Kacper Kosuń 48955

WYBRANA TECHNOLOGIA HYBRYDOWA

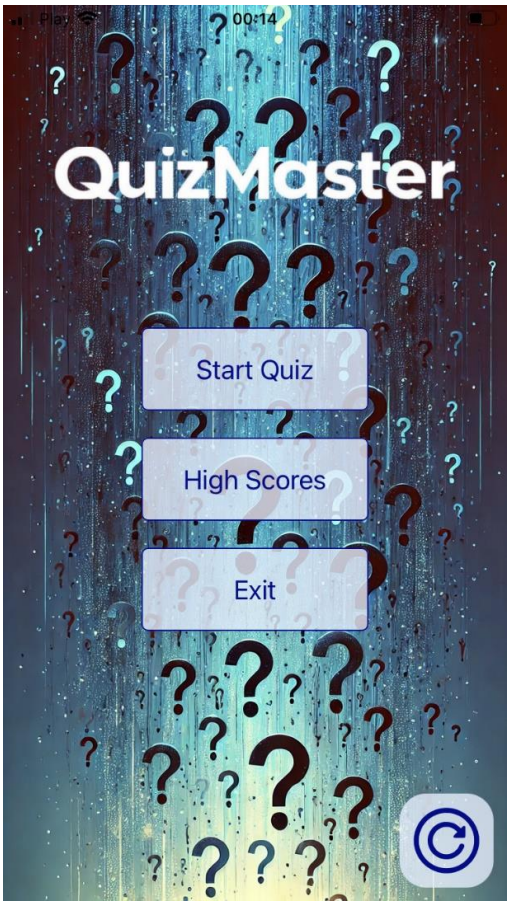
React Native to popularny framework opracowany przez Facebooka, który umożliwia tworzenie natywnych aplikacji mobilnych przy użyciu języka JavaScript i biblioteki React. Kluczową zaletą React Native jest to, że pozwala na rozwijanie aplikacji zarówno na platformę iOS, jak i Android, z użyciem jednej bazy kodu. Dzięki temu programiści mogą oszczędzać czas i zasoby, tworząc aplikacje działające na różnych systemach operacyjnych.

React Native wykorzystuje komponenty natywne, co oznacza, że aplikacje są szybkie, responsywne i wyglądają jak natywne aplikacje mobilne. Dodatkowo, umożliwia korzystanie z bogatego ekosystemu bibliotek React, co ułatwia implementację różnorodnych funkcji.

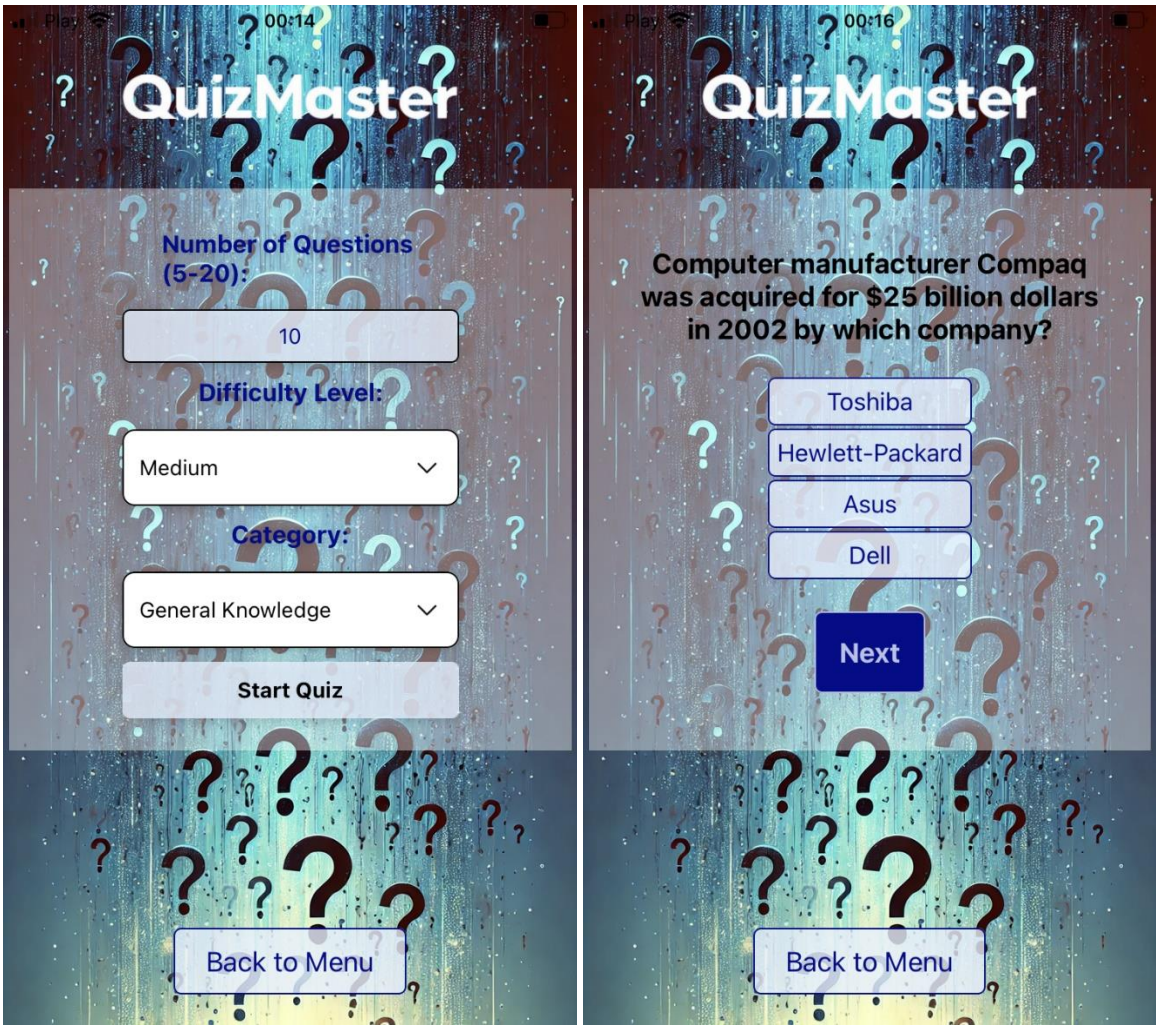
Repozytorium: https://github.com/Kacpay/ProgramowanieAplikacjiHybrydowych

MOJA APLIKACJA

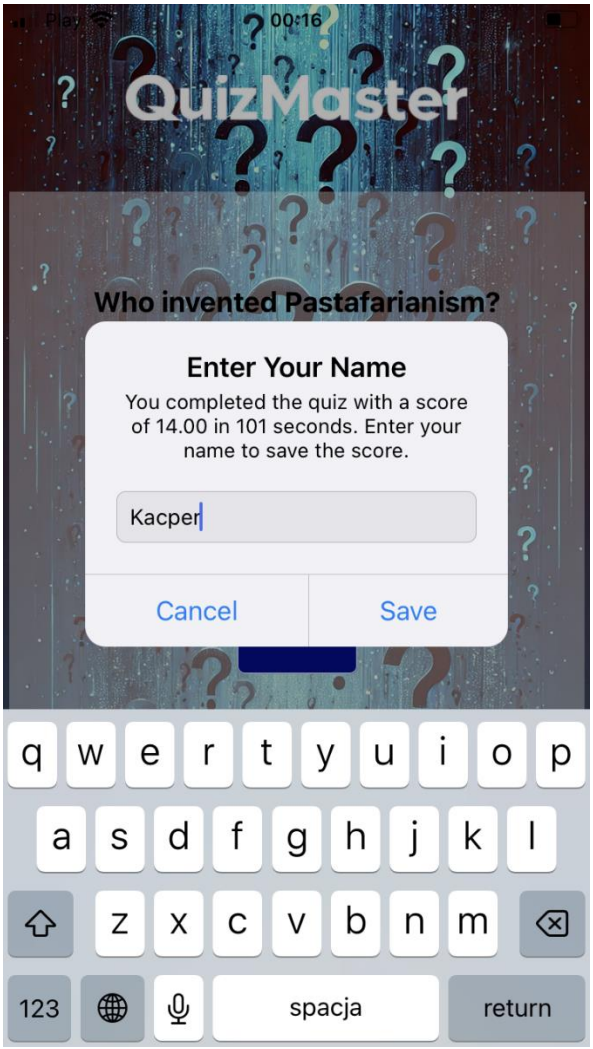
Menu główne

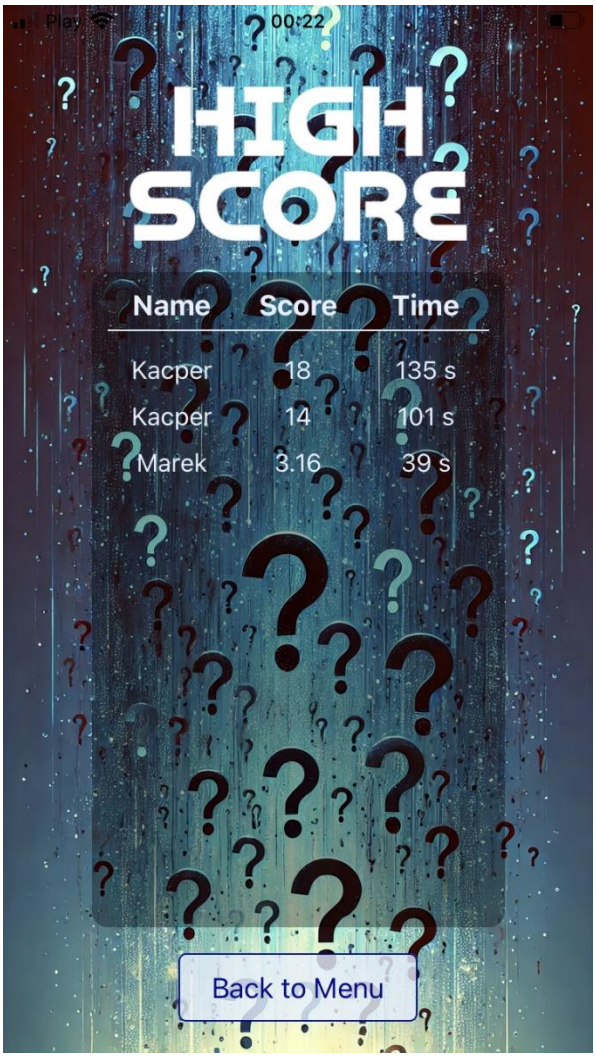


Komponenty ustawień/gry

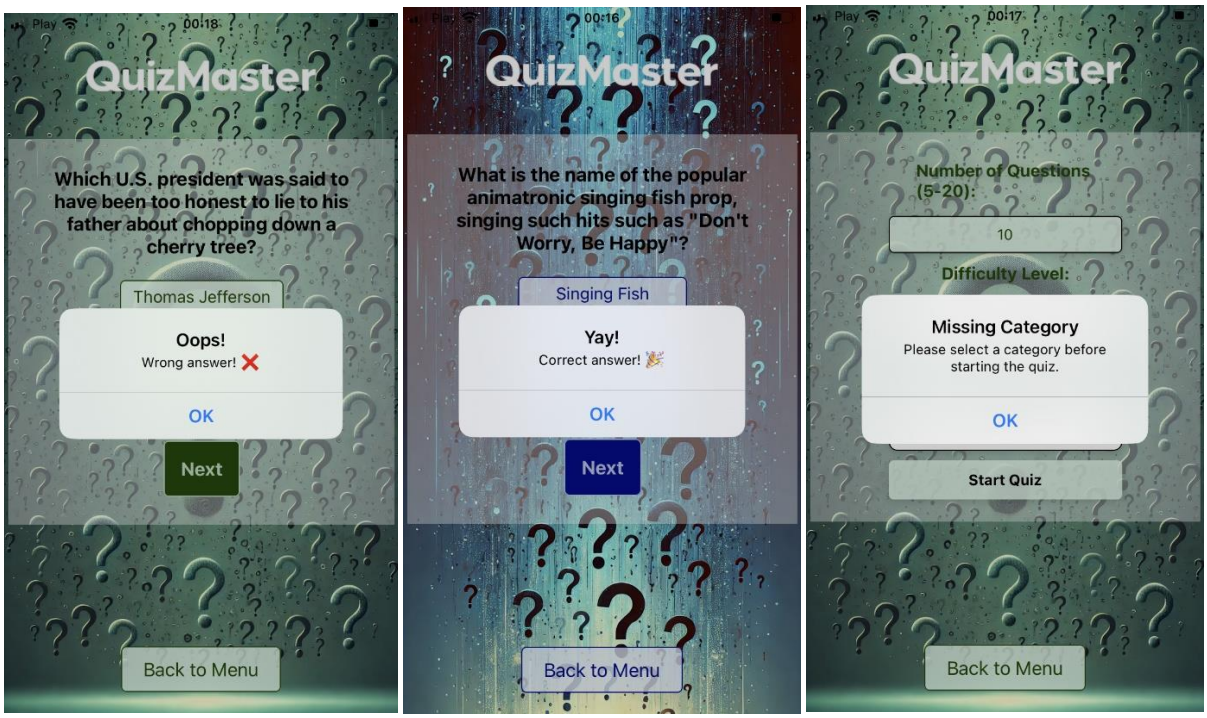


Zapis wyników





Powiadomienia/obsługa błędów



WZORCE PROJEKTOWE

W React Native nie ma jednego, powszechnie stosowanego wzorca projektowego, który byłby tak dominujący, jak BloC w Flutterze. Zamiast tego, w zależności od specyfiki projektu, preferencji zespołu programistycznego oraz skali aplikacji, stosuje się różne wzorce i praktyki, które najlepiej odpowiadają potrzebom danego przedsięwzięcia. W projektach React Native często stosuje się kombinację różnych wzorców projektowych i praktyk, aby sprostać specyficznym potrzebom aplikacji. W odróżnieniu od Fluttera, gdzie BloC może być dominującym wzorcem, w React Native podejście jest bardziej elastyczne.

W tym projekcie zostały użyte wzorce:

1. Wzorzec Komponentowy (Component-Based Architecture)

Gdzie?

- Każdy ekran (np. QuizSettings, HighScores) to osobny komponent.

- Stylizacja jest modularna (createStyles), co poprawia reużywalność kodu.

Zalety

- Ułatwia zarządzanie kodem.
- Pozwala na łatwe skalowanie aplikacji.
- Komponenty mogą być wielokrotnie używane w różnych miejscach.

2. Stan globalny – Context API (Wzorzec Provider)

Gdzie?

- ThemeContext → Przechowuje motyw (theme), a komponenty nasłuchują na jego zmiany.

Jak działa?

- useContext(ThemeContext) pozwala komponentom reagować na zmiany motywu.

Zalety

- Eliminuje konieczność przekazywania propsów przez wiele warstw komponentów.
- Efektywnie zarządza zmianami UI (np. ciemny/jasny motyw).

3. Wzorzec Fabryki (Dynamiczne tworzenie opcji kategorii)

Gdzie?

- useEffect w QuizSettings.jsx, który pobiera kategorie z API i mapuje je na { label, value }.

Jak działa?

- Tworzy dynamiczną listę kategorii quizu na podstawie danych z API.

Zalety

- Umożliwia łatwe rozszerzanie aplikacji o nowe kategorie bez zmiany kodu.

POŁĄCZENIE Z API

Jak aplikacja wykorzystuje połączenie z API?

Aplikacja korzysta z **Open Trivia Database API**, aby pobierać dane dotyczące kategorii oraz pytań quizowych. Połączenia z API są realizowane za pomocą żądań typu **GET**, a dane są dynamicznie przetwarzane i wykorzystywane w różnych częściach aplikacji.

Pobieranie kategorii quizu

Gdzie wykorzystywane?

Pobieranie kategorii quizu odbywa się w module ustawień quizu np. plik QuizSettings.jsx.

Jak działa?

- Aplikacja wysyła żądanie do API w celu pobrania dostępnych kategorii quizu.
- Po otrzymaniu odpowiedzi dane są konwertowane do formatu, który umożliwia ich wyświetlenie w rozwijanym menu wyboru kategorii.
- Pobraną listę kategorii aplikacja przechowuje w stanie (state), co umożliwia jej późniejsze wykorzystanie przez użytkownika.
- W przypadku błędu aplikacja go przechwytuje i uniemożliwia zawieszenie się programu.

Efekt

Użytkownik widzi listę dostępnych kategorii i może wybrać jedną z nich przed rozpoczęciem quizu.

Pobieranie pytań quizu

Gdzie wykorzystywane?

Pobieranie pytań następuje w ekranie quizu QuizGame.jsx po kliknięciu przycisku startu.

Jak działa?

- Aplikacja dynamicznie generuje adres URL na podstawie ustawień wybranych przez użytkownika (liczba pytań, poziom trudności, kategoria).

- Wysyłane jest żądanie do API, które zwraca losowe pytania w formacie JSON.
- Pobraną listę pytań aplikacja zapisuje w stanie i wykorzystuje do przeprowadzenia quizu.
- Jeśli API nie zwróci żadnych pytań, aplikacja informuje użytkownika i sugeruje zmianę ustawień quizu.

Efekt

Użytkownik otrzymuje losowy zestaw pytań, zgodny z wybranymi wcześniej ustawieniami.

Obsługa błędów połączenia z API

Aplikacja zapewnia obsługę błędów, aby uniknąć zawieszenia się programu w przypadku problemów z serwerem API. Jeśli wystąpi problem, np. brak dostępnych pytań dla wybranej kategorii, użytkownik otrzymuje odpowiedni komunikat zamiast pustego ekranu.

Przykładowe podejście do obsługi błędów obejmuje:

- Sprawdzenie, czy API zwróciło poprawne dane.
- Informowanie użytkownika o konieczności zmiany ustawień quizu, jeśli API nie zwróciło żadnych wyników.
- Obsługę błędów sieciowych i wyświetlanie odpowiednich komunikatów.

Efekt

Jeśli wystąpi problem z połączeniem lub API nie zwróci oczekiwanych wyników, aplikacja nie przestaje działać, lecz wyświetla użytkownikowi stosowny komunikat.