Marcel ŻAK

Quiz w React

Opis aplikacji

To jest interaktywna aplikacja quizowa stworzona w React, która pozwala użytkownikowi sprawdzić swoją wiedzę poprzez serię pytań wielokrotnego wyboru. Każde pytanie jest wyświetlane osobno i użytkownik ma 5 sekund na wybranie odpowiedzi, co jest wizualizowane za pomocą dynamicznego paska czasu. Po wybraniu odpowiedzi przyciski zmieniają kolor — zielony oznacza poprawną odpowiedź, a czerwony wskazuje błędną. Jeśli użytkownik nie zdąży odpowiedzieć w wyznaczonym czasie, pytanie jest traktowane jako nieodpowiedziane i automatycznie przechodzi do kolejnego. Aplikacja przechodzi przez wszystkie pytania po kolei, a po zakończeniu quizu wyświetla podsumowanie wyników, pokazując liczbę poprawnych odpowiedzi oraz całkowitą liczbę pytań. Dzięki temu użytkownik może szybko ocenić swój poziom wiedzy i powtórzyć quiz, jeśli chce się poprawić.

Tworzenie aplikacji

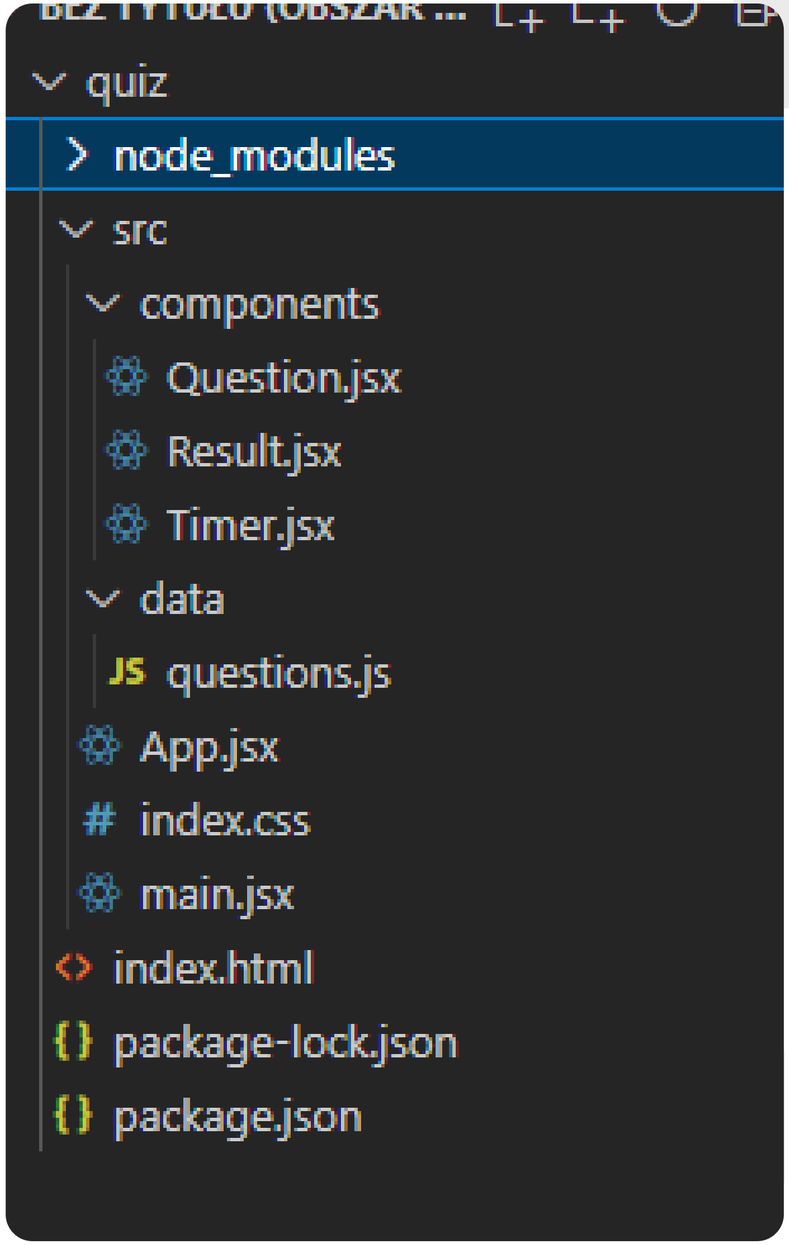
Tworzenie

```
$ npm create vite@latest quiz -- --template react
cd quiz
npm install
```

Start serwera

```
npm run dev
```

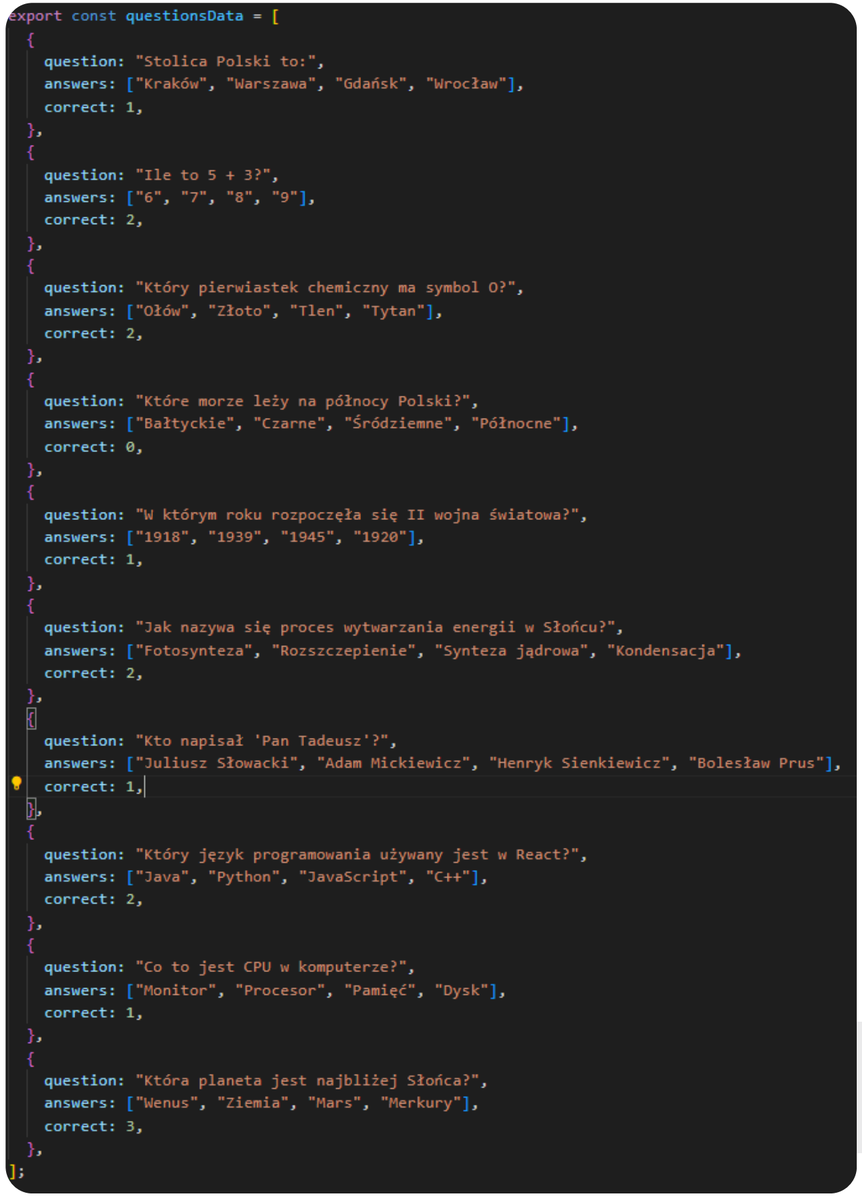
Struktura plików



Pytania

To jest kompleksowy zestaw pytań quizowych zapisanych jako tablica obiektów w JavaScript. Każdy obiekt zawiera treść pytania, listę możliwych odpowiedzi oraz indeks poprawnej odpowiedzi, co umożliwia łatwe i efektywne wykorzystanie ich w aplikacji quizowej. Pytania obejmują różnorodne dziedziny wiedzy — od geografii, matematyki, chemii, przez historię i literaturę, aż po informatykę — dzięki czemu quiz jest wszechstronny i angażujący. Struktura danych jest przejrzysta i łatwa do rozbudowy, co sprzyja dalszemu rozwojowi aplikacji.

02



```
export const questionsData = [
  {
    question: "Stolica Polski to:",
    answers: ["Kraków", "Warszawa", "Gdańsk", "Wrocław"],
    correct: 1,
  },
  {
    question: "Ile to 5 + 3?",
    answers: ["6", "7", "8", "9"],
    correct: 2,
  },
  {
    question: "Który pierwiastek chemiczny ma symbol O?",
    answers: ["Ołów", "Złoto", "Tlen", "Tytan"],
    correct: 2,
  },
  {
    question: "Które morze leży na północy Polski?",
    answers: ["Bałtyckie", "Czarne", "Śródziemne", "Północne"],
    correct: 0,
  },
  {
    question: "W którym roku rozpoczęła się II wojna światowa?",
    answers: ["1918", "1939", "1945", "1920"],
    correct: 1,
  },
  {
    question: "Jak nazywa się proces wytwarzania energii w Słońcu?",
    answers: ["Fotosynteza", "Rozszczepienie", "Synteza jądrowa", "Kondensacja"],
    correct: 2,
  },
  {
    question: "Kto napisał 'Pan Tadeusz'?",
    answers: ["Juliusz Słowacki", "Adam Mickiewicz", "Henryk Sienkiewicz", "Bolesław Prus"],
    correct: 1,
  },
  {
    question: "Który język programowania używany jest w React?",
    answers: ["Java", "Python", "JavaScript", "C++"],
    correct: 2,
  },
  {
    question: "Co to jest CPU w komputerze?",
    answers: ["Monitor", "Procesor", "Pamięć", "Dysk"],
    correct: 1,
  },
  {
    question: "Która planeta jest najbliżej Słońca?",
    answers: ["Wenus", "Ziemia", "Mars", "Merkury"],
    correct: 3,
  },
];
```

Marcel ŻAK

Logika pytań

Question.jsx to komponent odpowiedzialny za wyświetlanie konkretnego pytania oraz możliwych odpowiedzi. Użytkownik może kliknąć jedną z opcji, a komponent zaznaczy ją jako wybraną i na chwilę pokaże, czy odpowiedź była poprawna (zielona) czy błędna (czerwona). Następnie po dwóch sekundach informuje główny komponent (App) o wyniku tej odpowiedzi. Dodatkowo, jeśli użytkownik nie udzieli odpowiedzi w ciągu pięciu sekund, automatycznie uruchamia się funkcja, która traktuje pytanie jako błędnie udzielone i pokazuje poprawną odpowiedź. Dzięki temu komponent potrafi działać zarówno interaktywnie (kliknięcie), jak i reagować na upływ czasu.

```jsx
import React, { useState, useEffect } from 'react'; //Import React
import Timer from './Timer'; //Import Timera


export default function Question({ question, onAnswer }) {

  if (!question) return <div>ładowanie pytania...</div>;// Jesli brak pytania, wyswietlamy komunikat o ladowaniu
  const [selected, setSelected] = useState(null); //indeks odpowiedzi użytkownika
  const [answered, setAnswered] = useState(false); //zmienna sprawdzajaca czy jest odpowiedz na pytanie

  useEffect(() => {
    setSelected(null);
    setAnswered(false);
  }, [question]);

  // Funkcja wywoływana po kliknięciu odpowiedzi
  const handleAnswer = (index) => {
    if (answered) return; // Jesli uzytkownik juz odpowiedzial, ignorujemy klikniecie
    setSelected(index);//indeks wybranej odpowiedzi
    setAnswered(true);//odpowiedziano

    // Po 2 sekundach wywolujemy funkcje onAnswer z informacja, czy odpowiedz byla poprawna
    setTimeout(() => {
      onAnswer(index === question.correct);
    }, 2000);
  };

  // Funkcja wywolywana, gdy czas minie i uzytkownik nie odpowie
  const handleTimeUp = () => {
    if (!answered) {
      setAnswered(true);   // Zaznaczamy jako odpowiedziane
      setSelected(null);   // Brak wybranej odpowiedzi

      // Po 2 sekundach pokazuje,ze nie bylo odpowiedzi
      setTimeout(() => {
        onAnswer(false);
      }, 2000);
    }
  };
```

```jsx
  return (
    <>
      {/*jesli brak odpowiedzi wyswietla timer */}
      {!answered && <Timer duration={5} onTimeUp={handleTimeUp} />}

      {/* div z pytaniem i propsem question */}
      <div className="question">{question.question}</div>

      {/* Wyswietlenie listy odpowiedzi  */}
      <div className="answers">
        {question.answers.map((ans, idx) => {
          const isCorrect = idx === question.correct;//czy poprawna odpowiedz
          const isSelected = idx === selected;//czy poprawna jest wybrana
          //przypisanie CSS w zaleznosci od odpowiedzi
          let className = '';
          if (answered) {
            if (isCorrect) className = 'correct';
            else if (isSelected && !isCorrect) className = 'incorrect';
          }


          return (
            <button
              key={idx}// unikalny klucz
              onClick={() => handleAnswer(idx)}// kliknięcie = wybor odpowiedzi
              className={className}// CSS
              disabled={answered}// blokujemy kliknięcia po odpowiedzi
              type="button"//ustawienie typu button
            >
              {ans}                    {/* Tekst odpowiedzi */}
            </button>
          );
        })}
      </div>
    </>
  );
```

Marcel ŻAK

Question.jsx

Ten fragment kodu definiuje komponent funkcyjny React o nazwie Question, odpowiedzialny za renderowanie pojedynczego pytania w quizie oraz zarządzanie interakcją użytkownika z możliwymi odpowiedziami.

- Wykorzystuje hooki useState oraz useEffect do zarządzania stanem lokalnym i efektami ubocznymi.
- Importuje komponent Timer, który wizualizuje odliczanie czasu przeznaczonego na udzielenie odpowiedzi.
- Komponent przyjmuje dwa propsy:
  - question — obiekt reprezentujący bieżące pytanie wraz z odpowiedziami,
  - onAnswer — callback wywoływany po udzieleniu odpowiedzi, służący do przekazania informacji o poprawności wyboru.
- Implementuje warunek zabezpieczający przed renderowaniem komponentu bez danych pytania — w takim przypadku renderuje komunikat „Ładowanie pytania...”.
- Inicjuje dwa stany lokalne za pomocą useState:
  - selected — indeks aktualnie wybranej odpowiedzi lub null jeśli brak wyboru,
  - answered — flaga logiczna wskazująca, czy pytanie zostało już odpowiedziane, co umożliwia kontrolę dalszych interakcji użytkownika.

Ten fragment stanowi podstawowy szkielet mechanizmu zarządzającego procesem odpowiedzi na pytania oraz synchronizacją stanu komponentu z interfejsem użytkownika.

```jsx
import React, { useState, useEffect } from 'react'; //Import React
import Timer from './Timer'; //Import Timera


export default function Question({ question, onAnswer }) {

  if (!question) return <div>ładowanie pytania...</div>;// Jesli brak pytania, wyswietlamy komunikat o
  const [selected, setSelected] = useState(null); //indeks odpowiedzi użytkownika
  const [answered, setAnswered] = useState(false); //zmienna sprawdzajaca czy jest odpowiedz na pytani
```

Question.jsx

Ten fragment kodu zawiera logikę odpowiedzialną za resetowanie stanu pytania oraz obsługę odpowiedzi użytkownika i sytuacji, gdy czas na odpowiedź upłynął.
Funkcja useEffect jest wywoływana za każdym razem, gdy zmienia się obiekt question. Jej zadaniem jest zresetowanie stanu komponentu, ustawiając selected na null (brak wybranej odpowiedzi) oraz answered na false (pytanie nie zostało jeszcze odpowiedziane), co przygotowuje komponent do obsługi nowego pytania.
Funkcja handleAnswer obsługuje zdarzenie kliknięcia na jedną z odpowiedzi. Jeśli pytanie zostało już odpowiedziane (answered jest true), ignoruje kolejne kliknięcia. W przeciwnym razie ustawia wybraną odpowiedź (selected), oznacza pytanie jako odpowiedziane (answered = true) i po upływie 2 sekund wywołuje funkcję onAnswer, przekazując informację o tym, czy wybrana odpowiedź jest poprawna (porównując indeks wybranej odpowiedzi z indeksem poprawnej).
Funkcja handleTimeUp jest wywoływana, gdy timer odliczający czas na odpowiedź zakończy się, a użytkownik nie udzielił odpowiedzi. W takim przypadku ustawia stan answered na true i resetuje selected do null, a następnie po 2 sekundach wywołuje onAnswer z wartością false, sygnalizując, że odpowiedź nie została udzielona lub jest błędna.

```
useEffect(() => {
  setSelected(null);
  setAnswered(false);
}, [question]);

// Funkcja wywoływana po kliknięciu odpowiedzi
const handleAnswer = (index) => {
  if (answered) return; // Jesli uzytkownik juz odpowiedzial, ignorujemy klikniecie
  setSelected(index);//indeks wybranej odpowiedzi
  setAnswered(true);//odpowiedziano

  // Po 2 sekundach wywolujemy funkcje onAnswer z informacja, czy odpowiedz byla poprawna
  setTimeout(() => {
    onAnswer(index === question.correct);
  }, 2000);
};

// Funkcja wywolywana, gdy czas minie i uzytkownik nie odpowie
const handleTimeUp = () => {
  if (!answered) {
    setAnswered(true);   // Zaznaczamy jako odpowiedziane
    setSelected(null);   // Brak wybranej odpowiedzi

    // Po 2 sekundach pokazuje,ze nie bylo odpowiedzi
    setTimeout(() => {
      onAnswer(false);
    }, 2000);
  }
};
```

Question.jsx

Ten fragment kodu odpowiada za wizualne przedstawienie pojedynczego pytania w quizie, jego możliwych odpowiedzi oraz obsługę interakcji użytkownika. Jeśli użytkownik jeszcze nie odpowiedział, wyświetlany jest komponent Timer, który odlicza czas do końca pytania. Następnie wyświetlane jest samo pytanie oraz lista odpowiedzi w formie przycisków. Każdy przycisk reprezentuje jedną z możliwych odpowiedzi. Po zaznaczeniu odpowiedzi, przyciski zostają zablokowane, a odpowiedzi są wizualnie oznaczane kolorami – poprawne jako correct, a błędne jako incorrect. Odpowiednia klasa CSS jest przypisywana dynamicznie w zależności od poprawności i wyboru użytkownika. Dzięki temu użytkownik od razu widzi, która odpowiedź była prawidłowa.

```
return (
  <>
    {/*jesli brak odpowiedzi wyswietla timer */}
    {!answered && <Timer duration={5} onTimeUp={handleTimeUp} />}

    {/* div z pytaniem i propsem question */}
    <div className="question">{question.question}</div>

    {/* Wyswietlenie listy odpowiedzi  */}
    <div className="answers">
      {question.answers.map((ans, idx) => {
        const isCorrect = idx === question.correct;//czy poprawna odpowiedz
        const isSelected = idx === selected;//czy poprawna jest wybrana
       //przypisanie CSS w zaleznosci od odpowiedzi
        let className = '';
        if (answered) {
          if (isCorrect) className = 'correct';
          else if (isSelected && !isCorrect) className = 'incorrect';
        }


        return (
          <button
            key={idx}// unikalny klucz
            onClick={() => handleAnswer(idx)}// kliknięcie = wybor odpowiedzi
            className={className}// CSS
            disabled={answered}// blokujemy kliknięcia po odpowiedzi
            type="button"//ustawienie typu button
          >
            {ans}                    {/* Tekst odpowiedzi */}
          </button>
        );
      })}
    </div>
  </>
);
```

Marcel ŻAK

Result.jsx

Komponent Result to prosty komponent funkcyjny w React, odpowiedzialny za prezentację końcowego wyniku quizu. Importuje bibliotekę React, co jest niezbędne do poprawnego działania komponentów. Funkcja Result przyjmuje dwa argumenty (propsy): score, czyli liczbę poprawnych odpowiedzi użytkownika, oraz total, oznaczającą całkowitą liczbę pytań w quizie.
Wewnątrz komponentu znajduje się kontener , który został ostylowany za pomocą stylów inline. Stylizacja obejmuje wyśrodkowanie tekstu (textAlign: "center") oraz ustawienie marginesu górnego na 40 pikseli (marginTop: 40), co zapewnia estetyczny odstęp od górnej krawędzi. Wewnątrz tego kontenera umieszczono nagłówek <h2> z tekstem „Koniec quizu!”, informujący użytkownika o zakończeniu gry.
Bezpośrednio pod nagłówkiem znajduje się paragraf <p>, który dynamicznie wyświetla wynik gracza. W treści paragrafu wstawiane są wartości score oraz total, co daje komunikat w formacie „Twój wynik: X / Y”, gdzie X to liczba poprawnych odpowiedzi, a Y to łączna liczba pytań.
Komponent Result nie zawiera żadnej logiki interaktywnej ani stanu wewnętrznego – jego funkcją jest jedynie czytelne i estetyczne przedstawienie końcowego wyniku użytkownikowi po zakończonym quizie.

```jsx
import React from "react";


export default function Result({ score, total }) {
  return (
    <div
      style={{
        textAlign: "center", // wyśrodkowanie tekstu
        marginTop: 40        // odstęp od góry (w pikselach)
      }}
    >
      <h2>Koniec quizu!</h2> {/* Nagłówek informujący o zakończeniu */}
      <p>Twój wynik: {score} / {total}</p> {/* Wyświetlenie wyniku */}
    </div>
  );
}
```

Marcel ŻAK

Timer.jsx

Na stronie komponent Timer odpowiada za wyświetlanie paska postępu, który pokazuje upływający czas na odpowiedź w quizie. Pasek stopniowo się skraca, a po upływie określonego czasu (np. 5 sekund) automatycznie wywoływana jest funkcja informująca o przekroczeniu limitu czasu. Timer resetuje się przy każdym nowym pytaniu.

```jsx
import React, { useEffect, useState } from "react";

export default function Timer({ duration, onTimeUp }) {
  const [widthPercent, setWidthPercent] = useState(100);//ustawiamy timer na 100

  useEffect(() => {

    setWidthPercent(100);

    const start = Date.now();//czas rozpoczecia
    const interval = setInterval(() => {

      const elapsed = Date.now() - start;//ile czasu zostało
      const percent = Math.max(100 - (elapsed / (duration * 1000)) * 100, 0);// Obliczamy procent pozostalego czasu
      setWidthPercent(percent);//aktualizacja timera
      if (percent <= 0) {
        clearInterval(interval);//zatrzymanie czasu jesli sie skonczyl
        onTimeUp();//wywolanie przekazanej funkcji jako props
      }
    }, 50); // interwał co 50 ms

    return () => clearInterval(interval);// usuwamy interwal przy odmontowaniu komponentu
  }, [duration, onTimeUp]);

  return (
    <div className="timer">
      <div
        className="timer-progress"
        style={{ width: `${widthPercent}%` }} // szerokość oparta o stan
      />
    </div>
  );
}
```

Marcel ŻAK

Timer.jsx

Ten fragment kodu tworzy początek komponentu Timer w React, który służy do wizualizacji upływającego czasu w postaci paska postępu.
import React, { useEffect, useState } from "react";
– Importuje bibliotekę React oraz dwa hooki:
useEffect – do wykonania efektu ubocznego (odliczania czasu),
useState – do zarządzania stanem komponentu.
export default function Timer({ duration, onTimeUp }) {
– Definiuje komponent funkcyjny Timer, który przyjmuje dwa argumenty jako propsy:
duration – długość odliczania w sekundach,
onTimeUp – funkcja wywoływana po upływie czasu.
const [widthPercent, setWidthPercent] = useState(100);
– Tworzy zmienną stanu widthPercent, która odpowiada za szerokość paska czasu (w procentach). Początkowo ustawiona jest na 100%, czyli pełny pasek – odliczanie jeszcze się nie rozpoczęło.
setWidthPercent to funkcja aktualizująca tę wartość w czasie rzeczywistym.

```jsx
import React, { useEffect, useState } from "react";

export default function Timer({ duration, onTimeUp }) {
  const [widthPercent, setWidthPercent] = useState(100);//ustawiamy timer na 100
```

Timer.jsx

Ten fragment kodu stanowi główną logikę działania komponentu Timer – odpowiada za odliczanie czasu i animację paska postępu. Oto jego opis krok po kroku:

useEffect(() => { ... }, [duration, onTimeUp]);

To tzw. efekt uboczny, który uruchamia się za każdym razem, gdy komponent zostanie załadowany lub gdy zmienią się wartości duration lub onTimeUp. Jest to odpowiednie miejsce do zainicjowania timera.

setWidthPercent(100);

Na początku każdorazowego uruchomienia efektu (np. przy zmianie pytania), pasek postępu zostaje zresetowany do 100%, co oznacza pełen czas do wykorzystania.

const start = Date.now();

Zapisujemy dokładny czas rozpoczęcia odliczania w milisekundach.

const interval = setInterval(() => { ... }, 50);

Tworzymy interwał, który będzie wykonywany co 50 ms, by płynnie aktualizować szerokość paska postępu.

const elapsed = Date.now() - start;

Obliczamy ile czasu minęło od uruchomienia timera.

const percent = Math.max(100 - (elapsed / (duration * 1000)) * 100, 0);

Na podstawie upływu czasu (elapsed) wyliczamy, ile procent czasu pozostało.

- elapsed / (duration * 1000) – oblicza, jaki ułamek z całego czasu już minął,
- 100 - ... * 100 – przelicza ten ułamek na procenty pozostałego czasu,
- Math.max(..., 0) – zapobiega wartościom ujemnym (np. gdy interwał zadziała z opóźnieniem).

setWidthPercent(percent);

Aktualizujemy stan – zmieniamy szerokość paska postępu w interfejsie.

if (percent <= 0) { clearInterval(interval); onTimeUp(); }

Jeśli pasek dojdzie do zera (czas minął),

- zatrzymujemy interwał (clearInterval)
- i wywołujemy przekazaną funkcję onTimeUp(), która poinformuje aplikację, że czas się skończył.

return () => clearInterval(interval);

Zwracana funkcja czyszcząca interwał po odmontowaniu komponentu (np. przy zmianie pytania), aby uniknąć wycieków pamięci i konfliktów w aktualizacjach stanu.

```jsx
useEffect(() => {

  setWidthPercent(100);

  const start = Date.now();//czas rozpoczecia
  const interval = setInterval(() => {

    const elapsed = Date.now() - start;//ile czasu zostało
    const percent = Math.max(100 - (elapsed / (duration * 1000)) * 100, 0);// Obliczamy procent pozostalego czasu
    setWidthPercent(percent);//aktualizacja timera
    if (percent <= 0) {
      clearInterval(interval);//zatrzymanie czasu jesli sie skonczyl
      onTimeUp();//wywolanie przekazanej funkcji jako props
    }
  }, 50); // interwał co 50 ms

  return () => clearInterval(interval);// usuwamy interwal przy odmontowaniu komponentu
}, [duration, onTimeUp]);
```

Marcel ŻAK

Timer.jsx

To zewnętrzny kontener timera. Posiada klasę CSS timer, która może być użyta do ogólnego stylowania, np. wysokości, koloru tła, obramowania itd.
<div className="timer-progress" style={{ width: \${widthPercent}% ` }} />`
To wewnętrzny pasek postępu, którego szerokość (w procentach) jest dynamicznie ustawiana na podstawie stanu widthPercent.
Oznacza to, że z każdą aktualizacją (co 50 ms) ten pasek zmniejsza swoją szerokość, tworząc efekt upływającego czasu.
Klasa timer-progress może być stylowana np. kolorem, animacją lub zaokrągleniami.

```
return (
  <div className="timer">
    <div
      className="timer-progress"
      style={{ width: `${widthPercent}%` }} // szerokość oparta o stan
    />
  </div>
);
```

Marcel ŻAK

App.jsx

Ten kod definiuje główny komponent aplikacji quizowej w React. Zarządza on całym przebiegiem quizu: od wyświetlania pytań, przez ocenianie odpowiedzi użytkownika, aż po prezentację końcowego wyniku.
W skrócie:

- Przechowuje aktualny numer pytania (currentQuestionIndex), liczbę poprawnych odpowiedzi (score) i informację, czy zakończono quiz (showResult).
- Gdy użytkownik odpowie, sprawdza poprawność odpowiedzi i przechodzi do następnego pytania albo wyświetla wynik.
- W zależności od stanu pokazuje komponent Question (z pytaniem i odpowiedziami) lub Result (z końcowym wynikiem).

```jsx
import React, { useState } from 'react';

import Question from './components/Question';
import Result from './components/Result';
import { questionsData } from './data/questions';

export default function App() {
  const [currentQuestionIndex, setCurrentQuestionIndex] = useState(0);// indeks aktualnego pytania
  const [score, setScore] = useState(0);// poprawne odpowiedzi
  const [showResult, setShowResult] = useState(false);// sprawdzenie czy to ostatnie pytanie

  const handleAnswerClick = (isCorrect) => {
    if (isCorrect) setScore(score + 1);//poprawna odpowiedz score+=1

    const nextIndex = currentQuestionIndex + 1;//zwiekszamy indeks pytania

    if (nextIndex < questionsData.length) {//jesli indeks mniejszy niz dlugosc tablicy questionData
      setCurrentQuestionIndex(nextIndex);//nastepne pytanie
    } else {
      setShowResult(true);//pokaz wynik
    }
  };

  return (
    <div className="app-container">
      {showResult ? (//jesli koniec
        <Result score={score} total={questionsData.length} />//przekazanie propsu wyniku i ilosci pytan
      ) : (
        <Question
          question={questionsData[currentQuestionIndex]}//przekazanie propsu indeksu pytania
          onAnswer={handleAnswerClick}// przekazanie funkcji
        />
      )}
    </div>
  );
}
```

App.jsx

```jsx
import React, { useState } from 'react';

import Question from './components/Question';
import Result from './components/Result';
import { questionsData } from './data/questions';

export default function App() {
  const [currentQuestionIndex, setCurrentQuestionIndex] = useState(0);// indeks aktualnego pytania
  const [score, setScore] = useState(0);// poprawne odpowiedzi
  const [showResult, setShowResult] = useState(false);// sprawdzenie czy to ostatnie pytanie
```

import React, { useState } from 'react';
Importuje bibliotekę React oraz hook useState, który pozwala komponentowi funkcjonalnemu przechowywać i aktualizować dane w stanie (ang. state).
import Question from './components/Question';
import Result from './components/Result';
import { questionsData } from './data/questions';
Importuje trzy elementy potrzebne do działania aplikacji:

- Question – komponent odpowiedzialny za wyświetlanie pytania oraz możliwych odpowiedzi.
- Result – komponent, który pokazuje wynik końcowy po zakończeniu quizu.
- questionsData – tablica obiektów z pytaniami i odpowiedziami.

export default function App() {
Definiuje główny komponent aplikacji o nazwie App, który jest eksportowany jako domyślny i będzie używany do renderowania quizu.
const [currentQuestionIndex, setCurrentQuestionIndex] = useState(0);
Tworzy stan currentQuestionIndex, który przechowuje indeks aktualnie wyświetlanego pytania (na początku 0). Funkcja setCurrentQuestionIndex umożliwia zmianę tego indeksu.
const [score, setScore] = useState(0);
Tworzy stan score, który liczy liczbę poprawnych odpowiedzi użytkownika (na początku 0). Funkcja setScore aktualizuje wynik.
const [showResult, setShowResult] = useState(false);
Tworzy stan showResult, który kontroluje, czy quiz się zakończył. Jeśli wartość to true, aplikacja pokazuje wynik, w przeciwnym razie – kolejne pytanie. Początkowo ustawione na false.

Marcel ŻAK

App.jsx

const handleAnswerClick = (isCorrect) => {
Definicja funkcji handleAnswerClick, która przyjmuje jeden argument isCorrect — jest to wartość logiczna (true lub false), wskazująca, czy odpowiedź użytkownika była poprawna.
if (isCorrect) setScore(score + 1);
Jeśli odpowiedź jest poprawna (isCorrect === true), następuje inkrementacja stanu score o 1, czyli liczba poprawnych odpowiedzi użytkownika rośnie o jeden.
const nextIndex = currentQuestionIndex + 1;
Tworzymy zmienną nextIndex, która jest o 1 większa od aktualnego indeksu pytania, czyli wskazuje na kolejne pytanie w quizie.
if (nextIndex < questionsData.length) {
setCurrentQuestionIndex(nextIndex);
} else {
setShowResult(true);
}
Sprawdzamy, czy nextIndex jest mniejszy od długości tablicy questionsData (czyli czy są jeszcze pytania do wyświetlenia).

- Jeśli tak — aktualizujemy stan currentQuestionIndex, aby przejść do następnego pytania.
- Jeśli nie — oznacza to, że wszystkie pytania zostały już pokazane, więc ustawiamy stan showResult na true, co wyzwala wyświetlenie końcowego wyniku.

```
const handleAnswerClick = (isCorrect) => {
  if (isCorrect) setScore(score + 1);//poprawna odpowiedz score+=1

  const nextIndex = currentQuestionIndex + 1;//zwiekszamy indeks pytania

  if (nextIndex < questionsData.length) {//jesli indeks mniejszy niz dlugosc tablicy questionData
    setCurrentQuestionIndex(nextIndex);//nastepne pytanie
  } else {
    setShowResult(true);//pokaz wynik
  }
};
```

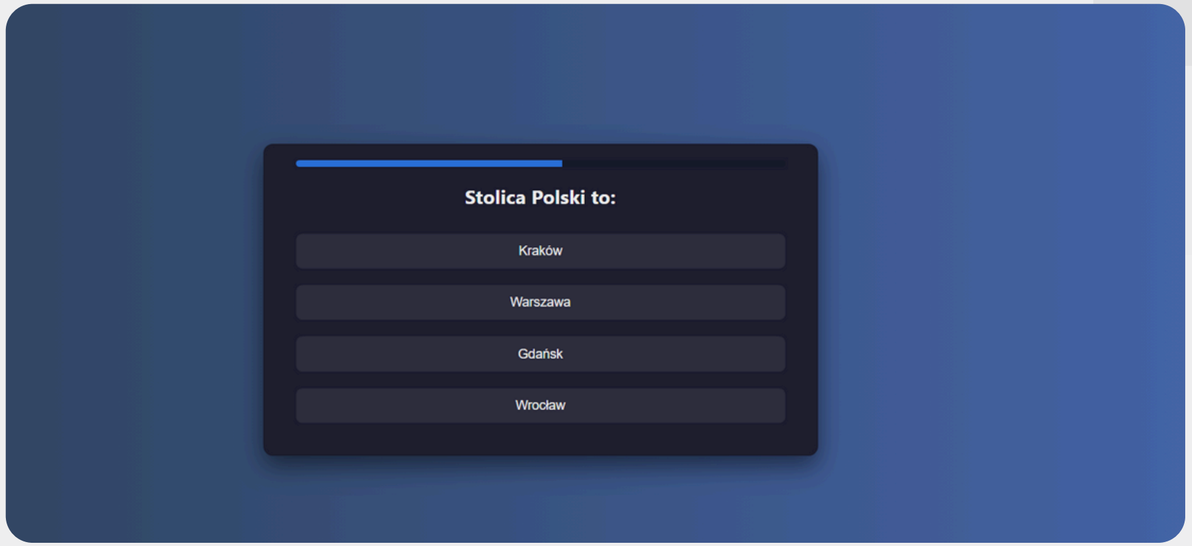
Marcel ŻAK

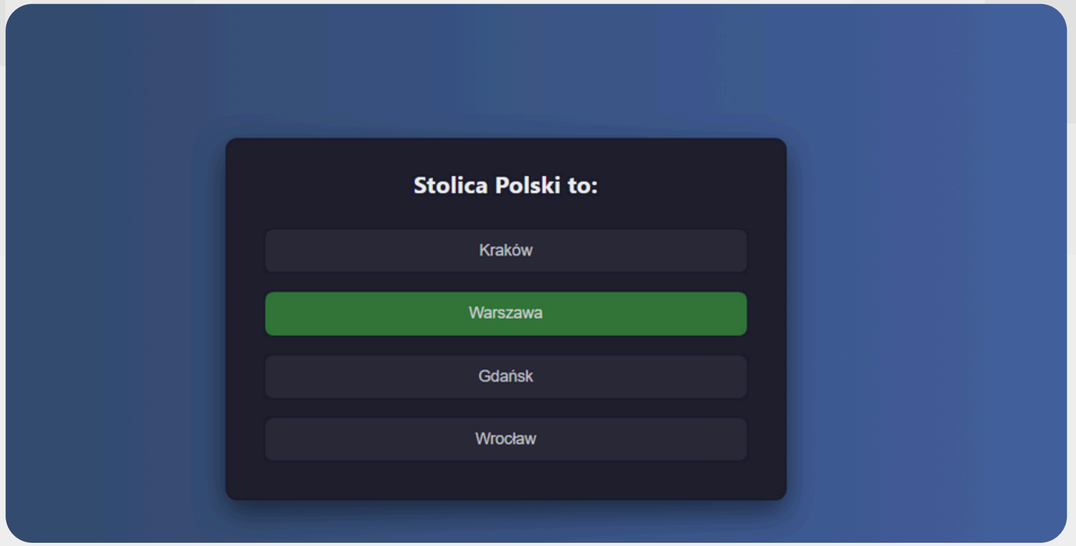
App.jsx

Ten fragment kodu odpowiada za wyświetlanie głównej zawartości aplikacji quizowej. Całość jest opakowana w element div z klasą CSS, która odpowiada za stylizację kontenera.
Logika opiera się na sprawdzeniu, czy quiz już się zakończył, czyli czy zmienna showResult jest ustawiona na true. Jeśli tak, aplikacja pokazuje komponent Result, który prezentuje użytkownikowi jego końcowy wynik – ile poprawnych odpowiedzi udało mu się udzielić na tle całkowitej liczby pytań.
Jeśli quiz nadal trwa, czyli showResult jest false, wtedy na ekranie pojawia się komponent Question. Ten komponent otrzymuje dane dotyczące aktualnego pytania, które ma wyświetlić, oraz funkcję, która obsługuje reakcję na odpowiedź użytkownika.
Dzięki temu mechanizmowi aplikacja dynamicznie przełącza się między wyświetlaniem pytań i podsumowaniem wyników w zależności od postępu użytkownika w quizie.

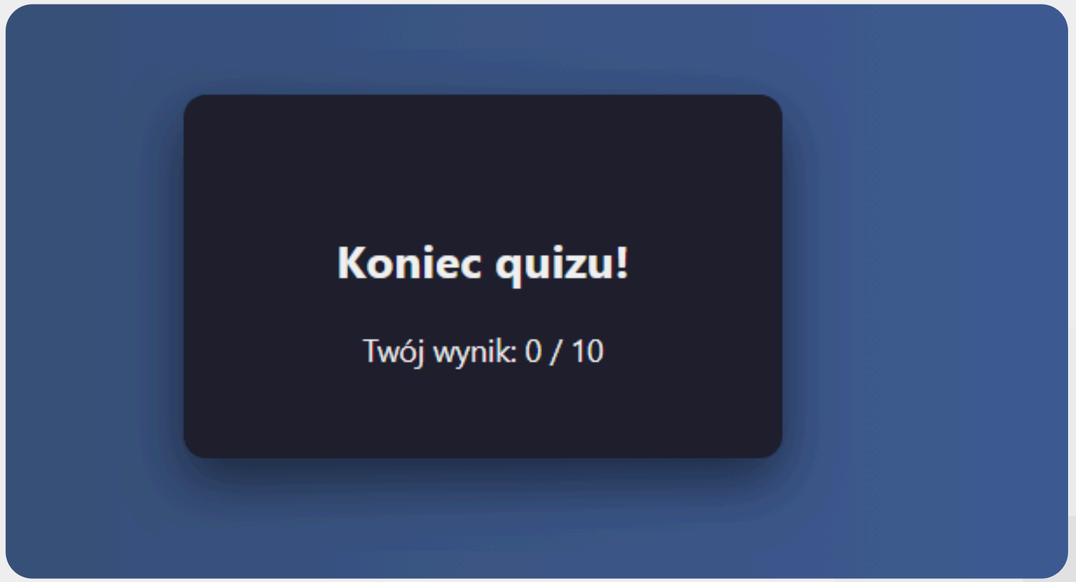
```jsx
return (
  <div className="app-container">
    {showResult ? (//jesli koniec
      <Result score={score} total={questionsData.length} />//przekazanie propsu wyniku i ilosci pytan
    ) : (
      <Question
        question={questionsData[currentQuestionIndex]}//przekazanie propsu indeksu pytania
        onAnswer={handleAnswerClick}// przekazanie funkcji
      />
    )}
  </div>
);
```

Marcel ŻAK


Stolica Polski to:
Kraków
Warszawa
Gdańsk
Wrocław


Stolica Polski to:
Kraków
Warszawa
Gdańsk
Wrocław


Stolica Polski to:
Kraków
Warszawa
Gdańsk
Wrocław


Koniec quizu!
Twój wynik: 0 / 10

Dziękuje