Sprawozdanie Frontend

Technologie użyte w projekcie

Frontend aplikacji RecipeHub został stworzony przy użyciu następujących technologii:

- **React**: Biblioteka JavaScript do budowy interfejsów użytkownika.
- **TypeScript**: Język programowania będący nadzbiorem JavaScript, zapewniający statyczne typowanie i lepszą kontrolę nad strukturą kodu.
- **Vite**: Narzędzie do budowy aplikacji, zapewniające szybki czas startu i wydajny proces budowania.
- **GraphQL**: Język zapytań do API, używany do komunikacji z backendem.
- **Apollo Client**: Biblioteka do obsługi GraphQL po stronie klienta, umożliwiająca zarządzanie stanem aplikacji i wykonywanie zapytań do API.
- **CSS Modules**: Mechanizm stylowania komponentów, zapewniający izolację stylów.
- **Chakra UI**: Biblioteka komponentów i design system, zapewniająca spójny wygląd aplikacji oraz łatwość w tworzeniu responsywnych interfejsów użytkownika.

Struktura projektu

Projekt frontendowy jest zorganizowany w modularny sposób, co ułatwia rozwój i utrzymanie. Główne elementy struktury to:

- **package.json**: Plik konfiguracyjny projektu, zawierający zależności i skrypty, np.:

```
"scripts": {
  "dev": "vite",
  "build": "vite build",
  "preview": "vite preview"
}
```

- **vite.config.ts**: Konfiguracja narzędzia Vite.
- **src/**: Główny katalog źródłowy, zawierający:
  - **App.tsx**: Główny komponent aplikacji.
  - **main.tsx**: Punkt wejściowy aplikacji, renderujący główny komponent.
  - **components/**: Katalog z komponentami wielokrotnego użytku, np.:
    - **Navbar.tsx**: Pasek nawigacyjny.
    - **Footer.tsx**: Stopka aplikacji.
  - **Views/**: Katalog z widokami aplikacji, np.:
    - **Home/**: Widok strony głównej.
    - **About/**: Widok strony "O nas".
    - **Contact/**: Widok strony kontaktowej.
  - **services/**: Katalog z logiką komunikacji z backendem, np.:
    - **graphql.tsx**: Konfiguracja Apollo Client i zapytania GraphQL.

Komunikacja z backendem

Komunikacja z backendem odbywa się za pomocą GraphQL i biblioteki Apollo Client. Przykładowa konfiguracja Apollo Client:

```
import { ApolloClient, InMemoryCache } from "@apollo/client";

const client = new ApolloClient({
  uri: "http://localhost:5000/graphql",
  cache: new InMemoryCache(),
});

export default client;
```

Zapytania GraphQL są definiowane w pliku `graphql.tsx` i wykonywane w komponentach za pomocą hooków, np.:

```
import { gql, useQuery } from "@apollo/client";

const GET_RECIPES = gql`
  query GetRecipes {
    recipes {
      id
      title
      description
    }
  }
`;

const Recipes = () => {
  const { loading, error, data } = useQuery(GET_RECIPES);

  if (loading) return <p>Loading...</p>;
  if (error) return <p>Error: {error.message}</p>;

  return (
    <ul>
      {data.recipes.map((recipe: any) => (
        <li key={recipe.id}>{recipe.title}</li>
      ))}
    </ul>
  );
};

export default Recipes;
```

Stylowanie

Stylowanie komponentów odbywa się za pomocą CSS Modules. Przykład użycia:

```
/* App.module.css */
.container {
  display: flex;
  justify-content: center;
  align-items: center;
}
```

```
import styles from "./App.module.css";

const App = () => {
  return <div className={styles.container}>Welcome to RecipeHub!</div>;
};

export default App;
```

Chakra UI

W projekcie RecipeHub wykorzystano bibliotekę Chakra UI jako design system. Chakra UI dostarcza gotowe komponenty, które są łatwe w użyciu, responsywne i dostosowane do różnych urządzeń. Dzięki temu możliwe jest szybkie tworzenie estetycznych i spójnych interfejsów użytkownika.

Przykład zastosowania Chakra UI w komponencie

Poniżej przedstawiono przykład użycia komponentów Chakra UI w komponencie `HeroHeader`:

```
import { Box, Heading, Text, Button } from "@chakra-ui/react";

const HeroHeader = () => {
  return (
    <Box
      bgGradient="linear(to-r, teal.500, green.500)"
      color="white"
      textAlign="center"
      py={10}
      px={5}
    >
      <Heading as="h1" size="2xl" mb={4}>
        Witaj w RecipeHub!
      </Heading>
      <Text fontSize="lg" mb={6}>
        Odkrywaj i dziel się swoimi ulubionymi przepisami kulinarnymi.
      </Text>
      <Button colorScheme="teal" size="lg">
        Rozpocznij teraz
      </Button>
    </Box>
  );
};
```

```
export default HeroHeader;
```

Wyjaśnienie kodu

- **Box**: Kontener z możliwością stylizacji, użyty do stworzenia tła gradientowego i wyrównania tekstu.
- **Heading**: Komponent do wyświetlania nagłówków, z wbudowanymi stylami.
- **Text**: Komponent do wyświetlania tekstu z możliwością dostosowania rozmiaru i odstępów.
- **Button**: Przycisk z predefiniowanymi stylami i obsługą różnych schematów kolorów.

Chakra UI znacząco przyspiesza proces tworzenia interfejsów użytkownika, zapewniając jednocześnie spójność wizualną w całej aplikacji.

Spójność projektu

Projekt jest spójny dzięki zastosowaniu:

- **TypeScript**: Zapewnia statyczne typowanie, co minimalizuje błędy i ułatwia refaktoryzację.
- **Apollo Client**: Ujednolica sposób komunikacji z backendem.
- **CSS Modules**: Zapewnia izolację stylów, co minimalizuje konflikty.
- **Modularna struktura katalogów**: Każdy moduł (np. komponenty, widoki) jest odseparowany, co ułatwia rozwój i utrzymanie projektu.

Dodatkowe punkty

- **Responsywność**: Aplikacja została zaprojektowana w sposób responsywny, aby działała poprawnie na różnych urządzeniach.
- **Wydajność**: Dzięki zastosowaniu Vite, aplikacja ładuje się szybko i działa wydajnie.
- **Łatwość rozbudowy**: Modularna struktura i użycie GraphQL umożliwiają łatwe dodawanie nowych funkcjonalności.
- **Testowanie**: Projekt zawiera podstawowe testy jednostkowe dla kluczowych komponentów.