Programowanie Front-End

Użyteczne biblioteki

mgr inż. Jakub Gogola

React

I co dalej?

- Biblioteka React zapewnia nam zbiór podstawowych narzędzi do tworzenia *Single-Page Applications* w oparciu o technologię *Client-Side Rendering'u*.
- Przy pomocy samej jednak biblioteki React i Vanilla JS **trudnym** byłaby realizacja wszystkich funkcjonalności, które chcemy obsłużyć w aplikacji

React
I co dalej?

- Aplikacja webowa może zostać rozszerzona o następujące funkcjonalności:
  - routing,
  - utrzymywanie globalnego stanu aplikacji,
  - stałe przechowywanie danych po stronie klienta (przeglądarki),
  - uwierzytelnienie i autoryzacja,
  - obsługa zapytań do innych serwisów - REST API, WebSocket
  - nagrywanie i odtwarzanie *video* i *audio*,
  - streaming *video* i *audio*,
  - …

React

I co dalej?

- Wymienione na poprzednim slajdzie funkcjonalności możemy realizować za pomocą gotowych bibliotek lub narzędzi dostępnych w Web API przeglądarki. Oto kilka przykładów:
  - routing - **React Router**,
  - utrzymywanie globalnego stanu aplikacji - **Redux**,
  - stałe przechowywanie danych po stronie klienta (przeglądarki) - **Local Storage**,
  - obsługa zapytań do innych serwisów - REST API, WebSocket - **Axios**, **Fetch**, **Socket.IO**,
  - nagrywanie i odtwarzanie *video* i *audio* - **Media Recorder**

React Router

Server-Side Routing

- W standardowych aplikacjach implementujących *Server-Side Routing* każde zapytanie użytkownika o daną (pod)stronę jest wysyłane do serwera i przez niego procesowane.

React Routing

Server-Side Routing - zalety i wady

- Zaletami routingu *Server-Side Routing'u* są:
  - SEO-friendly,
  - szybkie ładowanie, ponieważ wszystkie *obliczenia* są realizowane po stronie serwera
- Wady natomiast to:
  - każde zapytanie o kolejną (pod)stronę wymaga całkowitego przeładowania strony, które pogarsza *user experience*
  - każde przeładowanie strony może wymagać ponownego pobrania danych, nawet jeżeli te same dane, w takiej samej, niezmienionej formie, zostały pobrane już za poprzednim razem

React Router

Client-Side Routing

- W przypadku *Client-Side Routing'u* funkcjonalność nawigowania pomiędzy poszczególnymi podstronami przerzucona jest na klienta, czyli przeglądarkę

React Router

Client-Side Routing

- Zaletami Client-Side Routing'u są:
  - szybsza nawigacja pomiędzy poszczególnymi podstronami bez konieczności przeładowywania całej aplikacji,
  - lepszy *user experience* poprzez lepsze zarządzanie zasobami (animacje, dane z API…)
- Wady to:
  - mniej *SEO-friendly*,
  - pierwsze wczytanie aplikacji może trwać dłużej ze względu na konieczność załadowania kodu JavaScript odpowiedzialnego za wyrenderowanie strony

React Router

Biblioteka

- Biblioteka `React Router` zapewnia wsparcie dla *Client-Side Routing'u*, symulując *Server-Side Routing*
- Została stworzona i jest kompatybilna z biblioteką React - https://reactrouter.com/en/main
- Zapewnia pełne wsparcie dla obsługi historii przeglądarki

React Router

Router

- Podstawowym komponentem wykorzystywanym przez bibliotekę jest `Router`
- `Router` to *kontener*, który pozwala na nawigację pomiędzy poszczególnymi podstronami aplikacji reprezentowanymi przez komponenty `Routes` oraz `Route`

React Router

BrowserRouter

- Podstawowym i najczęściej wykorzystywanym *router'em* jest `BrowserRouter`, który zapewnia pełne wsparcie dla historii przeglądarki korzystając z `History API`
- Dzięki niemu użytkownik może zobaczyć historię odwiedzanych w aplikacji podstron w historii swojej przeglądarki.

React router

Demo

- Link do kodu: https://github.com/JakubGogola-IDENTT/dsw-frontend-lecture-2024/tree/main/lecture-4/react

Redux

Co to jest?

- Biblioteka Redux pozwala na zarządzanie globalnym stanem aplikacji
- Stan może być używany globalnie i dzielony pomiędzy różnymi częściami aplikacji

Redux

store

- **Store** to *kontener* na globalny stan aplikacji pełniący rolę *single source of truth*
- *Store* jest prostym obiektem języka JavaScript
- Komponent mogą odczytywać i modyfikować stan aplikacji używając *store'a* przy zachowaniu spójności w obrębie całej aplikacji

Redux

action

- **Actions** to proste obiekty, które opisują zdarzenie (*event)*, które wprowadza zmianę w stanie
- Akcja składa się z dwóch części:
  - `type` - pole opisujące jaką zmianę powoduje dana akcja
  - `payload` - dane, które są przekazywane przez akcję i mogą zostać zapisane w stanie

Redux

reducer

- **Reducer** to czysta funkcja *(pure function),* która jako swoje argumenty przyjmuje bieżący stan oraz akcję a następnie wylicza i zwraca nowy stan

Redux

Dispatch

- *Dispatching* to proces wysyłania akcji do store’a.
- Kiedy akcja jest wysyłana, wówczas store odczytuje jej zawartość i stwierdza czy należy zmodyfikować stan.

Redux

Demo

- Link do kodu: https://github.com/JakubGogola-IDENTT/dsw-frontend-lecture-2024/tree/main/lecture-4/react

Local Storage

Co to jest?

- **Local Storage** to API webowe, które pozwala zapisywać oraz odczytywać dane w formacie *key-value* w pamięci przeglądarki
- Local Storage jest częścią WebStorage API - https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API

Local Storage

Przykłady zastosowania

- zapisywanie ustawień strony dla danego użytkownika
- zapisywanie danych, które nie muszą być przechowywane po stronie serwera
- cache
- zapisywanie danych tymczasowych takich jak ulubione przedmioty lub zawartość koszyka
- przechowywanie danych uwierzytelniających (tokeny)

Local Storage

Demo

- Link do kodu: https://github.com/JakubGogola-IDENTT/dsw-frontend-lecture-2024/tree/main/lecture-4/react

Fetch

Co to jest

- Fetch API (https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) zapewnia interfejs pozwalający na pobieranie zasobów oraz komunikację z innymi serwisami.

Fetch

Demo

- Link do kodu: https://github.com/JakubGogola-IDENTT/dsw-frontend-lecture-2024/tree/main/lecture-4/react

Dokumentacja

- Przykłady z wykładu: https://github.com/JakubGogola-IDENTT/dsw-frontend-lecture-2024/tree/main/lecture-4
- React Router - https://reactrouter.com/en/main
- Redux - https://redux.js.org/
- Web Storage API - https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API
- Fetch API - https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

Dziękuję za uwagę!