POLITECHNIKA BYDGOSKA
Wydział Telekomunikacji,
Informatyki i Elektrotechniki

CHMURY OBLICZENIOWE
Informatyka Stosowana

Instrukcja 03 - Łączenie aplikacji frontend i backend

Kacper Krzyżniewski
Filip Pawłowski
Szymon Janiak

Cel laboratorium

Celem laboratorium jest połączenie aplikacji frontendowej z backendem przy użyciu zapytań `fetch` oraz zarządzanie stanem aplikacji za pomocą `slice` i `thunk` w Redux Toolkit. W trakcie laboratorium nauczymy się również wyświetlać dane z backendu na froncie, korzystając z `useSelector` w Redux.

Przykładowe rozwiązanie

Dodanie Slice i Thunk do Redux

W poprzednim laboratorium w ramach ćwiczenia zostało przedstawione przykładowe rozwiązanie, do którego możemy wykorzystać Redux. Teraz wykorzystamy go w bardziej praktyczny sposób.

1. W folderze `src/features` stwórz plik `scriptsSlice.js`:

```
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";

// Thunk do pobierania danych z backendu
export const fetchScripts = createAsyncThunk(
  "scripts/fetchScripts",
  async () => {
    const response = await fetch("http://localhost:4000/scripts");
    return await response.json();
  }
);

const scriptsSlice = createSlice({
  name: "scripts",
  initialState: {
    items: [],
    status: "idle", // idle | loading | succeeded | failed
    error: null,
  },
  reducers: {},
  extraReducers: (builder) => {
```

```
    builder
      .addCase(fetchScripts.pending, (state) => {
        state.status = "loading";
      })
      .addCase(fetchScripts.fulfilled, (state, action) => {
        state.status = "succeeded";
        state.items = action.payload;
      })
      .addCase(fetchScripts.rejected, (state, action) => {
        state.status = "failed";
        state.error = action.error.message;
      });
  },
});

export default scriptsSlice.reducer;
```

2. Zarejestruj scriptsSlice w pliku src/store/store.js:

```
import { configureStore } from "@reduxjs/toolkit";
import scriptsReducer from "../features/scriptsSlice";

export const store = configureStore({
  reducer: {
    scripts: scriptsReducer,
  },
});
```

Integracja z istniejącymi widokami

1. Aktualizacja głównego widoku:

W pliku src/pages/ScriptsPage.jsx zaimplementuj wyświetlanie listy skryptów za pomocą Redux Toolkit.

```
import React, { useEffect } from "react";
import { useSelector, useDispatch } from "react-redux";
import { fetchScripts } from "../features/scriptsSlice";

export default function MainView() {
  const dispatch = useDispatch();
  const scripts = useSelector((state) => state.scripts.items);
  const status = useSelector((state) => state.scripts.status);
  const error = useSelector((state) => state.scripts.error);

  useEffect(() => {
    if (status === "idle") {
      dispatch(fetchScripts());
    }
  }, [status, dispatch]);

  return (
    <div>
      <h2>Available Scripts</h2>
      {status === "loading" && <p>Loading...</p>}
```

```
      {status === "failed" && <p>Error: {error}</p>}
      <ul>
        {scripts.map((script) => (
          <li key={script.id}>
            <h3>{script.title}</h3>
            <p>{script.description}</p>
          </li>
        ))}
      </ul>
    </div>
  );
}
```

2. Aktualizacja szczegółowego widoku:

W pliku src/pages/ScriptPage.jsx zaimplementuj obsługę wyświetlania szczegółów wybranego skryptu.

```
import React from "react";
import { useSelector } from "react-redux";
import { useParams } from "react-router-dom";

export default function DetailView() {
  const { scriptId } = useParams();
  const script = useSelector((state) =>
    state.scripts.items.find((s) => s.id === parseInt(scriptId))
  );

  if (!script) {
    return <p>Script not found</p>;
  }

  return (
    <div>
      <h2>{script.title}</h2>
      <p>{script.description}</p>
```

```
    </div>
  );
}
```

3. Aktualizacja routingu:

Upewnij się, że odpowiednie ścieżki w `src/App.jsx` prowadzą do widoków zaktualizowanych w tym laboratorium.

```
import ScriptsPage from "./pages/ScriptsPage";
import ScriptPage from "./pages/ScriptPage";
import HomePage from "./pages/HomePage";

<Route path="/" element={<HomePage />} />
<Route path="/scripts/" element={<ScriptsPage />} />
<Route path="/script/:scriptId" element={<ScriptPage />} />
```

Zadanie do samodzielnego wykonania

Twoim zadaniem jest rozszerzenie funkcjonalności z Laboratorium nr 2 o obsługę zapytań stworzonych w Laboratorium nr 1. Należy dodać odpowiednie `slice` i `thunk`, a następnie zintegrować je z istniejącymi widokami aplikacji.

Wymagania

1. Stworzenie Slice i Thunk dla nowych danych: Utwórz `slice` oraz `thunk` w folderze `src/features`, odpowiadające zapytaniom z Laboratorium nr 1. Obsłuż różne stany: `loading`, `succeeded`, `failed`.

2. Podłączenie Slice do Store: Dodaj nowy `slice` do pliku `src/store/store.js`.

3. Wyświetlenie danych na frontendzie: Zintegruj nowe dane z widokiem głównym (`ScriptsPage.jsx`). Dodaj szczegółowe wyświetlanie danych w widoku szczegółowym (`ScriptPage.jsx`).

4. Obsługa błędów i stanów ładowania: - Zaimplementuj odpowiednie komunikaty dla stanów `loading` i `failed`.

Efekt końcowy

Po wykonaniu zadania aplikacja powinna: Obsługiwać dane z Laboratorium nr 1. Być w pełni funkcjonalna, wyświetlając dane w istniejących widokach. Umożliwiać użytkownikowi płynne przechodzenie między widokiem głównym a szczegółowym.