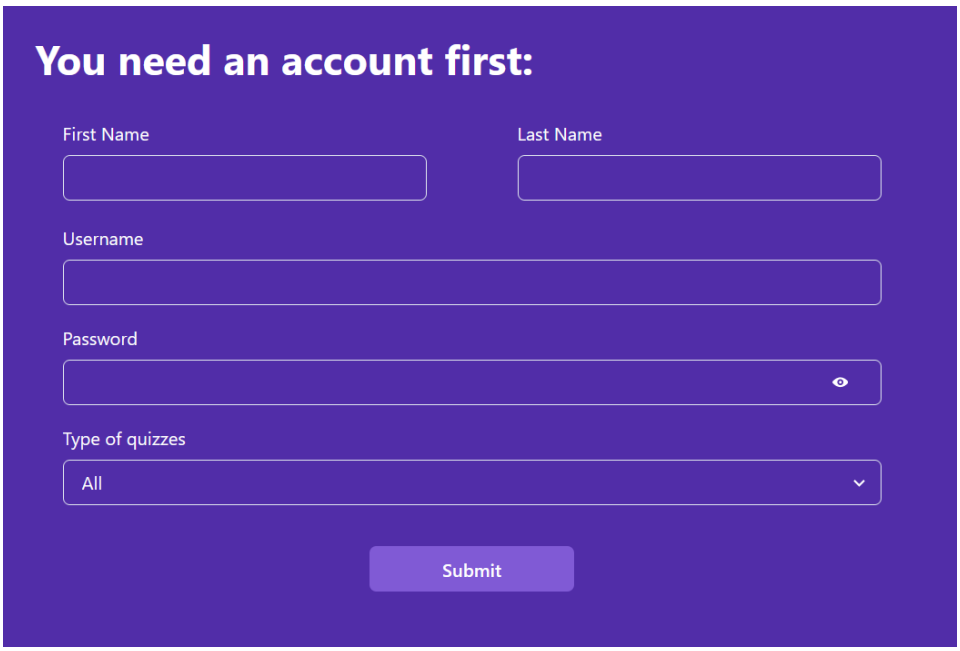


Figure 8: Attempting a quiz while not being Logged In

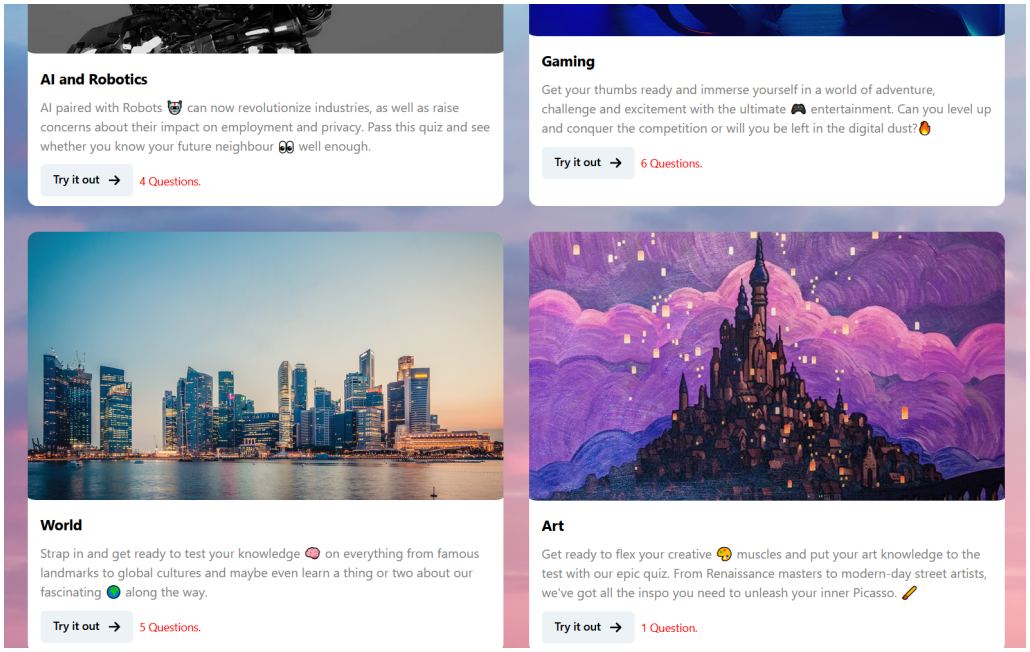


Figure 9: The main page featuring all the quizzes

```
3     const quizArray = useQuizContext();
4
5     return (
6         <FullScreenSection ...>
7             <Heading ...>
8                 Featured Quizzes
9             </Heading>
10            <Box
11                display="grid"
12                gridTemplateColumns="repeat(2,minmax(0,1fr))"
13                gridGap={8}
14            >
15                {
```

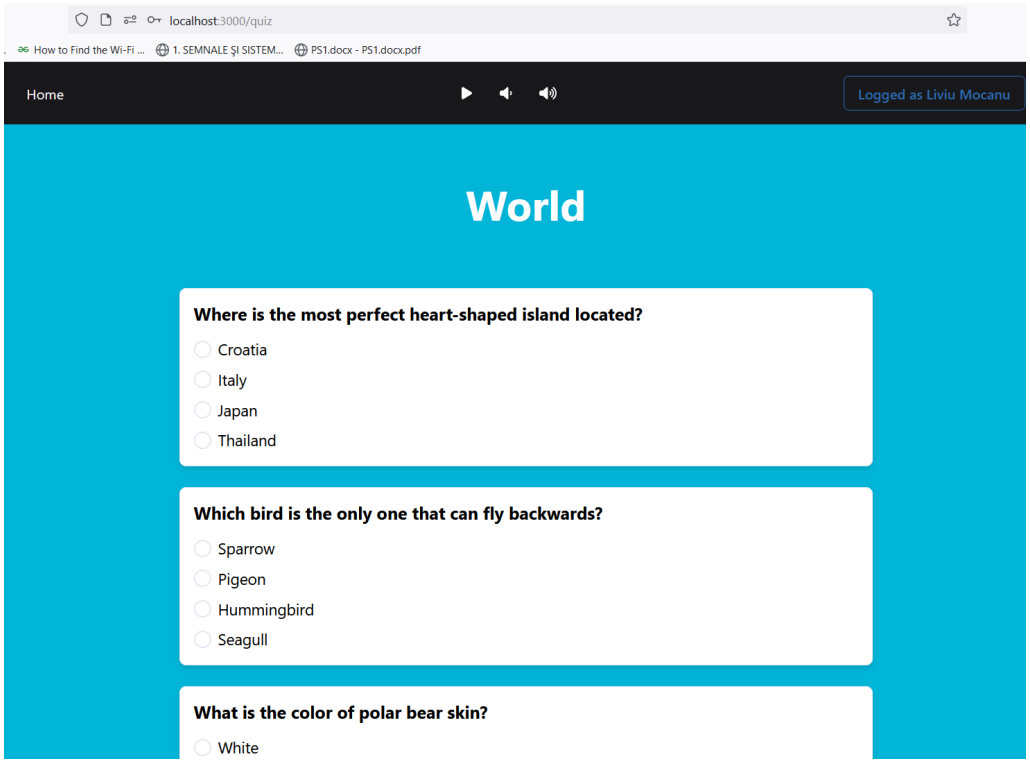


Figure 10: The 'World' Quiz page

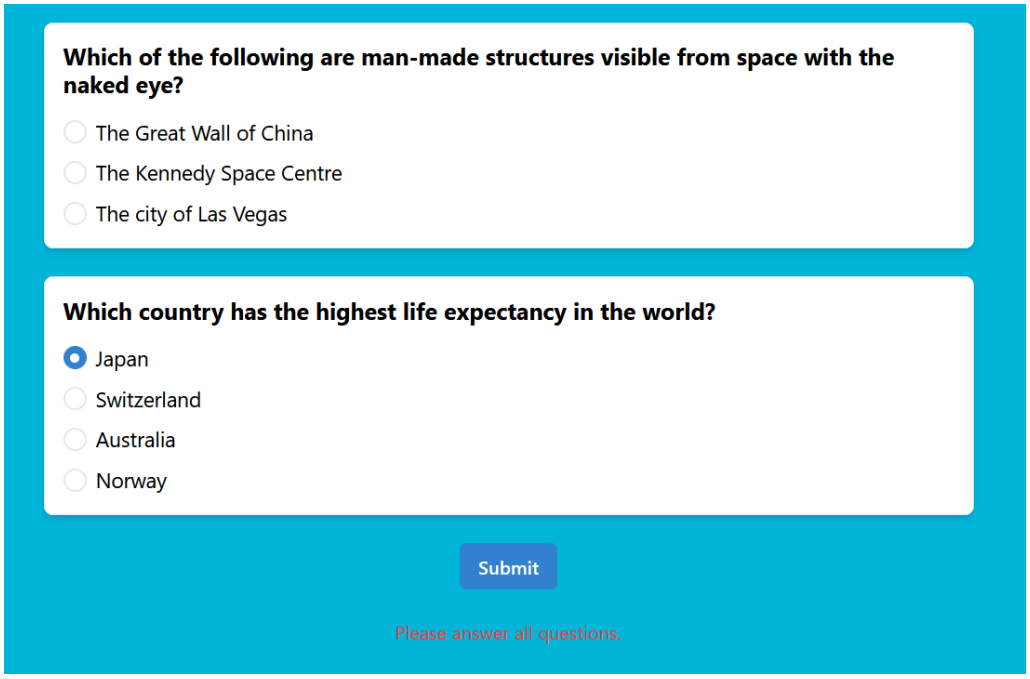


Figure 11: Quiz Validation before submitting

```
16                  quizArray.map((quiz) => {
17                      const loggedUser = JSON.parse(localStorage.getItem("
    loggedInUser")) || {};
18                      const userId = loggedUser.id || null;
19                      const storedQuizScores = JSON.parse(localStorage.getItem("
    quizScore")) || {};
20                      const userScores = storedQuizScores[userId] || {};
21                      const score = userScores[quiz.id] ?? null;
22
```

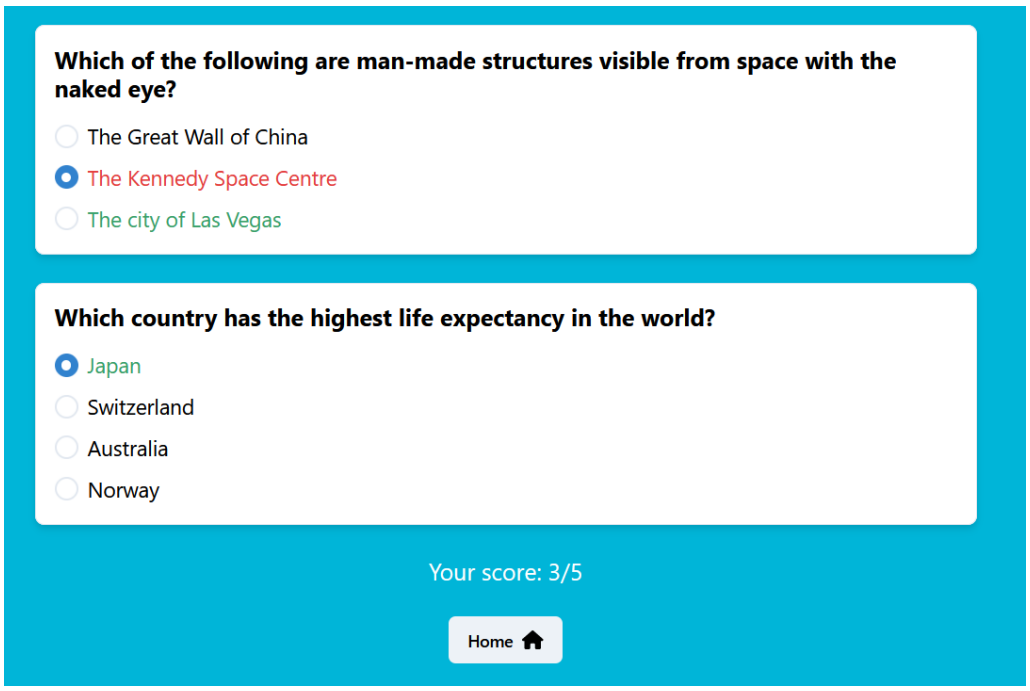


Figure 12: Quiz Feedback after submit

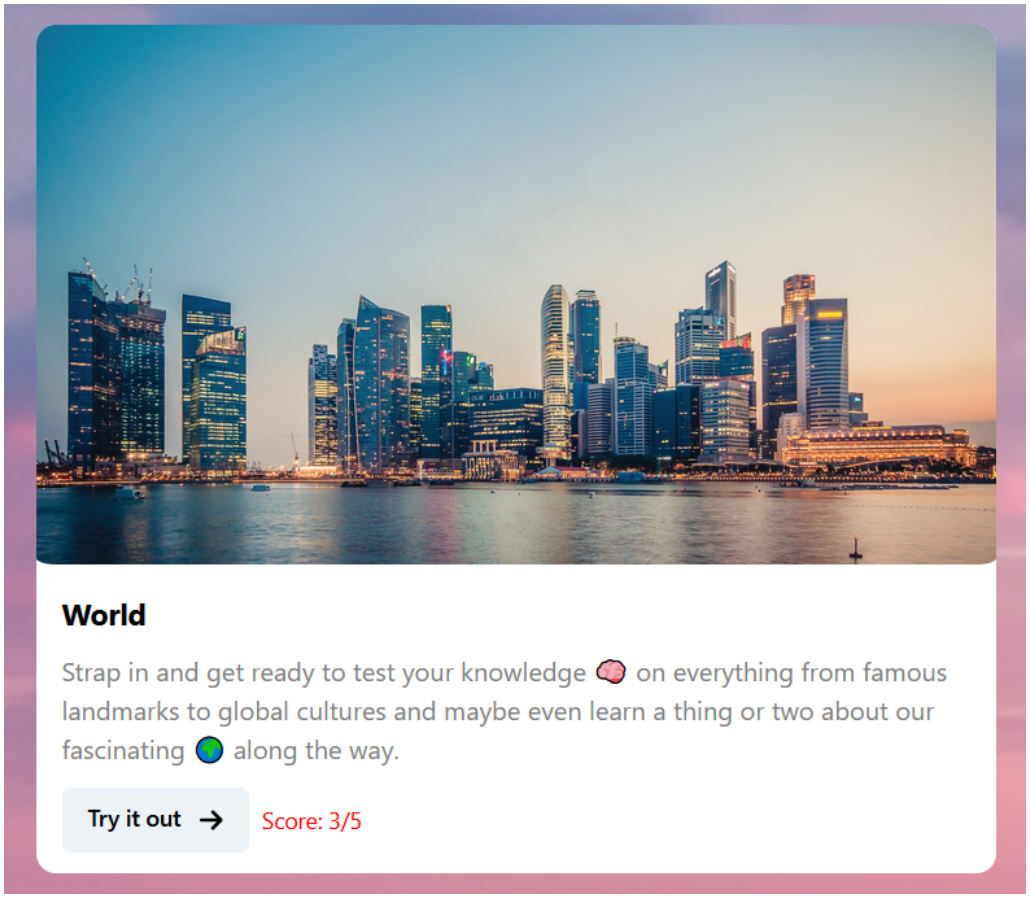


Figure 13: Main Page score update after finishing the quiz

```
23                    return (
24                        <Card
25                            key={quiz.id}
26                            title={quiz.title}
27                            description={quiz.description}
```

```
                                result={score !== null ?
                                    `Score: ${score}/${quiz.questions.length}` :
                                    `${quiz.questions.length} ${quiz.questions.length > 1 ? `Questions.` : `Question.`}`
                                }
                                imageSrc={quiz.getImageSrc()}
                                id={quiz.id}
                            />
                        )
                    })
                }
            </Box>
        </FullScreenSection>
    );
};
```

Listing 7: QuizzesSection mapping the children as Cards

3 Conclusion

This laboratory work for Web Programming involved implementing a quiz application that meets the basic requirements of registering and logging a user, as well as viewing quizzes and play them separately. Additionally, I added several features that enhance the functionality of the application, including a southing background music that doesn't stop on URL change, controlling said music, making the Header appear and disappear depending on scrolling as well as a avatar for the User with their credentials. The application also has an attractive design with a personalized touch.

This laboratory work provided a valuable opportunity for me to improve my knowledge and skills in React. Through this project, I gained a deeper understanding of React concepts such as hooks, state, props and routing. Moreover it was a great introduction to Chakra UI, Formik and Yup.

References

[1] React useEffect hook, *https://legacy.reactjs.org/docs/hooks-effect.html*
Accessed on May 2, 2023.

[2] React Context tutorial, *https://legacy.reactjs.org/docs/context.html*
Accessed on May 3, 2023.

[3] Formik Tutorial, *https://formik.org/docs/tutorial*
Accessed on May 3, 2023.

[4] Formik Yup validation schema tutorial *https://formik.org/docs/guides/validation*
Accessed on May 3, 2023.

[5] Chakra UI Style props, *https://chakra-ui.com/docs/styled-system/style-props*
Accessed on May 4, 2023.