Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Department of Software Engineering and Automation

Web Programming

Laboratory work #4

Quiz app

Author:
Popa Eugeniu
std. gr. FAF-202

Supervisor:
Alexei Șerșun

Chișinău 2023

1 Task

- Pick a frontend framework
- 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
- The app should have attractive UI
- Consume Quiz API to fetch data from backend server

2 Results

This laboratory work was developed using Vue.js.

Vue CLI is used to create and manage the application. I have also used the Vuetify framework, but the components I added (some buttons, input fields and alerts) do not have a predefined complex functionality.

All the necessary logic for the app is implemented, including:

- view a list of quizzes
- view a single quiz
- create a user (and storing its ID somewhere)
- as a logged user, play a predefined quiz
  - submit the responses out of the list (multiple choice)
  - the user can pick a quiz and play it
  - show the total score
- use of my own components

Additional features:

- background music, which you can turn on and off from every page, and it will not turn off on page change
- temporary logged in status, where the user gets logged out after 12 hours

3 Code snippets

This is the main.js file of the app:

```
import { createApp } from "vue";
import App from "./App.vue";
import router from './router'
import store from './store'

import "vuetify/styles";
import { createVuetify } from "vuetify";
import * as components from "vuetify/components";
import * as directives from "vuetify/directives";

import "@mdi/font/css/materialdesignicons.min.css";

const vuetify = createVuetify({
  components,
  directives,
});

createApp(App).use(store).use(router).use(vuetify).mount("#app");
```

The first line imports the createApp function from the Vue.js library, which is used to create a new Vue.js application instance. The App variable is a reference to the root component of the application.

The following lines import the router and store objects from their respective files. router is a Vue.js plugin that enables client-side routing in the application, while store is a centralized state management system that helps manage and share data across components.

The next few lines import Vuetify's components and directives, and then use the createVuetify function to create a new instance of Vuetify. The components and directives variables are passed as arguments to createVuetify so that Vuetify knows which components and directives to register.

Finally, the createApp function is called to create a new Vue.js application instance, which is then passed to the use method along with the store, router, and vuetify objects. The mount method is then called to mount the application on the DOM element with the ID of app.

This is the App.vue component of the app:

```
<template>
  <router-view />
</template>

<script>
import AudioPlayer from "./components/AudioPlayer.vue";

export default {
  components: {
    AudioPlayer,
  },
};
</script>

<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #42b983;
}

html, body {
```

```
  background-color: #271c36;
  height: 100hv;
}
</style>
```

The template section contains a single element, router-view. This element is a placeholder that is used to render the component matching the current route path. When a user navigates to a different route, the router-view element will update to display the appropriate component.

The script section imports the AudioPlayer component from the ./components/AudioPlayer.vue file and adds it to the component's components object. This allows the AudioPlayer component to be used in the template section.

The style section sets some basic styles for the application. The #app selector sets the font-family, text-align, and color for the top-level application element. The html, body selector sets the background color and height for the entire page.

This is the SigupView.vue component:

```
<template>
  <div id="signup-main-container">
    <h1>Sign Up</h1>
    <div id="signup-form-container">
      <v-form ref="form" @submit.prevent="submitForm" class="my-10" style="width: 90%
    ">
        <v-text-field
          v-model="name"
          :rules="nameRules"
          label="Name"
          class="custom-text-field"
        ></v-text-field>
        <v-text-field
          v-model="surname"
          :rules="surnameRules"
          label="Surname"
          class="custom-text-field"
        ></v-text-field>
        <v-btn type="submit" id="submit-button">Submit</v-btn>
      </v-form>
    </div>
    <v-alert v-if="alert" text="Please fill out all the fields" type="warning"
    variant="tonal" class="alert"></v-alert>
    <v-alert v-if="error" text="Error occured. The user already exists or there is no
     access to the internet" type="error" variant="tonal" class="alert"></v-alert>
  </div>
  <AudioPlayer />
</template>

<script>
import axios from 'axios';

import AudioPlayer from "../components/AudioPlayer.vue";

export default {
  name: 'SignupView',
  components: {
    AudioPlayer,
  },
  data() {
    return {
      name: '',
```

```
      nameRules: [
        value => !!value || 'Name is required',
      ],
      surname: '',
      surnameRules: [
        value => !!value || 'Surname is required',
      ],
      formData: {
        name: '',
        surname: '',
      },
      alert: false,
      error: false,
      now: null,
      userData: {
        userId: null,
        date: null,
        time: null,
      },
    }
  },
  created() {
    const userData = localStorage.getItem("userData");
    if (userData) {
      const user = JSON.parse(userData);
      if (user && user.userId && user.userId !== "undefined" && user.date && user.
  time) {
        this.$router.push('/');
      }
    }
  },
  methods: {
    submitForm() {
      if (this.name && this.surname) {
        this.formData.name = this.name;
        this.formData.surname = this.surname;
        this.login();
      } else {
        this.showAlert();
      }
    },
    login() {
      axios.post('https://late-glitter-4431.fly.dev/api/v54/users', {
        data: this.formData
      }, {
        headers: {
          'Content-Type': 'application/json',
          'X-Access-Token': process.env.VUE_APP_ACCESS_TOKEN,
        }
      })
        .then(response => {
          this.now = new Date();
          this.userData.userId = response.data.id;
          this.userData.date = this.now.toLocaleDateString();
          this.userData.time = this.now.toLocaleTimeString();
          localStorage.setItem("userData", JSON.stringify(this.userData));
          this.$router.push('/');
        })
        .catch(error => {
          console.log(error);
```

```
          this.showError();
        });
    },
    showAlert() {
      this.alert = true;
      this.alertTimeout = setTimeout(() => {
        this.alert = false;
      }, 5000);
    },
    showError() {
      this.error = true;
      this.alertTimeout = setTimeout(() => {
        this.error = false;
      }, 5000);
    }
  }
}
</script>

<style>
#signup-main-container {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
}

h1 {
  margin-top: 3%;
  color: #F5F5F5;
}

#signup-form-container {
  width: 50%;
  height: auto;
  margin-top: 3%;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  background-color: #382a4b;
  border-radius: 25px;
}

.custom-text-field .v-messages__message {
  color: red;
}

#submit-button {
  margin-top: 1%;
  background-color: #42b983;
  color: #F5F5F5;
}

.alert {
  width: 20%;
  margin-top: 3%;
}
</style>
```

This component defines a sign-up form with some basic validation and error handling logic.

The <template> section defines the HTML structure of the component, which includes a header, a form with two text fields for name and surname, and a submit button. There are also two v-alert elements that will be conditionally displayed based on the alert and error flags in the component's data.

The <script> section defines the behavior of the component. The import statement brings in an AudioPlayer component from another file. The export default statement defines the main object that represents the component. It has a name property which is set to 'SignupView'. It also has a data property which is a function that returns an object containing various properties such as name, surname, nameRules, surnameRules, formData, alert, error, now, and userData. These properties are used to store data and state of the component.

The created() lifecycle hook checks for the existence of userData in the browser's local storage. If it exists and contains valid data, the user is redirected to another page using this.$router.push('/').

The methods property defines various functions that handle user actions and form submissions. The submitForm() function is called when the user clicks the submit button. It checks if both name and surname fields are filled, and if so, it sets the formData object with the user's input and calls the login() function. If either field is empty, it calls the showAlert() function to display an error message.

The login() function sends a POST request to a specified API endpoint using the axios library. It passes the formData object as the request body, along with an access token as a header. If the request is successful, it stores some user data in the browser's local storage, sets the userData object in the component's data, and redirects the user to another page using this.$router.push('/'). If the request fails, it calls the showError() function to display an error message.

The showAlert() and showError() functions set the alert and error flags in the component's data to true, which triggers the display of the v-alert elements. They also set a timeout to reset the flags after a few seconds, effectively hiding the error messages.

The <style> section defines some basic CSS styles for the component's HTML elements, such as the font color, background color, and margin. It also sets the color of the validation error messages for the text fields, and the color and width of the v-alert elements.

This is the HomeView.vue component:

```
<template>
  <div id="home-main-container">
    <div id="header">
      <AudioPlayer />
      <v-btn id="logout-button" @click="logout">
      <v-icon>{{'mdi-logout'}}</v-icon></v-btn>
    </div>
    <h1 id="main-header">Available Quizzes</h1>
    <QuizList :quizzes="quizzes" />
  </div>
</template>

<script>
import axios from 'axios';

import QuizList from '../components/QuizList.vue'
import AudioPlayer from "../components/AudioPlayer.vue";

export default {
  name: 'HomeView',
  components: {
    QuizList,
    AudioPlayer
  },
```

```
  data() {
    return {
      quizzes: [],
    }
  },
  mounted() {
    this.getQuizzes();
  },
  methods: {
    getQuizzes() {
      axios.get('https://late-glitter-4431.fly.dev/api/v54/quizzes', {
        headers: {
          'X-Access-Token': process.env.VUE_APP_ACCESS_TOKEN,
        }
      })
        .then(response => {
          this.quizzes = response.data;
        })
        .catch(error => {
          console.log(error);
        });
    },
    logout() {
      localStorage.clear();
      this.$router.push('/signup');
    }
  },
}
</script>

<style>
#home-main-container {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
}

#main-header {
  margin: 1% 0 0 0;
  color: #F5F5F5;
}

#logout-button {
  margin: 3% 2% 3% 2%;
  background-color: #b94256;
  color: #F5F5F5;
}

#header {
  width: 20%;
  margin-top: 3%;
  display: flex;
  flex-direction: row;
  justify-content: center;
  align-items: center;
}
</style>
```

This component defines home view of the application. Here is a detailed explanation of the code:

The <template> section defines the structure of the HTML page. It contains several elements, including two custom components AudioPlayer and QuizList. It also has a header section that includes a logout button.

The <script> section contains the JavaScript code for the component. It starts by importing axios, a popular JavaScript library used for making HTTP requests, and two custom components QuizList and AudioPlayer. It also defines a default export object that contains several properties.

The name property sets the name of the component to "HomeView".

The components property lists the custom components that the HomeView component uses.

The data property returns an object that initializes the quizzes array to an empty array. This array will be populated later with data fetched from an API.

The mounted method is a lifecycle hook that is called when the component is mounted to the DOM. In this method, the getQuizzes method is called to fetch the list of available quizzes.

The getQuizzes method sends an HTTP GET request to an API endpoint to fetch a list of quizzes. The request is made using the axios library, which is imported earlier. If the request is successful, the data is assigned to the quizzes array. If the request fails, an error message is logged to the console.

The logout method is called when the user clicks on the logout button. It clears the user's local storage and redirects them to the signup page.

The <style> section defines the CSS styles for the component. It styles the header and logout button, and centers the main content of the page.

This is the QuizView.vue component:

```
<template>
  <div id="quiz-main-container">
    <h1 class="header-text">{{ quizzData.title }}</h1>
    <div id="quiz-container" v-for="(question, index) in quizzData.questions" :key="
    index">
      <h2>{{ question.question }}</h2>
      <div id="answer" v-for="(answer, answerIndex) in question.answers" :key="
    answerIndex"
        :class="{'selected': question.selectedAnswer === answer}"
        @click="selectAnswer(question, answer)">
        <h3>{{ answer }}</h3>
      </div>
    </div>
    <v-alert v-if="alert" text="Please respond to all the questions before submitting
    " type="error" variant="tonal" class="submit-questions-alert"></v-alert>
    <v-btn id="submit-answers-button" @click="submitQuiz">Submit</v-btn>
  </div>
</template>

<script>
import axios from 'axios';

export default {
  name: 'QuizView',
  props: {
    id: String,
  },
  data() {
    return {
      quizzId: '',
      quizzData: [],
      questionData: {
        question_id: Number,
        answer: String,
        user_id: Number
```

```
    },
    questionsNumber: 0,
    counter: 0,
    score: 0,
    alert: false,
    receivedResponses: 0
  }
},
created () {
  if (this.id) {
      this.quizzId = this.id;
  }
},
mounted() {
  this.getQuizData();
},
methods: {
  getQuizData() {
    axios.get(`https://late-glitter-4431.fly.dev/api/v54/quizzes/${this.quizzId}`, {
      headers: {
        'X-Access-Token': process.env.VUE_APP_ACCESS_TOKEN,
      }
    })
      .then(response => {
        this.quizzData = response.data;
        this.quizzData.questions.forEach(question => {
          question.selectedAnswer = '';
          this.questionsNumber++;
        });
      })
      .catch(error => {
        console.log(error);
      });
  },
  selectAnswer(question, answer) {
    question.selectedAnswer = answer;
  },
  submitQuiz() {
    let promises = [];
    this.quizzData.questions.forEach(question => {
      this.quizzData.questions.forEach(question => {
        if (question.selectedAnswer != '') {
          this.counter = this.counter + 1;
        }
      });
      if (this.counter != this.questionsNumber) {
        this.counter = 0;
        this.showAlert();
      }
      else {
        this.counter = 0;
        this.questionData.question_id = question.id;
        this.questionData.answer = question.selectedAnswer;
        const user = JSON.parse(localStorage.getItem("userData"));
        this.questionData.user_id = user.userId;
        promises.push(axios.post(`https://late-glitter-4431.fly.dev/api/v54/quizzes/${this.quizzId}/submit`, {
          data: this.questionData
        }, {
```

```
            headers: {
              'X-Access-Token': process.env.VUE_APP_ACCESS_TOKEN,
            }
          }
          ));
        }
      });

      Promise.all(promises)
        .then(responses => {
          responses.forEach(response => {
            this.receivedResponses = this.receivedResponses + 1;
            if (response.data.correct === true) {
              this.score = this.score + 1;
            }
          });
          if (this.receivedResponses === this.questionsNumber) {
            this.$store.dispatch("saveQuizResult", {
              quizId: this.quizzId,
              score: this.score,
              numQuestions: this.questionsNumber,
            });
            this.$router.push('/');
          }
        })
        .catch(error => {
          console.log(error);
        });
    },
    showAlert() {
      this.alert = true;
      this.alertTimeout = setTimeout(() => {
        this.alert = false;
      }, 5000);
    },
  },
}
</script>

<style>
#quiz-main-container {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
}

h1 {
  margin: 3% 3% 0 3%;
  color: #F5F5F5;
}

h2 {
  margin-bottom: 3%;
}

#quiz-container {
  width: 40%;
  min-width: 20%;
  height: auto;
```

```
  margin-top: 3%;
  padding-top: 3%;
  padding-bottom: 3%;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  background-color: #382a4b;
  border-radius: 25px;
}

#answer {
  width: 50%;
  min-width: 150px;
  height: 30%;
  margin: 2%;
  padding: 1%;
  justify-content: center;
  align-items: center;
  background-color: #271c36;
  border-radius: 25px;
  cursor: pointer;
}

#answer.selected {
  background-color: #42b983;
  color: #F5F5F5;
}

#submit-answers-button {
  margin: 3%;
  background-color: #42b983;
  color: #F5F5F5;
}

.top-left-image {
  position: absolute;
  top: 0;
  right: 0;
}

.submit-questions-alert {
  width: 30%;
  margin-top: 2%;
}
</style>
```

This component is responsible for rendering a quiz interface with a list of questions and their corresponding answers.

The component receives a prop called ”id”, which represents the ID of the quiz to be displayed. When the component is created, it assigns the value of the ”id” prop to the ”quizzId” data property.

When the component is mounted, it calls the ”getQuizData” method to retrieve the quiz data from an API endpoint. This method makes an HTTP GET request to the API endpoint, passing the ”quizzId” and an access token in the request headers. When the API response is received, the quiz data is assigned to the ”quizzData” data property. Additionally, the ”questionsNumber” data property is set to the number of questions in the quiz, and each question object in the ”quizzData” array is augmented with a ”selectedAnswer” property initialized to an empty string.

The template of the component defines the HTML structure of the quiz interface. The quiz

title is displayed at the top, followed by a container for each question, which displays the question text and a list of answer options. Each answer option is rendered as a clickable div with a class of "answer". When an answer is selected, its corresponding question object's "selectedAnswer" property is updated to the selected answer.

When the "submitQuiz" method is called (by clicking the "Submit" button), it loops through each question in the quiz and checks if an answer has been selected for each one. If an answer has not been selected for a question, an error alert is displayed. Otherwise, a POST request is made to another API endpoint to submit the answer for that question. The request data includes the ID of the question, the selected answer, and the ID of the current user (retrieved from local storage). The access token is passed in the request headers. Each POST request is added to an array of promises.

After all the promises have been created, the "Promise.all" method is called to wait for all the requests to complete. Once all requests are complete, the "then" method is called, which loops through each response, increments the "receivedResponses" count, and updates the "score" count if the response is correct. When the "receivedResponses" count matches the "questionsNumber" count, the "saveQuizResult" action is dispatched to save the quiz result to the store, and the user is redirected to the home page.

The "showAlert" method displays an error alert for 5 seconds if the user tries to submit the quiz without answering all the questions.

This is the index.js file which represents the store of the app:

```
import { createStore } from "vuex";
import audio from "./modules/audio";

export default createStore({
  state: {
    quizResults: [],
  },
  getters: {},
  mutations: {
    addQuizResult(state, result) {
      state.quizResults.push(result);
    },
  },
  actions: {
    saveQuizResult({ commit }, { quizId, score, numQuestions }) {
      const quizResult = {
        quizId,
        score,
        numQuestions,
      };
      localStorage.setItem(`quizResult-${quizId}`, JSON.stringify(quizResult));
      commit("addQuizResult", quizResult);
    },
  },
  modules: {
    audio,
  },
});
```

This is a Vuex store module for managing state and actions related to a quiz application.

The code imports the createStore function and a module named audio from Vuex. It then creates a new Vuex store using the createStore function and exports it as the default export.

The store's state contains a quizResults array that will store the results of completed quizzes.

The store has a single mutation addQuizResult, which takes the state and a result parameter. It adds the result object to the quizResults array in the state.

The store's only action is saveQuizResult, which takes a context object and a payload object with the quizId, score, and numQuestions properties.

The action creates a new quizResult object with the payload data, saves it to localStorage with a key that includes the quizId, and then commits the addQuizResult mutation with the quizResult object as the payload.

The store also includes a modules property that imports the audio module. This allows for separate state, mutations, and actions related to audio to be managed within the same store.

This is the audio.js file which represents the audio module:

```
const state = {
  audio: null,
  isMuted: true
}

const mutations = {
  SET_AUDIO(state, audio) {
    state.audio = audio
  },
  SET_MUTED(state, isMuted) {
    state.isMuted = isMuted
    state.audio.muted = isMuted
  }
}

const actions = {
  createAudio({ state, commit }, { src, loop, autoplay, volume }) {
    if (!state.audio) {
      const audio = new Audio(src);
      audio.loop = loop;
      audio.autoplay = autoplay;
      audio.volume = volume;
      commit("SET_AUDIO", audio);
    }
  },
  playAudio({ state }) {
    if (state.audio.paused) {
      state.audio.play();
    }
  },
  pauseAudio({ state }) {
    if (!state.audio.paused) {
      state.audio.pause();
    }
  },
};

export default {
  namespaced: true,
  state,
  mutations,
  actions
}
```

This is a Vuex module that manages state and actions related to an audio player.

The module exports an object with the following properties:

namespaced is set to true, which means that the module's mutations and actions will be scoped under a namespace, preventing naming conflicts with other modules.

state is an object that contains two properties:

audio: a reference to an Audio object or null if no audio has been created. isMuted: a boolean indicating whether the audio is currently muted. mutations is an object that contains two mutation functions:

SET_AUDIO: takes the state and an audio parameter and sets the audio property of the state to the audio parameter. SET_MUTED: takes the state and an isMuted parameter, sets the isMuted property of the state to the isMuted parameter, and sets the muted property of the audio object to the isMuted parameter. actions is an object that contains three action functions:

createAudio: takes a context object and a payload object with the src, loop, autoplay, and volume properties. If the audio property of the state is null, it creates a new Audio object with the provided properties, sets the properties of the Audio object accordingly, and commits the SET_AUDIO mutation with the Audio object as the payload. playAudio: takes a context object and plays the audio object if it is paused. pauseAudio: takes a context object and pauses the audio object if it is playing. The module allows for the creation of a single Audio object and provides actions to play, pause, and mute/unmute the audio.

This is the index.js file which represents the router of the app:

```
import { createRouter, createWebHistory } from "vue-router";
import HomeView from "../views/HomeView.vue";
import SignupView from "../views/SignupView.vue";
import QuizView from "../views/QuizView.vue";

const routes = [
  {
    path: "/",
    name: "home",
    component: HomeView,
    meta: { requiresAuth: true },
  },
  {
    path: "/signup",
    name: "signup",
    component: SignupView,
  },
  {
    path: "/quiz/:id",
    name: "quiz",
    component: QuizView,
    props: true,
  }
];

const router = createRouter({
  history: createWebHistory(),
  routes,
});

router.beforeEach((to, from, next) => {
  const requiresAuth = to.matched.some((record) => record.meta.requiresAuth);
  const userData = localStorage.getItem("userData");
  if (requiresAuth && !userData) {
    if (to.path === "/signup") {
      next();
    } else {
      next("/signup");
    }
  } else {
```

```
    const user = JSON.parse(userData);
    if (user && user.userId && user.userId !== "undefined" && user.date && user.time)
    {
      const currentDate = new Date();
      const savedDate = new Date(user.date + " " + user.time);
      if (currentDate.toDateString() === savedDate.toDateString()) {
        if (currentDate.getTime() - savedDate.getTime() >= 12 * 60 * 60 * 1000) {
          localStorage.clear();
          if (to.path === "/signup") {
            next();
          } else {
            next("/signup");
          }
        } else {
          next();
        }
      } else {
        localStorage.clear();
        if (to.path === "/signup") {
          next();
        } else {
          next("/signup");
        }
      }
    } else {
      localStorage.clear();
      if (to.path === "/signup") {
        next();
      } else {
        next("/signup");
      }
    }
  }
});

export default router;
```

This code defines a Vue Router instance and specifies the routes for the application.

First, the code imports the createRouter and createWebHistory functions from the vue-router package. The createRouter function creates a new router instance while the createWebHistory function creates a history object to manage browser history.

Next, the routes array is defined. This array contains route objects that define the path, component, and optional meta properties for each route.

The router.beforeEach method is used to define a navigation guard that is executed before each navigation event. This navigation guard checks if the route being navigated to requires authentication by checking if the requiresAuth property is set to true in the route's meta object. If the route requires authentication, the user's data is retrieved from the local storage using the localStorage.getItem("userData") method.

If the user's data is not found or is invalid, the user is redirected to the signup page using the next("/signup") method. If the user's data is found and is valid, the code checks whether the user has been authenticated within the last 12 hours. If the user has been authenticated within the last 12 hours, the user is allowed to proceed with the navigation. Otherwise, the user is redirected to the signup page.

The code also defines dynamic routes using the :id syntax in the quiz route. The props: true property allows the id parameter to be passed as a prop to the QuizView component.

Finally, the router object is exported as the default export of the module.

Overall, this code implements a simple authentication and authorization mechanism by requiring users to sign up and log in before accessing certain routes in the application. It also ensures that users are logged out after a certain period of inactivity.

This is the QuizList.vue component:

```
<template>
  <div id="quiz-list-container">
    <div class="quiz-preview" v-for="(quiz, index) in quizzes" :key="index" :class="{
     'quiz-completed': isQuizCompleted(quiz.id) }" @click="!isQuizCompleted(quiz.id)
    && navigateToQuiz(quiz.id)">
      <div class="quiz-title">
        <h2>{{ quiz.title }}</h2>
      </div>
      <div class="quiz-body">
        <p v-if="!isQuizCompleted(quiz.id)">Questions count: {{ quiz.questions_count
    }}</p>
        <p v-if="isQuizCompleted(quiz.id)">Score: {{ getQuizScore(quiz.id) }}</p>
      </div>
    </div>
  </div>
</template>

<script>
export default {
  name: 'QuizList',
  props: {
    quizzes: Array,
  },
  methods: {
    navigateToQuiz(quizId) {
      this.$router.push({
        name: "quiz",
        params: {
          id: quizId,
        },
      });
    },
    isQuizCompleted(quizId) {
      return localStorage.getItem(`quizResult-${quizId}`) !== null;
    },
    getQuizScore(quizId) {
      const quizResult = JSON.parse(localStorage.getItem(`quizResult-${quizId}`));
      return `${quizResult.score} / ${quizResult.numQuestions}`;
    }
  }
}
</script>

<style>
#quiz-list-container {
  width: 80%;
  height: auto;
  margin-top: 3%;
  margin-bottom: 3%;
  padding-top: 3%;
  padding-bottom: 3%;
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
  justify-content: space-evenly;
```

```
  background-color: #382a4b;
  border-radius: 25px;
}

.quiz-preview {
  width: 25%;
  min-width: 150px;
  height: auto;
  min-height: 125px;
  margin: 20px;
  padding: 20px;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  color: #382a4b;
  background-color: #42b983;
  border-radius: 25px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
  cursor: pointer;
  transition: background-color 0.2s ease-in-out;
}

.quiz-preview:hover {
  background-color: #368a68;
}

.quiz-preview.quiz-completed {
  background-color: #b94256;
  cursor: default;
}

.quiz-preview.quiz-completed:hover {
  background-color: #8a363e;
}

.quiz-title {
  margin-bottom: 10px;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.quiz-title h2 {
  margin: 0;
  font-size: 24px;
  font-weight: bold;
}

.quiz-body p {
  color:#F5F5F5;
  font-size: 18px;
  margin: 0;
}
</style>
```

This code defines a Vue.js component called ”QuizList”, which displays a list of quizzes. The component receives an array of quizzes as a prop.

The component template consists of a div with id ”quiz-list-container” that contains a loop using the v-for directive to iterate through each quiz in the quizzes array. For each quiz, a div with class

”quiz-preview” is created. The class is set to ”quiz-completed” if the quiz is completed, which is determined by checking if the quiz result is stored in the local storage.

The ”quiz-preview” div displays the quiz title and either the number of questions or the quiz score depending on whether the quiz has been completed or not. The quiz preview div is also clickable, and if the quiz is not completed, it will navigate to the quiz page by calling the ”navigateToQuiz” method with the quiz id.

The component script contains three methods: ”navigateToQuiz”, ”isQuizCompleted”, and ”getQuizScore”. The ”navigateToQuiz” method uses the Vue.js router to navigate to the quiz page by calling the $router.push method with the quiz id as a parameter.

The ”isQuizCompleted” method checks whether the quiz result is stored in the local storage by calling the localStorage.getItem method with the quiz result key. If the quiz result is not null, it means that the quiz has been completed.

The ”getQuizScore” method retrieves the quiz result from the local storage by calling the localStorage.getItem method with the quiz result key. The quiz score is then returned as a string in the format ”score / numQuestions”.

The component style defines the layout and style of the ”quiz-preview” div. The quiz preview is styled with a green color if it is not completed and a red color if it is completed. The quiz preview also changes its background color when hovered over with the cursor.

This is the AudioPlayer.vue component:

```
<template>
    <v-btn id="audio-button" :color="isMuted ? '#b94256' : '#42b983'" @click="
    toggleAudio">
      <v-icon>{{ isMuted ? 'mdi-volume-off' : 'mdi-volume-high' }}</v-icon>
    </v-btn>
</template>

<script>
export default {
  computed: {
    audio() {
      return this.$store.state.audio.audio;
    },
    isMuted() {
      return this.$store.state.audio.isMuted;
    },
  },
  mounted() {
    this.$store.dispatch("audio/createAudio", {
      src: require("../music/audio.mp3"),
      loop: true,
      autoplay: false,
      volume: 0.5,
    });
  },
  methods: {
    toggleAudio() {
      if (this.isMuted) {
        this.audio.play();
      } else {
        this.audio.pause();
      }
      this.$store.commit("audio/SET_MUTED", !this.isMuted);
    },
  },
  watch: {
    isMuted(newValue) {
```

```
      const button = document.getElementById("audio-button");
      button.style.backgroundColor = newValue ? "#42b983" : "#b94256";
    },
  },
};
</script>

<style>
  #audio-button {
    margin: 3% 2% 3% 2%;
  }
</style>
```

The template of the component defines a button with an ID of "audio-button" that has a dynamic color based on whether the audio is muted or not (:color="isMuted ? '#b94256' : '#42b983'"). The button also has an icon that changes between a volume off icon and a volume high icon based on whether the audio is muted or not (isMuted ? 'mdi-volume-off' : 'mdi-volume-high').

The component's script section defines a computed property audio that returns the audio object from the Vuex store's audio module. It also defines a computed property isMuted that returns the isMuted property from the same Vuex store module.

The mounted() lifecycle hook dispatches an action createAudio to the Vuex store's audio module with the audio source, loop, autoplay and volume options. This creates an HTML5 Audio object and stores it in the Vuex store's audio module.

The toggleAudio() method plays or pauses the audio and toggles the isMuted property in the Vuex store's audio module by committing a mutation SET_MUTED.

The watch section watches for changes to the isMuted property in the Vuex store's audio module and changes the background color of the button based on whether the audio is muted or not.

The component's style section defines a margin for the button.

4 Conclusion

In conclusion, completing this laboratory work is a great way to gain hands-on experience in developing a web application that consumes a RESTful API.

The lab work covers important aspects of web development, such as creating a user interface with attractive design, consuming a Quiz API, implementing user authentication, and managing state. It also encourages the use of third-party packages and testing the API using Postman.

Overall, completing this lab work helped me gain practical knowledge in developing web applications and enhance my skills in using Vue.js and related tools.

5 GitHub

`https://github.com/eugencic/quiz-app`

6 Screenshots

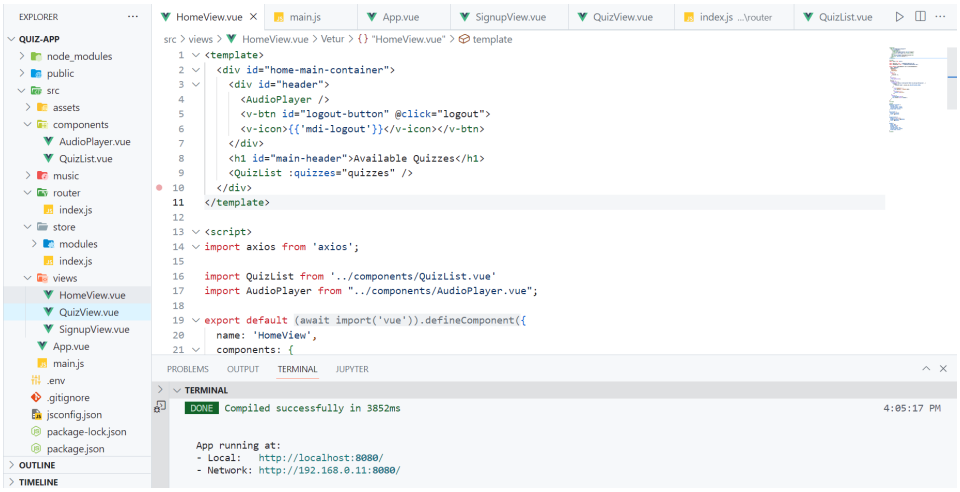

Figure 1: Project structure

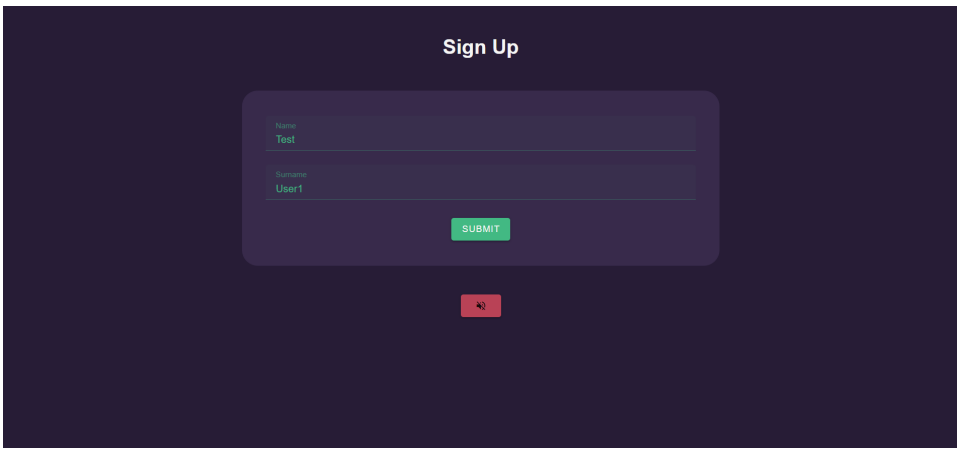

Figure 2: Sign up page

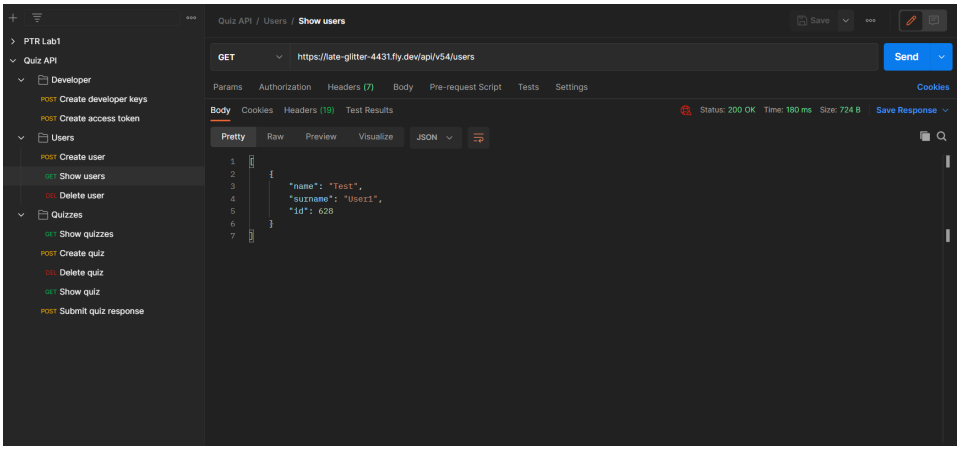

Figure 3: Example of a new added user

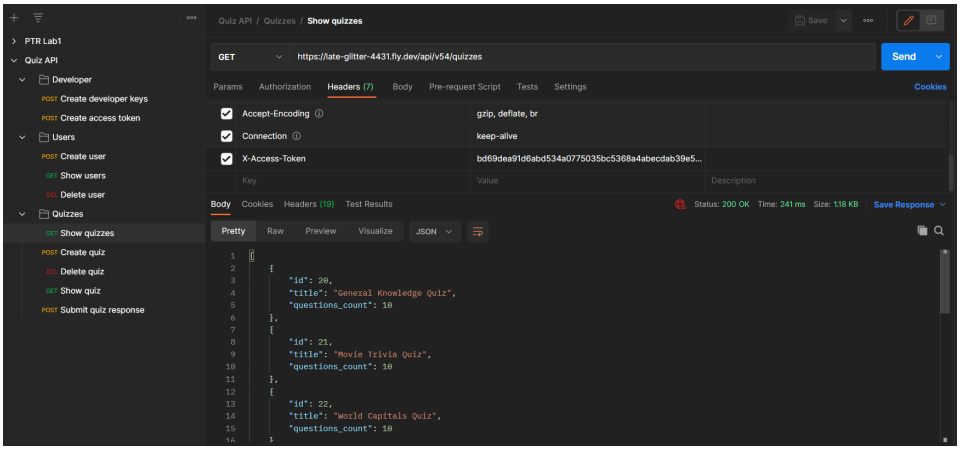

Figure 4: Example of a list of quizzes

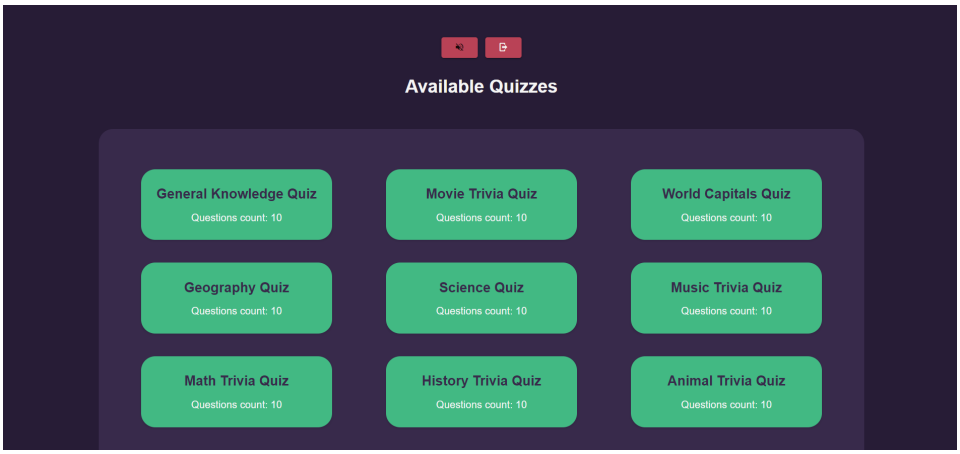

Figure 5: Main page

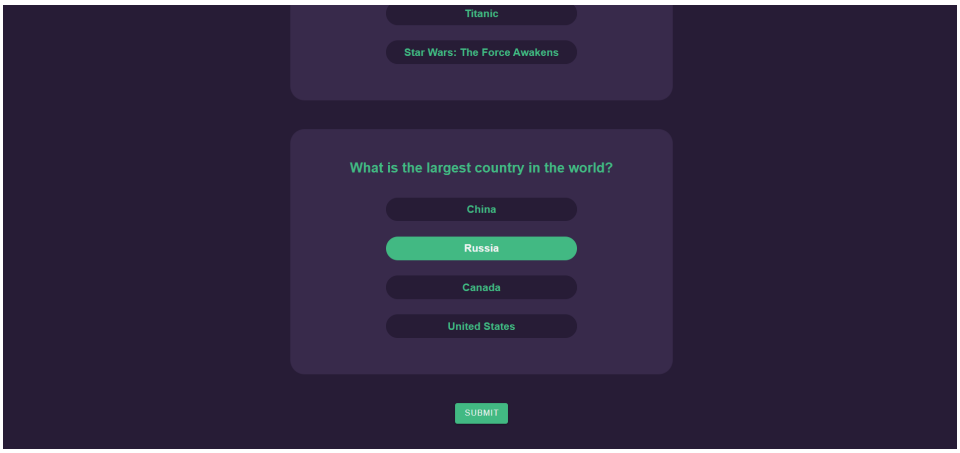

Figure 6: Example of completing a quiz

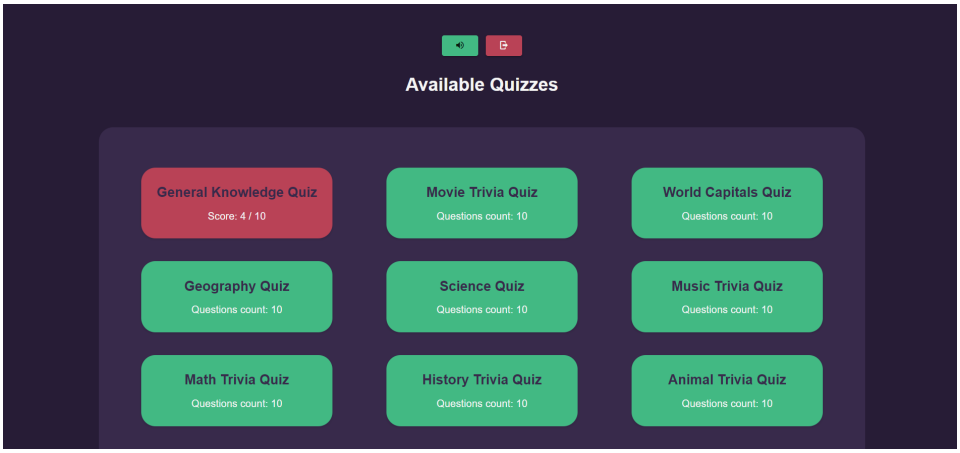

Figure 7: Example of a completed quiz

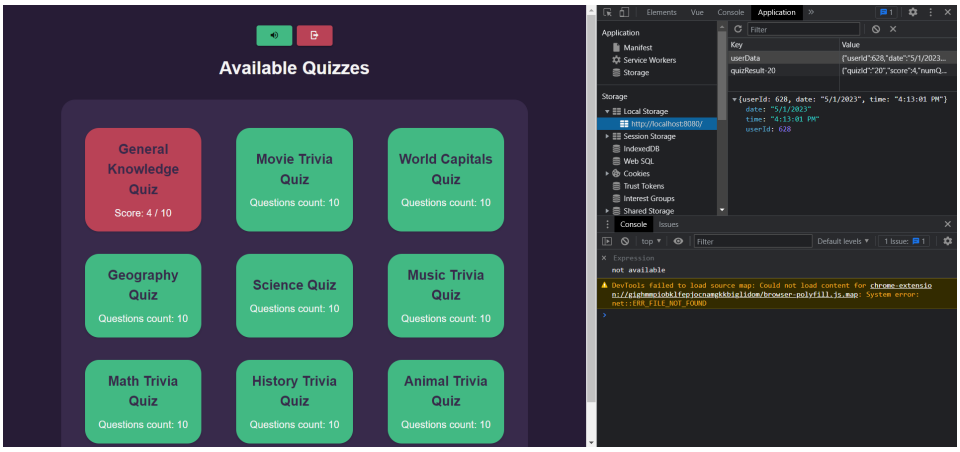

Figure 8: User and quiz data