Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

Особливості розробки веб-застосунків з використанням технології React на прикладі онлайн тесту

Текстова частина до курсової роботи

за спеціальністю „Інженерія Програмного Забезпечення”

Керівник курсової роботи

Борозенний С.О.

_____

(підпис)

“____” _____ 2023 р.

Виконав студент

Кривошеєв І.С.

“____” _____ 2023 р.

КИЇВ 2023

Зміст

Вступ

У сучасному світі існує багато технологій для створення веб-застосунків. Згідно даних платформ StackOverflow і Github, React є найпопулярнішою та однією з найулюбленіших бібліотек серед фреймворків та бібліотек, які застосовують для створення фронтенду веб-додатків.[12] Метою даної роботи є створення сучасного веб-застосунку з використанням бібліотеки React.js, використання можливостей React.js для створення односторінкового застосунку (*single-page application*) – популярної технології створення веб-застосунків, у якому весь необхідний код завантажується разом зі сторінкою, а сторінка не оновлюється і не перенаправляє користувача до іншої сторінки у процесі роботи з нею. За тему створеного застосунку було обрано онлайн квіз, який обирає питання з різних публічних API. Кожне питання має обмеженість у часі та бали, які можна заробити за це питання. Після завершення є можливість огляду питань в квізі. Для відображення основних компонентів UI та розмітки сторінки було обрано бібліотеку Bootstrap, React-Bootstrap використано для легшого поєднання Bootstrap разом з React. Також будуть показані можливості автоматизованого модульного тестування React застосунків за допомогою фреймворку тестування Jest та бібліотеки React Testing Library.

Анотація

Роботу присвячено розробці веб-додатків за допомогою бібліотеки React. Було створено застосунок з використанням даної технології, також були використанні бібліотеки інтерфейсу Bootstrap. Застосунок створений у вигляді онлайн квізу з випадковими питаннями з можливістю специфікації теми та складності. Також будуть показані можливості створення модульних тестів на React компоненти та JavaScript функції за допомогою фреймворку тестування Jest та бібліотеки React Testing Library.

Розділ 1. Огляд існуючих рішень

1.1 Trivia API

Trivia API – публічний API з 9828 питаннями, розділеними на 10 різних категорій. Є публічний ендпоінт який видає до 50 різних випадкових питань. Питання мають лише одну правильну відповідь та можуть мати від 2 до 4 варіантів відповідей. Також є можливість вибрати складність питань, категорію питань, теги присутні в питаннях. Також створений веб-застосунок для показу можливостей API (Рисунок 1).

Застосунок видає 10 випадкових відповідей, показує кількість правильних відповідей. При цьому немає можливості перегляду питань після закінчення квізу, не показується правильна опція після відповіді на питання.[9]

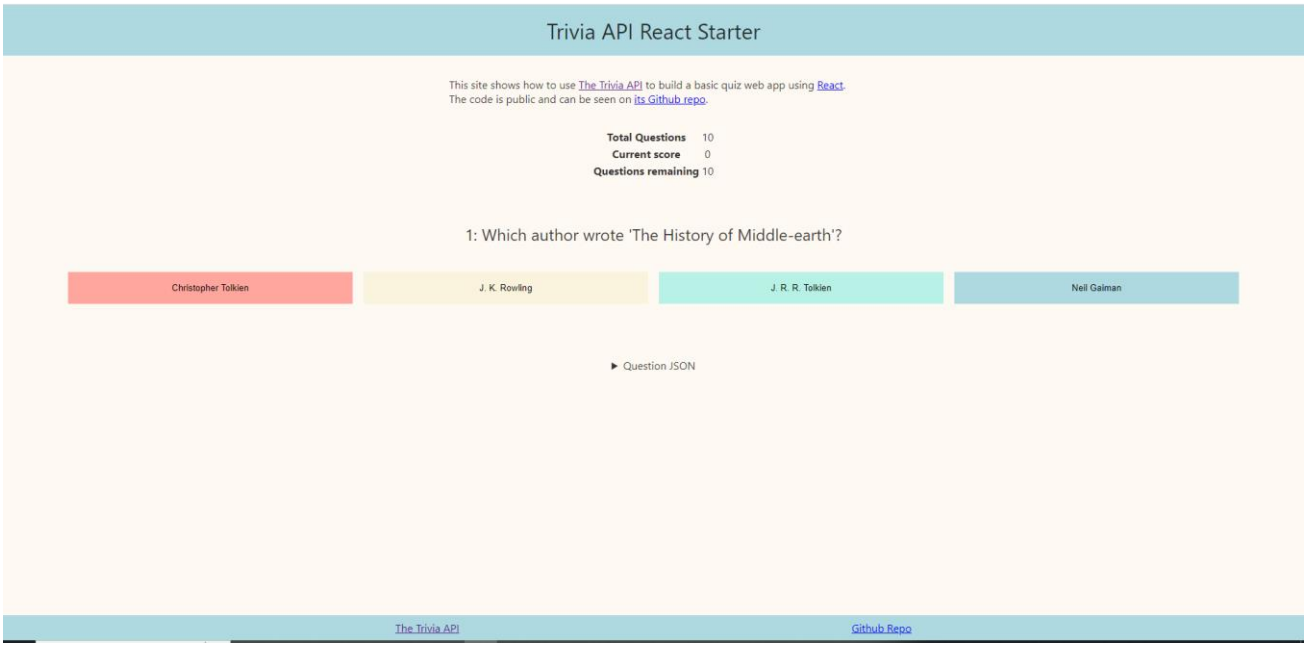


Рисунок 1. Trivia API React Starter

1.2 Open Trivia DB

Open Trivia Database надає абсолютно безкоштовний JSON API для використання в будь-яких проектах. Всього доступно 4098 питань. Для використання цього API не потрібен ключ API, можна робити запити для видачі до 50 випадкових запитань на будь-які теми. Також можна робити запит по конкретній категорії питань або по конкретній складності. Питання мають одну правильну відповідь та від 2 до 4 варіантів відповідей.

При цьому питання в Open Trivia DB може містити текст який містить символи Юнікод або інші спеціальні символи. З цієї причини API повертає результати в закодованому форматі. Всього є 4 формати:

- За замовчуванням (HTML Codes)
- Застаріле URL кодування
- URL кодування (RFC 3986)
- Base64 кодування

Через це при обробці відповідей цього API необхідно декодувати текст з відповідного формату. Також ресурс не надає веб-застосунку для тестування API.[10]

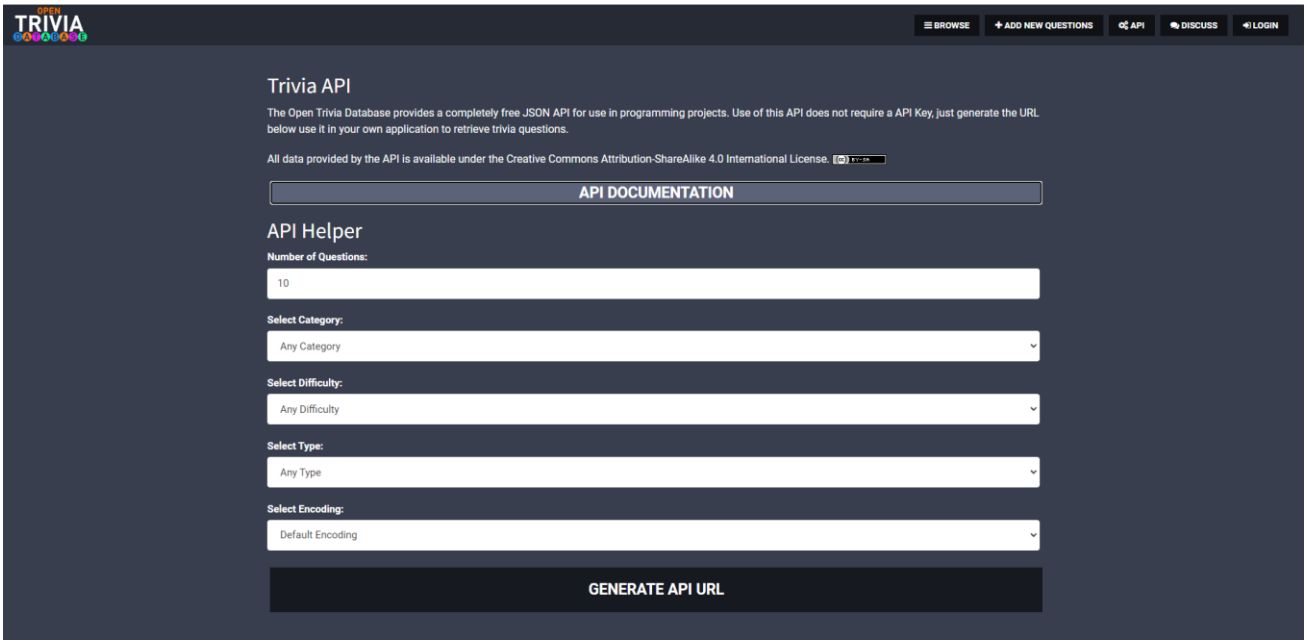


Рисунок 2. Open Trivia API

1.3 QuizAPI

Quiz API — це простий HTTP REST API для технічних тестів, включаючи широкий спектр тем, як-от:

- Linux
- DevOps
- Networking
- Програмування
- Хмарні технології
- Docker
- Kubernetes
- Та інші

API надає доступ до випадкових питань по даним темам, має можливість фільтрації тестів за категорією або складністю. Кожне питання має від 2 до 6 можливих відповідей та може мати декілька правильних. Для доступу до API потрібен ключ, проте його можна безкоштовно сформувати після реєстрації на

платформі. Також доступний веб-застосунок для проходження випадкових квізів (Рисунок 3).

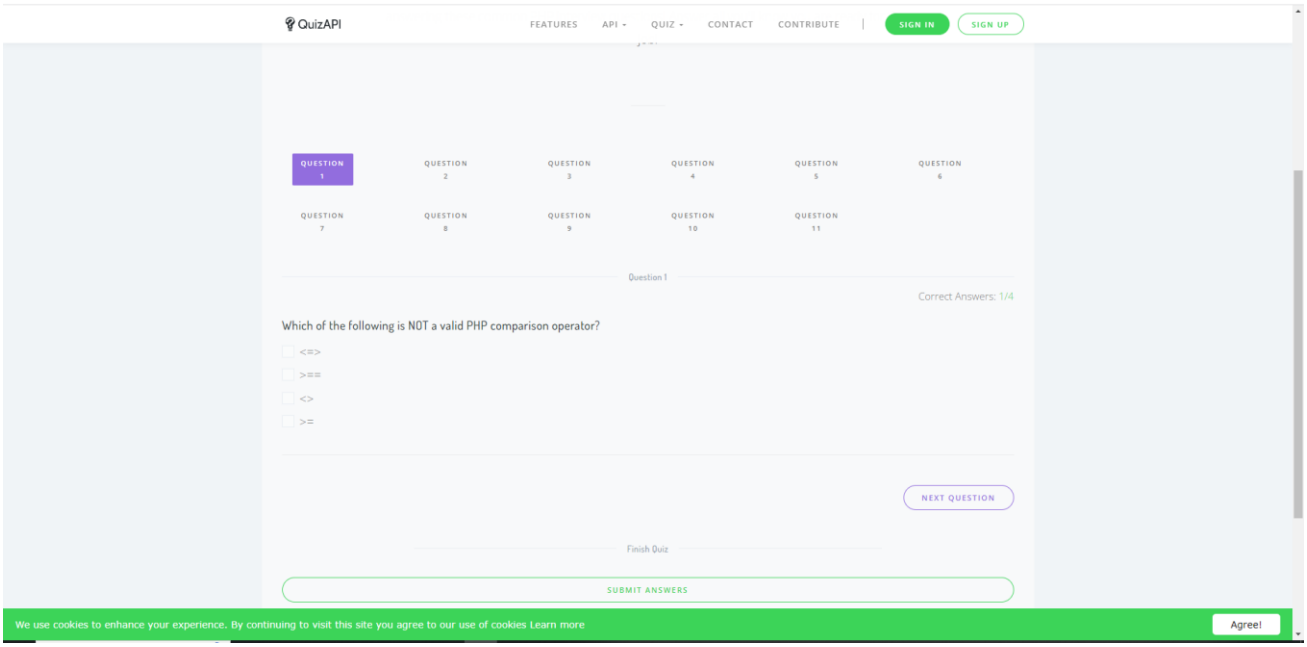


Рисунок 3. QuizAPI

У застосунку є можливість перегляду результатів квізу, можна побачити які відповіді були правильними, їх кількість. При цьому інтерфейс відповідей завжди показує, що питання може мати декілька правильних відповідей, не дивлячись на те, що більшість питань має лише одну правильну відповідь. Також через те, що API знаходиться у розробці при його тестуванні були виявлені деякі помилки. Так, для деяких запитань не вказана їх категорія, немає переліку всіх категорій.[11]

Розділ 2. Огляд використаних технологій та інструментів

2.1 React

React – одна з найпопулярніших JavaScript бібліотек для створення складних фронтенд застосунків і користувацьких інтерфейсів. React є лаконічним та достатньо простим для освоєння, при цьому завдяки його популярності існує велика кількість допоміжних бібліотек, а сама бібліотека має докладну документацію. Також React можна застосовувати не лише для веб-інтерфейсів, а й для мобільних додатків, використовуючи React Native.

Розробка застосунків на React достатньо сильно відрізняється від розробки на звичайному JavaScript, бо React дотримується парадигми декларативного програмування. Для створення інтерфейсу потрібно лише описати, як різні частини інтерфейсу виглядають у кожному стані додатку і React ефективно оновить та відрендерить лише потрібні компоненти, коли дані зміняться.

Для ефективного оновлення інтерфейсу використовується віртуальний DOM. Віртуальний DOM - це копія реального DOM, яка зберігається в пам'яті комп'ютера. React здійснює зміни на віртуальному DOM, а потім здійснює перерендеринг на реальному DOM лише в тих випадках, коли змінюється стан компонента. Бібліотека ReactDOM робить можливим взаємодію між віртуальним DOM та реальним DOM, що дозволяє React оновлювати відображення сторінки з мінімальними затратами на ресурси.

Іще одна особливість React – компоненти, які є однією з ключових концепцій React. React дозволяє об’єднати вашу розмітку, CSS і JavaScript у власні «компоненти», які є елементи інтерфейсу користувача для вашої програми і які можна використовувати в різних частинах програми. Також важливо, що розмітка всередині компонентів написана з використанням JSX. JSX - це

розширення синтаксису JavaScript, що дозволяє описувати структуру інтерфейсу користувача за допомогою синтаксису, схожого на HTML. JSX дозволяє вбудовувати JavaScript-код безпосередньо в HTML-подібний код, що дозволяє легко контролювати відображення компонентів за певної умови, а також відображати списки.

Взагалі в React є два типи компонентів – функціональні та компоненти класів. React підтримує обидва типи, але документація рекомендує використовувати лише функціональні компоненти, тому всі компоненти в проекті є функціональними. Такі компоненти є функціями JavaScript, що повертають розмітку JSX. Такі компоненти мають бути “чистими” – для однакових вхідних даних результат завжди має бути однаковим. Під час рендерингу компонентів не має створюватися жодних побічних ефектів і компоненти не можуть залежати один від одного під час рендерингу. Для передачі даних в компонент використовуються пропси – батьківський компонент передає дані, а дочірній компонент їх отримує у вигляді об’єкта – аргумента функції. Для зберігання та зміни стану компонента використовуються хуки – вбудовані React функції.

Деякі з них:

- useState – використовується для зберігання інформації. Повертає масив з двох значень – поточний стан даних та функцію що змінює дані. Приймає аргумент – початковий стан даних
- useRef – також зберігає інформацію, проте зміна даних ref не викликає нового рендерингу компонента.
- useEffect – використовується для синхронізації компонента з зовнішніми системами (наприклад для запиту на бекенд). Приймає два аргументи – функцію з логікою ефекту та масив залежностей. Функція виконується при створені компоненту, а також кожен раз коли одне зі значень в масиві залежностей змінюється.

- useContext – використовується для передачі даних між компонентами в обхід пропсів. Батьківський клас створює компонент – провайдр контексту і передає йому значення. Будь-який дочірній компонент може використати useContext і отримати відповідне значення.
- Власні хуки – власні функції, що використовують React хуки, і потрібні для перевикористання однакової логіки між компонентами, або для інкапсуляції певної логіки.[1]

2.2 Bootstrap

Bootstrap - це набір інструментів для розробки веб-інтерфейсів. Він містить набір готових CSS-стилів і JavaScript-плагінів для створення привабливого і динамічного дизайну веб-сторінок.

Bootstrap був розроблений компанією Twitter з метою створення спільного набору інструментів для розробки веб-додатків з однорідним дизайном. Він включає в себе CSS-стилі для таких елементів, як кнопки, форми, таблиці, навігаційні панелі, картки, модальні вікна і багато іншого.

Також наявні CSS-стилі для розмітки сторінки та додавання відступів і паддінгу для елементів інтерфейсу. Наявна можливість розділення сторінки на ряди і колонки, що спрощує створення складного інтерфейсу. Також розмітка може автоматично підлаштовуватися під різні розміри екрану, збільшуючи кількість елементів в одному рядку для більших екранів.

Bootstrap також містить JavaScript-плагіни для динамічного функціонування елементів на сторінці, таких як меню, випадаючі списки, модальні вікна та інше.[2]

2.3 React-Bootstrap

React-Bootstrap – бібліотека, створена для заміни Bootstrap в React додатках. Кожен елемент переписаний як React компонент без використання бібліотеки jQuery. Використані компоненти React-Bootstrap:

- Contatiner, Row, Column – для розмітки елементів у різні ряди та колонки.
- Nav, NavBar – відображення меню та тексту з верхньої частини сторінки.
- Form, Form.Check.Input – відображення html форм та checkbox/radio button елементів інтерфейсу.
- Button – відображення різнокольорових кнопок.
- Alert, Badge – елементи для більш наглядного відображення текстових елементів інтерфейсу.
- Та інші.[3]

2.4 React-Bus

Невелика React бібліотека для використання дизайн паттерну Event Bus в React (рисунок 4).

Event Bus дозволяє одному компоненту створити подію і передати його в централізоване місце, яка називається шина подій, з якого підписники до цієї шини можуть отримати дані події. Завдяки такому дизайну незалежні компоненти можуть комунікувати, не знаючи про один одного. [4]

Для створення події в бібліотеці використовується хук useBus, який повертає об’єкт, що містить функцію emit, що приймає два аргументи – назву події і дані події. Для підписки до шини використовується хук useListener, який отримує два аргументи – назву події і функцію обробник події.[5]

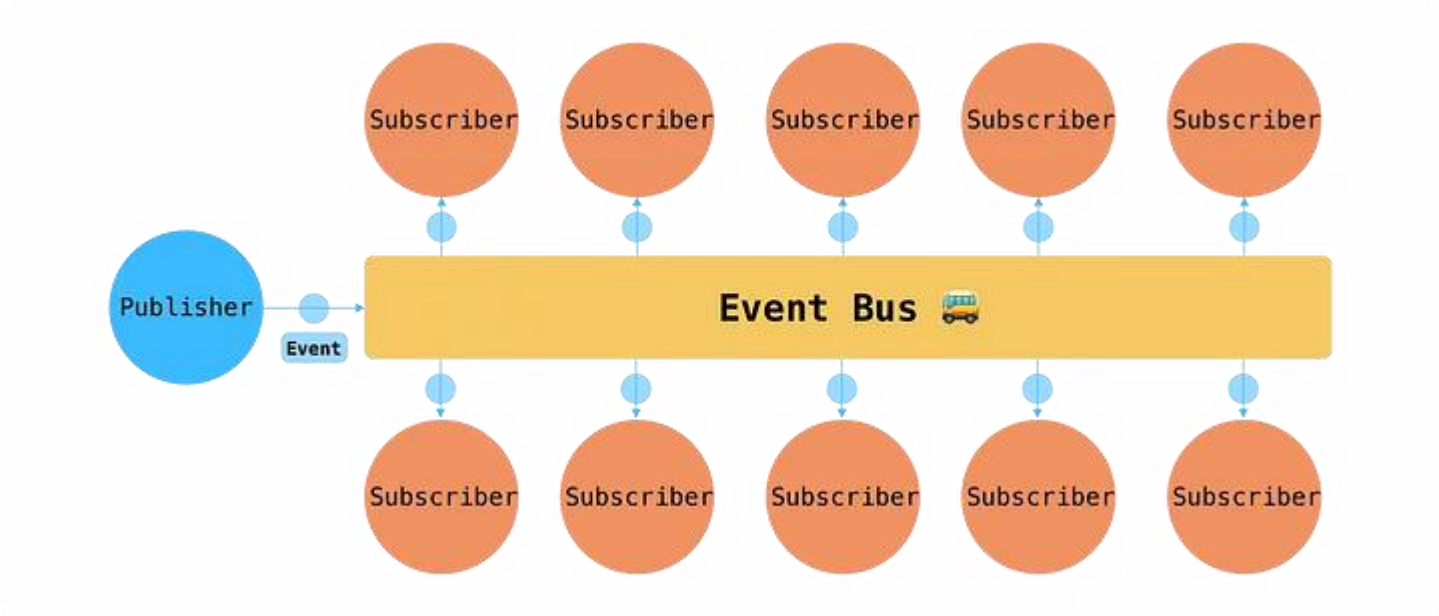


Рисунок 4. Event Bus pattern

2.5 Jest.js

Jest.js – фреймворк тестування застосунків, написаних на JavaScript з акцентом на простоті. Має багато допоміжних бібліотек, що дозволяє його застосовувати для тестування таких фронтенд фреймворків та бібліотек як Angular, Vue, React.

Тести написанні на Jest вимагають мінімуму попередньої конфігурації. Jest має багатий API, в тому числі для того щоб створювати об’єкти-імітації для будь якого модуля за межами видимості тесту та для відслідковування викликів сторонніх функцій. Також Jest виконує тести паралельно і реорганізовує порядок виконання тестів на основі того, як довго вони виконуються. [6]

Взагалі в цій роботі буде реалізоване модульне тестування React застосунку, для чого Jest підходить якнайкраще.

Модульне тестування (Unit testing) — це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми. Модулем називають найменшу частину програми, яка може бути протестованою. В даному випадку це окремі компоненти та допоміжні функції. Модульне тестування потрібно, щоб упевнитися, що код відповідає вимогам

архітектури, має очікувану поведінку та що зміна реалізації функції чи компоненту не змінила очікувану поведінку. Також модульні тести можуть слугувати певної документацією коду, бо відображають його очікувану поведінку та оновлюються зі зміною тестованих модулів (в той час як звичайна письмова документація може повільно реагувати на зміни коду).[7]

2.6 React Testing Library

React Testing Library – достатньо проста і невелика бібліотека для тестування React компонентів. Може працювати з будь-яким фреймворком тестування, проте рекомендується використовувати Jest.

Основний принцип тестування, на якому побудована ця бібліотека – «Чим більше ваші тести схожі на те, як використовується ваша програма, тим більшу впевненість вони вам дають». Тому при тестуванні використовуються реальні елементи DOM, а не компоненти React. Для цього в тесті спочатку виконується метод render, якому передається тестований React компонент. Потім, за допомогою допоміжних методів getByText, getByTestId, getByPlaceholderText та багатьох інших, можна перевірити чи існують відповідні елементи в DOM, чи містять вони потрібний текст або потрібний CSS клас. Також можна симулювати події, зроблені користувач, наприклад, натискання на кнопку чи зміну поля вводу тексту за допомогою методів fireEvent.[8]

Розділ 3. Реалізація додатку

3.1 Технічне завдання

До початку проходження тесту користувачу надається можливість встановити параметри тесту :

- Кількість питань (від 5 до 20). 20 – максимум, бо це максимальна кількість випадкових питань підтримувана Quiz API. (У інших платформ максимум 50).
- API питань – одне з трьох варіантів: Trivia API, Quiz API, Open Trivia DB
- (Необов'язковий) складність – один з трьох варіантів: Проста, Середня, Складна. Якщо не вказано питання можуть бути будь-якої складності.
- (Необов'язковий) категорія – в залежності від API надається можливість додати категорію питань. Якщо не вказано питання можуть бути будь-якої категорії.

Після старту квізу користувачу буде показано перше питання тесту із варіантами відповідей. Після натискання на варіант користувач побачить правильний варіант та зможе перейти на наступне питання. Якщо користувач відповів правильно, то йому зараховуються бали за питання (від одного до трьох в залежності від складності). Також користувач може побачити максимальну кількість балів за квіз і набрані на даний момент бали. Кожне питання має обмежений час на відповідь. Після завершення часу користувач втрачає можливість отримати бали за питання. Якщо питання має декілька правильних варіантів відповідей, то користувач матиме можливість вибрати декілька відповідей. Такі питання також коштують в два рази більше балів, проте для їх отримання користувач має правильно вказати всі варіанти відповіді.

Після завершення тесту користувач може побачити список всіх питань тесту, загальну оцінку за тест. Також користувач зможе перейти до будь-якого питання

і знову побачити варіанти відповідей на нього а також свою оцінку за це питання.

В будь-який момент проходження тесту користувач може його перезапустити, втративши свій прогрес. Також він має можливість переключення між двома варіантами відображення інтерфейсу – світлий і темний.

3.2 Можливі покращення

- Додавання авторизації користувачів, можливість перегляду найкращих результатів інших користувачів.
- Додавання більшої кількості API питань.
- Можливість створення власних питань.

3.3 Розробка додатку

3.3.1 Початкові налаштування

Для створення проекту було використано create-react-app. Create React App – зручний інструмент для швидкого створення початкового React застосунку. Цей інструмент створює проект з вже налаштованими інструментами розробки. Команда для створення проекту: *npx create-react-app name*.

Для встановлення React-Bootstrap використовувалися наступні команди – *npm install bootstrap*, *npm install react-bootstrap*. Також у кореневому компоненті застосунку (App.js) потрібно імпортувати стилі Bootstrap для правильної роботи React-Bootstrap (*import 'bootstrap/dist/css/bootstrap.min.css'*). Для встановлення React-Bus – відповідно *npm install react-bus*.

Для запуску застосунку використовується команда *react-scripts start.*

Для більш швидкого та комфортного відлагодження застосунку можна використовувати інструмент React Developer Tools – розширення під Google Chrome, яке дозволяє бачити структуру компонентів на сторінці, а також значення змінних у цих компонентах (Рисунок 5).

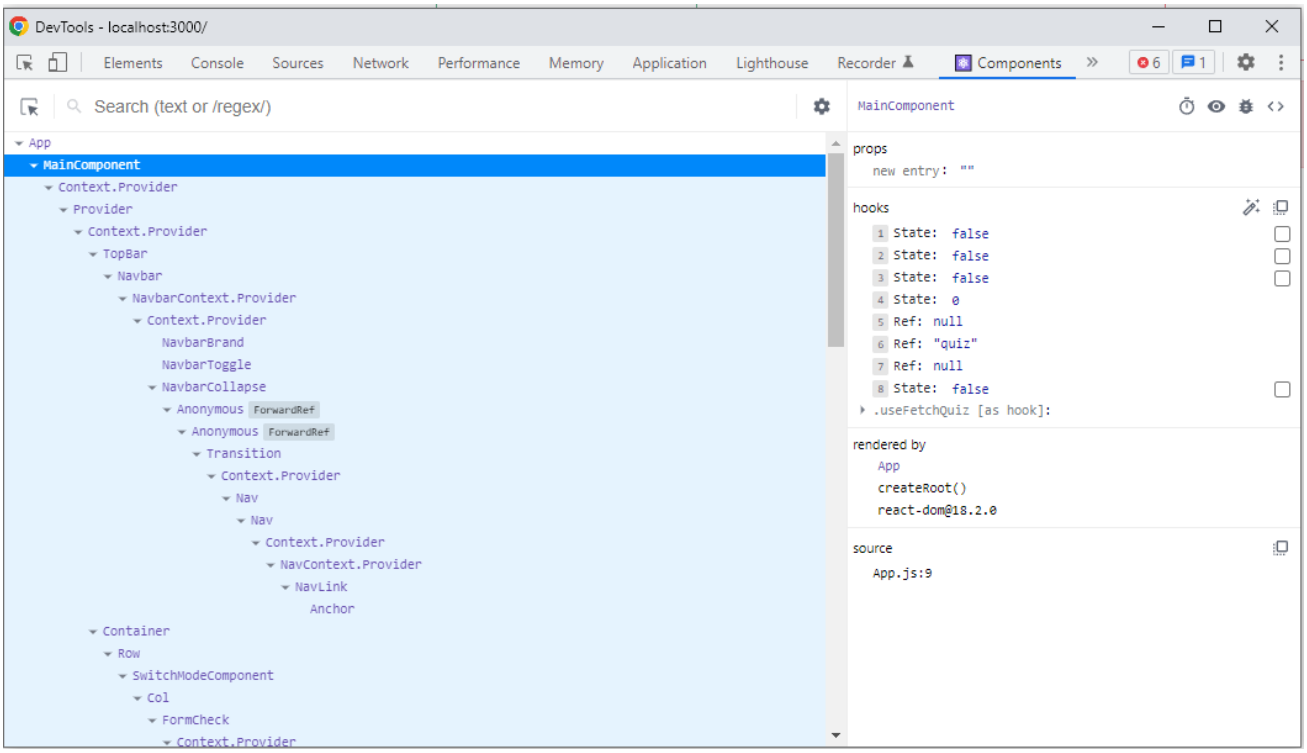


Рисунок 5. React Developer Tools

3.3.2 Інтерфейс початку тесту

Для початку тесту потрібно встановити з якого API отримувати питання, а також кількість питань. Також можна встановити необов’язкові параметри складності і категорії тесту. Після цього потрібно натиснути кнопку початку тесту (Рисунок 6).

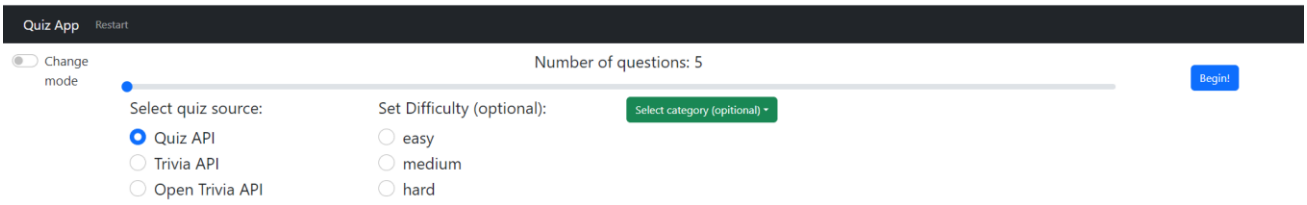


Рисунок 6. Інтерфейс початку тесту

Цей інтерфейс реалізовано у вигляді React компоненту InitialPage.js (Рисунок 7).

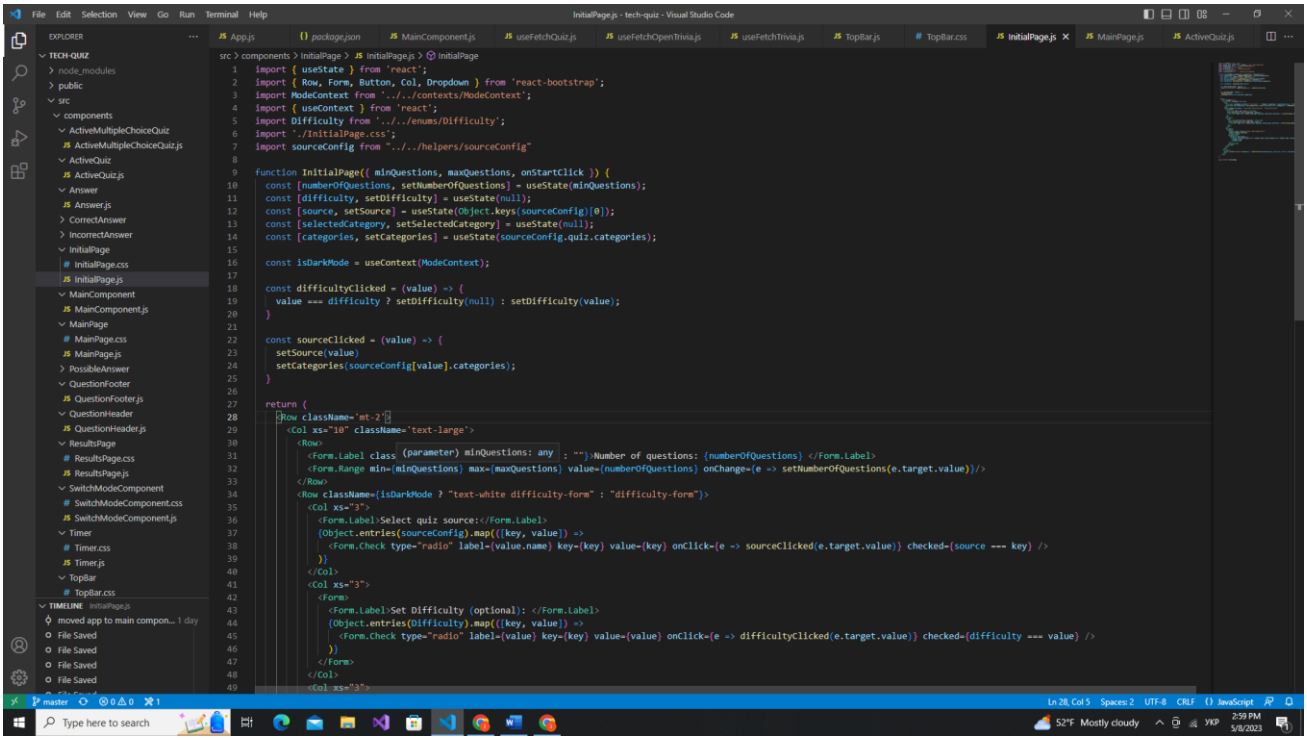


Рисунок 7. InitialPage.js

Як можна побачити, для зберігання наших параметрів тесту використовується хук useState. Для кожного параметру тесту ми зберігаємо значення у відповідній змінній, а для зміни цього значення використовуємо функції зміни – setState().

Так, наприклад, для зміни значення складності використовуємо Form.Check onClick props, який виконуємо передану в нього функцію при натисканні на відповідний елемент інтерфейсу. Для виводу всіх значень складності або категорії прямо в розмітці JSX використовується JavaScript метод масиву map, який повертає відповідний JSX елемент для кожного елементу масиву.

Після виставлення всіх параметрів і натискання на кнопку “Begin” викликається функція, передана в props цього компонента з вибраними параметрами – onStartClick. Також можна зазначити, що так як props передається у вигляді об’єкту, то можна використати деструктор об’єктів JavaScript і працювати відразу зі значеннями властивостей об’єкта.

3.3.3 Контекст кольору інтерфейсу

Як можна побачити, з лівої сторони екрану користувач може змінити колір відображення інтерфейсу. За замовчуванням інтерфейс світлий, проте можна змінити колір на темний (Рисунок 8).

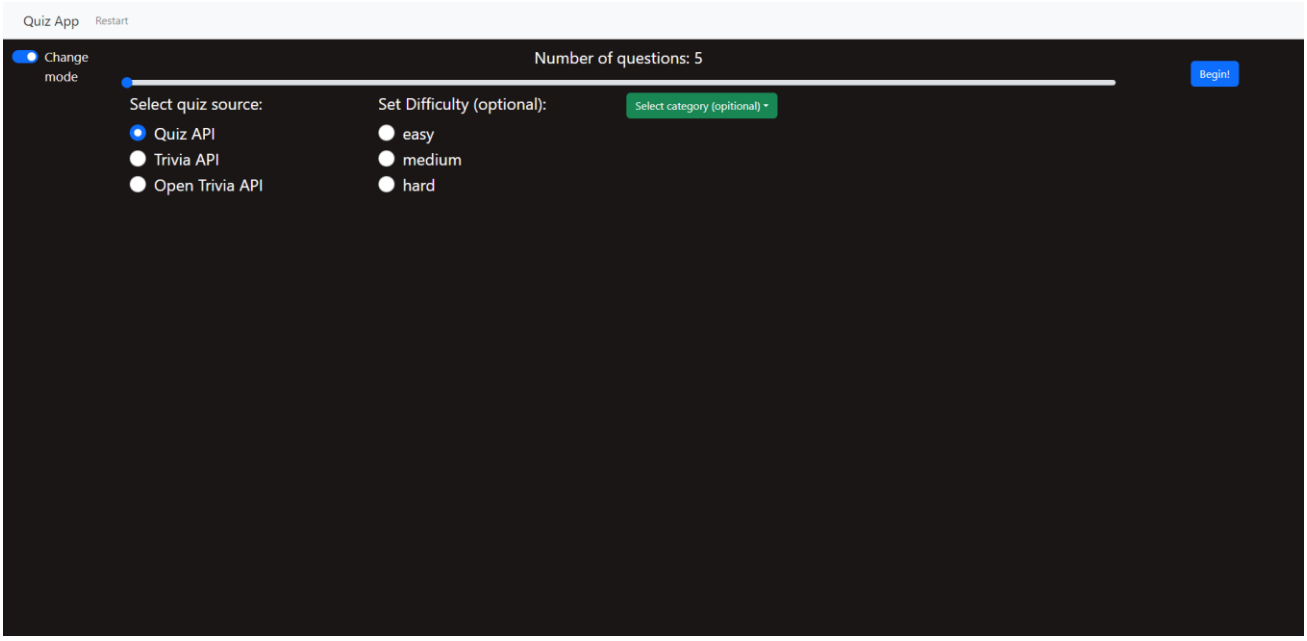


Рисунок 8. Темний режим інтерфейсу

Так як значення який режим інтерфейсу зараз використовується може знадобитися при відображенні будь-якого компоненту, то це значення краще не передавати через props. Для цього можна використати React контекст. Для цього в окремому файлі за допомогою React.createContext створюємо ModeContext (Рисунок 9).

```
1  import React from "react";
2  const ModeContext = React.createContext();
3  export default ModeContext;
```

Рисунок 9. ModeContext

Потім імпортуємо його в найвищий в ієрархії компонент і створюємо провайдр контексту:

<ModeContext.Provider value={isDarkMode}>… </ModeContext.Provider>

Також в цьому компоненті за допомогою useState зберігаємо значення контексту, і змінюємо в потрібний момент.

Тепер у будь-якому іншому компоненті можна використати хук useContext і отримати значення кольору інтерфейсу :

const isDarkMode = useContext(ModeContext);

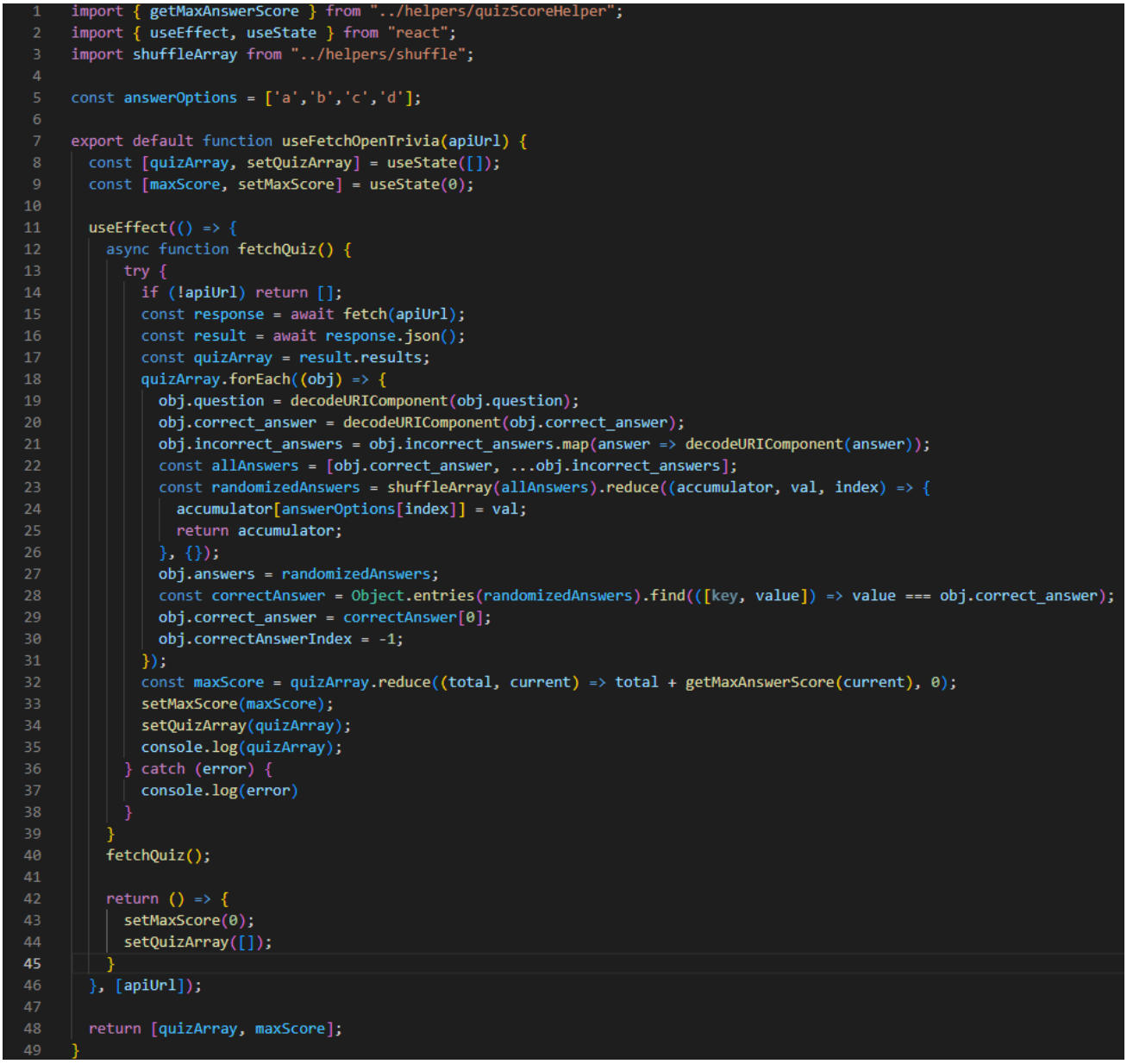
3.3.4 Отримання питань тестів

За параметрами для тесту потрібно отримати питання тесту (Рисунок 10).

```
26  let quizURL = `${currentSource.url}`;
27  quizURL += `${currentSource.limitParam}=${numberOfQuestions}`;
28  if (difficulty.current) {
29    quizURL += `&${currentSource.difficultyParam}=${difficulty.current}`;
30  }
31  if (category.current) {
32    quizURL += `&${currentSource.categoryParam}=${category.current}`;
33  }
34
35  quizURL = startQuiz ? quizURL : "";
36
37  const [quiz, maxScore] = currentSource.hook(quizURL);
38
```

Рисунок 10. Створення Url запиту

Для цього за параметрами формуємо URL. У окремому об’єкті sourceConfig зберігаються дані про ендпоінти даних тестів – який URL кожного з них, які категорії доступні, які назви URL параметрів для додавання складності і категорії, а також який хук використати для отримання даних. Для отримання даних з різних API використовуються різні хуки, бо потрібно перетворити дані питань в один формат. Так, наприклад, для Open Trivia API використовуємо хук useFetchOpenTrivia.js (Рисунок 11).



```
import { getMaxAnswerScore } from "../helpers/quizScoreHelper";
import { useEffect, useState } from "react";
import shuffleArray from "../helpers/shuffle";

const answerOptions = ['a','b','c','d'];

export default function useFetchOpenTrivia(apiUrl) {
  const [quizArray, setQuizArray] = useState([]);
  const [maxScore, setMaxScore] = useState(0);

  useEffect(() => {
    async function fetchQuiz() {
      try {
        if (!apiUrl) return [];
        const response = await fetch(apiUrl);
        const result = await response.json();
        const quizArray = result.results;
        quizArray.forEach((obj) => {
          obj.question = decodeURIComponent(obj.question);
          obj.correct_answer = decodeURIComponent(obj.correct_answer);
          obj.incorrect_answers = obj.incorrect_answers.map(answer => decodeURIComponent(answer));
          const allAnswers = [obj.correct_answer, ...obj.incorrect_answers];
          const randomizedAnswers = shuffleArray(allAnswers).reduce((accumulator, val, index) => {
            accumulator[answerOptions[index]] = val;
            return accumulator;
          }, {});
          obj.answers = randomizedAnswers;
          const correctAnswer = Object.entries(randomizedAnswers).find(([key, value]) => value === obj.correct_answer);
          obj.correct_answer = correctAnswer[0];
          obj.correctAnswerIndex = -1;
        });
        const maxScore = quizArray.reduce((total, current) => total + getMaxAnswerScore(current), 0);
        setMaxScore(maxScore);
        setQuizArray(quizArray);
        console.log(quizArray);
      } catch (error) {
        console.log(error)
      }
    }
    fetchQuiz();

    return () => {
      setMaxScore(0);
      setQuizArray([]);
    }
  }, [apiUrl]);

  return [quizArray, maxScore];
}
```

Рисунок 11. useFetchOpenTrivia.js

Для отримання даних використовуємо useEffect, в якому за допомогою функції fetch і отриманого URL отримуємо питання тесту, декодуємо їх з URI формату (бо Open Trivia повертає текст зашифрований таким форматом), для кожного питання створюємо масив зі всіма відповідями і рандомізуємо порядок елементів у ньому. З хуку повертаємо два значення – список питань та максимально можливий бал за весь тест, які зберігаються у відповідних змінних state. Аналогічним чином працюють два інших хуки.

3.3.5 Інтерфейс питання тесту

Для відображення конкретного питання тесту використовуємо компоненти MainPage.js, ActiveQuiz.js (Рисунок 12) та ActiveMultipleChoiceQuiz.js (Рисунок 13).

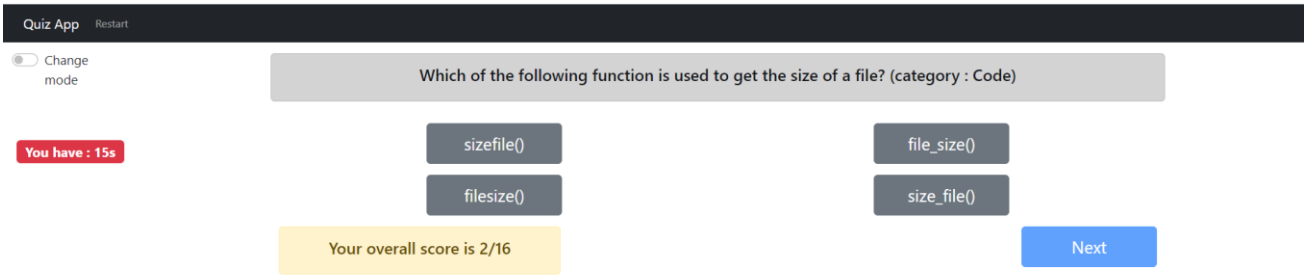


Рисунок 12. ActiveQuiz.js

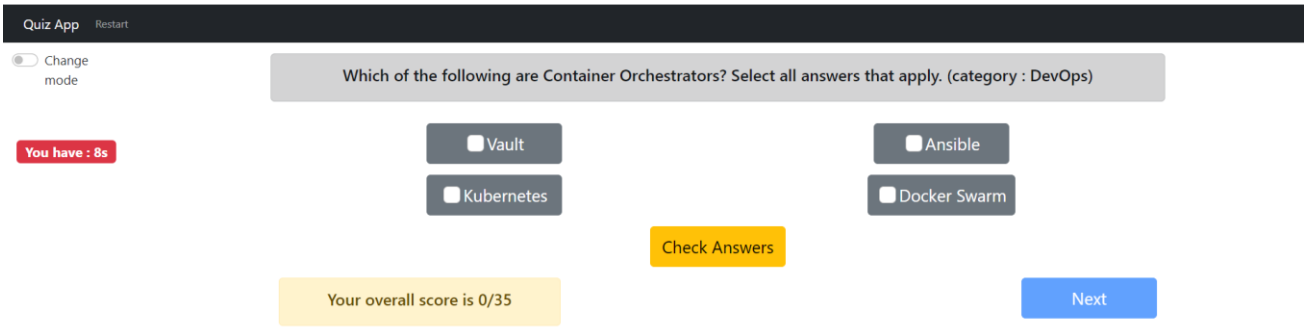


Рисунок 13. ActiveMultipleChoiceQuiz.js

MainPage контролює яке з питань потрібно відобразити, чи відображати ActiveQuiz.js або ActiveMultipleChoiceQuiz.js в залежності від типу питання, а також таймер, загальну набрану кількість балів з квіз і кнопку переходу на наступне питання. ActiveQuiz відображає питання, можливі відповіді (від одного до трьох рядків відповідей по дві кнопки на кожному), а також інтерфейс коли відповідь була дана, у якому показується правильна відповідь і кількість

отриманих балів за це питання (Рисунок 14).

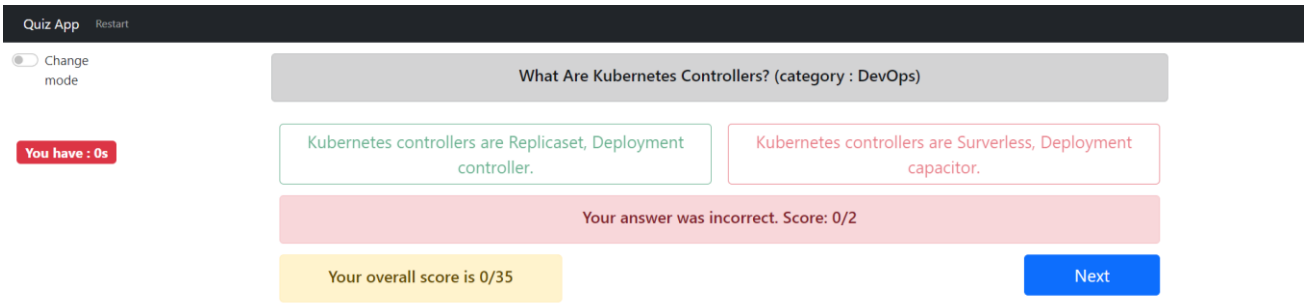


Рисунок 14, ActiveQuiz коли відповідь дана

ActiveMultipleChoiceQuiz відображає аналогічну інформацію, дозволяючи вказувати декілька правильних відповідей. Однакові елементи інтерфейсу – текст питання, три стани кнопок (з можливою відповіддю, правильна відповідь, не правильна відповідь) винесені в окремі компоненти і використовуються в обох варіантах питань.

3.3.6 Таймер відповіді

Для кожного питання є обмеження на час відповіді. Банер з лівої частини екрану відображає користувачу час який залишився для відповіді. Для цього використовується компонент Timer.js (Рисунок 15). При створенні компонента кількість секунд виставляється на початкове значення в залежності від складності питання. Також застосовується useEffect, у якому створюється інтервал з функцією оновлення значення таймеру кожну секунду. Для зберігання значення intervalId використовується useRef, а не useState, бо при зміні цього значення не потрібно оновлювати інтерфейс. Якщо користувач відповів на питання, таймер потрібно зупинити. Для цього в props компоненту передається значення timerStopped, яке використовується як залежність для useEffect. Таким чином при зміні цього значення функція передана у useEffect знову буде виконана і інтервал навпаки буде скасовано.

```
import { useEffect, useRef, useState } from "react";
import { Badge } from "react-bootstrap";
import getTimeInSecondsByDifficulty from "../../helpers/quizTimerHelper";
import './Timer.css';
import { useBus } from "react-bus";
import EventBus from "../../enums/EventBus";

function Timer({ difficulty, timerStopped }) {
  const initialTime = getTimeInSecondsByDifficulty(difficulty);
  const [timeLeft, setTimeLeft] = useState(initialTime);
  const interval = useRef(null);
  const bus = useBus();

  useEffect(() => {
    if (timerStopped && interval.current) {
      clearInterval(interval.current)
      interval.current = null
    } else if (!timerStopped && !interval.current) {
      interval.current = setInterval(() => {
        setTimeLeft(prevTime => {
          prevTime--;
          if (prevTime <= 0) bus.emit(EventBus.TIMER_END, true);
          return prevTime
        });
      }, 1000);
    }
  }, [timerStopped, timeLeft])
  return (
    <Badge className="timer-badge" bg="danger">
      You have : {timeLeft}s
    </Badge>
  )
}

export default Timer;
```

Рисунок 15. Timer.js

Якщо таймер добігає кінця, то використовуючи react-bus надсилається повідомлення, що таймер добіг кінця до того як користувач відповів.

Для реагування на це повідомлення в компоненті ActiveQuiz використовується хук useListener:

useListener(EventBus.TIMER_END, () => onPossibleAnswerClick(-1));

3.3.7 Сторінка результатів тесту

При закінченні тесту відображається інформація про набрані бали, а також список всіх питань з інформацією чи на це питання було дано правильну відповідь. При кліку на текст питання на сторінці результатів знову

відкривається відповідне питання з варіантами відповідей (без можливості редагувати відповідь) (Рисунок 16).

Рисунок 16. Інтерфейс результатів тесту

3.4 Модульне тестування додатку

3.4.1 Запуск тестів

При створенні додатку за допомогою інструменту create react app Jest встановлюється автоматично. Для встановлення React Testing Library необхідно ввести команду:

npm install --save-dev @testing-library/react

Для запуску тестів використовується команда *npm test.* Jest автоматично запускає всі тести, потім виводить у консоль інформацію про їх успішність (Рисунок 17).

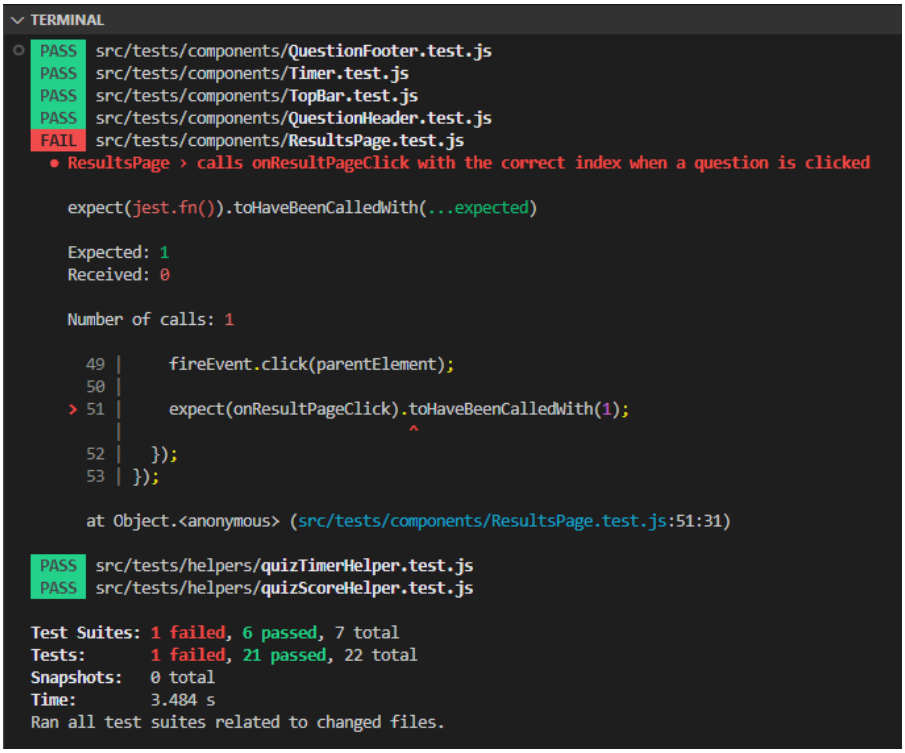


Рисунок 17. Приклад «npm test» із одним провальним тестом

3.4.2 Просте тестування допоміжної функції

Для прикладу простих тестів візьмемо тести для допоміжної функції *getMaxAnswerScore*, що знаходиться в файлі quizScoreHelper.js. Ця функція приймає питання тесту як аргумент і повертає кількість балів за це питання при правильній відповіді користувача. Ця кількість балів залежить від складності питання та від того чи у нього є декілька правильних відповідей.

Для тестування створюємо два тест кейси – один для звичайного питання і один для питання з декількома правильними відповідями. Кожен кейс позначається Jest методом *it* (або *test*), в який передаємо стрічку – опис тесту та функцію самого тесту. Так як ми маємо три варіанти для складності, то щоб не повторювати однаковий код для тестування кожної складності можна застосувати метод it.each, який приймає таблицю з даними для тест кейсу. В таблиці зазначаємо кожну можливу складність та відповідний очікуваний результат. В самому тест кейсі створюємо об’єкт питання, викликаємо функцію *getMaxAnswerScore* з відповідним об’єктом та за допомогою методу

expect(result).toEqual(expectedResult); перевіряємо правильність результату (Рисунок 18).

```
import Difficulty from "../../enums/Difficulty";
import { getMaxAnswerScore } from "../../helpers/quizScoreHelper";

describe('getMaxAnswerScore', () => {
  it.each([[Difficulty.EASY, 1], [Difficulty.MEDIUM, 2], [Difficulty.HARD, 3]])
    ('for difficulty %s should return score %i', (difficulty, expectedResult) => {
      const question = { difficulty };

      const result = getMaxAnswerScore(question);

      expect(result).toEqual(expectedResult);
    })

  it.each([[Difficulty.EASY, 2], [Difficulty.MEDIUM, 4], [Difficulty.HARD, 6]])
    ('for difficulty %s should return score %i if question is multiple choice', (difficulty, expectedResult) => {
      const question = { difficulty, multiple_correct_answers: "true" };

      const result = getMaxAnswerScore(question);

      expect(result).toEqual(expectedResult);
    })
})
```

Рисунок 18.quizScoreHelper.test.js

3.4.3 Тестування React компоненту

Для прикладу тестування React компоненту можна навести тести до компоненту *Timer*, який відображає кількість секунд, що залишилася у користувача для відповіді на питання. Для цього в тесті використовуємо метод render, у який передаємо JSX розмітку зі створеним компонентом *Timer* та необхідними props (Рисунок 19).

```
10
11  describe("Timer", () => {
12    const expectedTimerValue = 10;
13    const difficulty = Difficulty.EASY;
14
15    beforeAll(() => {
16      jest.useFakeTimers();
17      jest.spyOn(global, 'setInterval');
18    })
19
20    beforeEach(() => {
21      getTimeInSecondsByDifficulty.mockReturnValue(expectedTimerValue);
22    })
23
24    it('renders timer with initial time from getTimeInSecondsByDifficulty', () => {
25      const { getByText } = render(<Timer difficulty={difficulty} timerStopped={true}></Timer>)
26
27      expect(getTimeInSecondsByDifficulty).toHaveBeenCalledWith(difficulty);
28      expect(getByText(`You have : ${expectedTimerValue}s`)).toBeInTheDocument();
29    })
30
31    it('sets interval to update time if timerStopped is false', () => {
32      const updateTimeInMilliseconds = 1000;
33
34      render(<Timer difficulty={difficulty} timerStopped={false}></Timer>);
35
36      expect(setInterval).toHaveBeenCalledWith(expect.anything(), updateTimeInMilliseconds);
37    })
38
39    it('updates timer value after 1 second', async () => {
```

Рисунок 19. Timer.test.js

Метод *render* повертає об’єкт з функціями для отримання інформації про елементи в DOM. За допомогою функції *getByText* перевіряємо чи існує елемент з правильною кількістю секунд, що залишилися для відповіді. Також цей компонент викликає допоміжну функцію *getTimeInSecondsByDifficulty*, що повертає початковий час на виконання тесту в залежності від складності питання. Так як повинен тестуватися тільки компонент *Timer* не можна спиратися на реалізацію цієї функції. Для імітації допоміжної функції використовуємо метод *jest.mock:*

```
jest.mock("../../helpers/quizTimerHelper", () => {
  return {
    __esModule: true,
    default: jest.fn(),
  }
```

});

Тепер всі експорти з модуля *quizTimerHelper* будуть імітовані і *getTimeInSecondsByDifficulty* повертатиме Jest функцію імітації. Для того щоб вказати, яке значення потрібно даній функції повернути, в самому тест кейсі до створення компоненту *Timer* використовуємо метод

getTimeInSecondsByDifficulty.mockReturnValue(expectedTimerValue);

Також після створення компоненту можна перевірити, що він дійсно викликав цю функцію з потрібними аргументами за допомогою виклику методу :

expect(getTimeInSecondsByDifficulty).toHaveBeenCalledWith(difficulty);

Висновок

Під час виконання курсової роботи було продемонстровано можливості бібліотеки React для розробки сучасних односторінкових веб-застосунків. За допомогою цієї технології було створено веб-додаток у вигляді онлайн-квізу за різними темами, з багатьма можливостями для кастомізації онлайн тесту, з обмеженнями по часу для кожного тесту, з підрахунком балів користувача та з можливістю перегляду помилок після проходження тесту. Для створення привабливого інтерфейсу було використано можливості бібліотеки Bootstrap. Також було застосовано дизайн паттерн Event Bus для взаємодії між незалежними компонентами. Іще було продемонстровано можливості фреймворку Jest для модульного тестування JavaScript коду, а також React Testing Library у поєднанні з Jest для автоматизованого тестування React компонентів.

Подальший розвиток додатку можливий у вигляді додавання користувачами власних питань, зберігання найкращих результатів тестів користувачів, а також додавання e2e тестів для всієї системи.

Список джерел

1. Документація бібліотеки React [Електронний ресурс]. – Режим доступу: https://react.dev/reference/react
2. Документація бібліотеки Bootstrap [Електронний ресурс]. – Режим доступу: https://getbootstrap.com/docs/5.3/getting-started/introduction/
3. Документація бібліотеки React Bootstrap [Електронний ресурс]. – Режим доступу: https://react-bootstrap.github.io/
4. How to Use Event Bus in React Architecture [Електронний ресурс]. – Режим доступу: https://betterprogramming.pub/how-to-use-event-bus-in-react-architecture-f90485477647
5. Документація бібліотеки React Bus [Електронний ресурс]. – Режим доступу: https://github.com/goto-bus-stop/react-bus
6. Документація фреймворку тестування Jest [Електронний ресурс]. – Режим доступу: https://jestjs.io/docs/getting-started
7. Модульне тестування програмного забезпечення [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Unit_testing
8. Документація бібліотеки React Testing Library [Електронний ресурс]. – Режим доступу: https://testing-library.com/docs/react-testing-library/intro
9. Офіційна сторінка платформи «Trivia API» [Електронний ресурс]. – Режим доступу: https://the-trivia-api.com/
10. Документація API «Open Trivia Database» [Електронний ресурс]. – Режим доступу: https://opentdb.com/api_config.php
11. Офіційна сторінка платформи «Quiz API» [Електронний ресурс]. – Режим доступу: https://quizapi.io/
12. Популярність фронтенд фреймворків [Електронний ресурс]. – Режим доступу: https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190