Programowanie w chmurze. Aplikacja typu Single Page App. Serverless

Wprowadzenie:
W tej karcie pracy nauczymy się wdrażać aplikację typu SPA w środowisku chmurowym GitHub Pages oraz zaprogramować te aplikacje z wykorzystaniem HTML, CSS oraz czystej implementacji języka JavaScript. Współcześnie można wykorzystywać różne frameworki JavaScriptowe do tworzenia tego typu aplikacji w szybszy sposób.

Single Page Application (SPA) to rodzaj aplikacji internetowej, w której nawigacja odbywa się poprzez asynchroniczne ładowanie poszczególnych elementów strony, takich jak sekcje lub całe widoki, bez konieczności przeładowywania całej strony. Dzięki temu użytkownik doświadcza płynniejszej interakcji, ponieważ zmiany w interfejsie są natychmiastowe i nie wymagają pełnego odświeżenia przeglądarki.

W aplikacjach SPA cała zawartość jest zazwyczaj ładowana jednorazowo przy pierwszym wejściu na stronę, a późniejsze interakcje z użytkownikiem prowadzą do dynamicznej aktualizacji wyświetlanych danych za pomocą JavaScript. To podejście pozwala na szybsze działanie aplikacji oraz lepsze wykorzystanie zasobów sieciowych, ponieważ jedynie zmieniane elementy są przesyłane między serwerem a klientem.
Więcej : https://developer.mozilla.org/en-US/docs/Glossary/SPA

Spróbujmy utworzyć pierwszą naszą aplikację typu SPA wykorzystując VanillaJS i środowisko Visual Studio Code bądź Atom aby stworzyć prawie najprostszą możliwą implementację aplikacji typu SPA.
Według dokumentacji na stronie http://vanilla-js.com/ Vanilla JS to szybki, lekki, cross-platformowy framework do budowania aplikacji w JavaScript. Tak naprawdę jest to czysty JavaScript a nazwa

funkcjonuje wśród środowiska developerskiego trochę prześmiewczo. Vanilla JS to także próba zwrócenia uwagi na sam język JavaScript i jego możliwości bez dodatkowych bibliotek.

Język JavaScript i DOM zapewnia wszystko co jest potrzebne do tworzenia aplikacji internetowych. DOM czyli Document Object Model to model opisujący jak zaprezentować tekstowe dokumenty HTML w postaci modelu obiektowego w pamięci komputera. Stanowi API dla dokumentów HTML i XML. Odpowiada za dwie rzeczy: zapewnia reprezentację struktury dokumentu oraz określa, w jaki sposób odnosić się do tej struktury z poziomu skryptu. DOM łączy stronę ze skryptami bądź językami programowania.

Tworzymy aplikację SPA z wykorzystaniem HTML i JS:

1. Utwórz nowy projekt o nazwie SPA-JS w wybranym przez siebie środowisku do tworzenia aplikacji internetowych. Utwórz katalog o nazwie js w którym znajdzie się implementacja kodu JavaScript.
2. Utwórz plik index.html w którym znajdować się będzie prosty kod HTML naszej aplikacji. Powinien wyglądać tak:

```
<!doctype html>
<html lang="en">
<head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
 <title>SPA PIAC TEST</title>
 <link rel="stylesheet" href="style.css">
</head>
<body>
  <header class="header">
    <ul class="Header-links-ul">
      <li class="header-link" id="about-link">About Me</li>
      <li class="header-link" id="contact-link">Contact</li>
    </ul>
  </header>
  <main>
   <h1 class="title">Hello World!</h1>
   <p>Lorem Ipsum is simply dummy text of the printing and typesetting industry...</p>
  </main>
  <script src="js/router.js"></script>
</body>
</html>
```

W tym pliku definiujemy dwa kontenery o nazwie header reprezentujący menu z linkami About Me i Contact oraz Main w którym wyświetlać się będą treści tych podstron, bez wywoływania ich w nowej zakładce.

3. W kolejnym kroku utwórz plik style.css w którym zdefiniujemy kaskadowe arkusze styli. Kod powinien wyglądać tak jak poniżej:

```
html, body {
   font-family: sans-serif;
   text-align: center;
   height: 100%;
   margin: 0;
   padding: 0;
   width: 100%;
}

header {
   display: flex;
   justify-content: space-around;
   align-items: center;
}

.Header-links-ul {
   width: 60%;
   list-style: none;
   display: flex;
   justify-content: space-around;
}

.header-link {
   padding: 0.4rem;
   border-radius: 2px;
   cursor: pointer;
}

.header-link:hover {
   border-bottom: 1px solid white;
}

main {
   padding: 2rem;
   font-size: 1rem;
}

.title {
  font-size: 3rem;
}
```

4. Utwórz plik router.js w katalogu js z następującym kodem:

```
 let pageUrls = {
   about: '/index.html?about',
   contact:'/index.html?contact'
};

function OnStartUp() {
   popStateHandler();
}

OnStartUp();

document.querySelector('#about-link').addEventListener('click', (event) => {
   let stateObj = { page: 'about' };
   document.title = 'About';
   history.pushState(stateObj, "about", "?about");
   RenderAboutPage();
});

document.querySelector('#contact-link').addEventListener('click', (event) => {
   let stateObj = { page: 'contact' };
   document.title = 'Contact';
   history.pushState(stateObj, "contact", "?contact");
   RenderContactPage();
});

function RenderAboutPage() {
   document.querySelector('main').innerHTML = `
      <h1 class="title">About Me</h1>
      <p>Lorem Ipsum is simply dummy text of the printing and typesetting industry...</p>`;
}

function RenderContactPage() {
   document.querySelector('main').innerHTML = `
      <h1 class="title">Contact with me</h1>
      <p>Lorem Ipsum is simply dummy text of the printing and typesetting industry...</p>`;
}

function popStateHandler() {
   let loc = window.location.href.toString().split(window.location.host)[1];

   if (loc === pageUrls.contact){ RenderContactPage(); }
   if(loc === pageUrls.about){ RenderAboutPage(); }
}

window.onpopstate = popStateHandler;
```

Powyższy kod zarządza nawigacją w aplikacji SPA bez przeładowywania całej strony. Umożliwia użytkownikowi przełączanie się między różnymi sekcjami aplikacji ("About" i "Contact") poprzez manipulację historią przeglądarki i dynamiczne aktualizowanie zawartości strony.
Oto szczegółowe wyjaśnienie, co robi każda część tego kodu:

- Deklaracja pageUrls:
  Obiekt pageUrls zawiera mapowanie nazw stron na ich odpowiednie URL-e z parametrami zapytania. Używane są do identyfikacji aktualnej strony w aplikacji.
- Funkcja OnStartUp:
  Funkcja OnStartUp jest wywoływana na początku, aby zainicjować stan aplikacji. W tym przypadku wywołuje funkcję popStateHandler, która renderuje stronę na podstawie aktualnego URL-a.
- Dodanie nasłuchiwaczy zdarzeń do linków:
  document.querySelector('#about-link').addEventListener('click', ...): Dodaje nasłuchiwacz zdarzeń do elementu z ID about-link. Po kliknięciu zmienia tytuł dokumentu, aktualizuje historię przeglądarki za pomocą history.pushState, i wywołuje funkcję RenderAboutPage, aby załadować treść strony "About". Podobnie działa nasłuchiwacz dla #contact-link, który ładuje stronę "Contact".
- Funkcje renderujące (RenderAboutPage i RenderContactPage):
  Te funkcje zmieniają zawartość elementu <main> w dokumencie HTML, aby wyświetlić odpowiednią treść dla stron "About" i "Contact".
- Funkcja popStateHandler:
  Ta funkcja jest wywoływana, gdy zmienia się stan historii przeglądarki (np. użytkownik używa przycisków "Wstecz" lub "Dalej"). Sprawdza aktualny URL i renderuje odpowiednią stronę na podstawie parametru zapytania w URL-u.
- Obsługa zdarzenia onpopstate:
  window.onpopstate = popStateHandler;: Rejestruje funkcję popStateHandler jako obsługę zdarzenia dla zmiany stanu historii przeglądarki.

5. Możesz dodać formularz kontaktowy na stronie "Contact". Dodaj poniższy kod do funkcji RenderContactPage w pliku router.js:

```
function RenderContactPage() {
  document.querySelector('main').innerHTML = `
    <h1 class="title">Contact with me</h1>
    <form id="contact-form">
      <label for="name">Name:</label>
      <input type="text" id="name" name="name" required>
      <label for="email">Email:</label>
      <input type="email" id="email" name="email" required>
      <label for="message">Message:</label>
      <textarea id="message" name="message" required></textarea>
      <button type="submit">Send</button>
    </form>`;

  document.getElementById('contact-form').addEventListener('submit', (event) => {
    event.preventDefault();
    alert('Form submitted!');
```

```
  });
}
```

6. Dodajmy możliwość przełączania między trybem jasnym a ciemnym za pomocą dodatkowego przełącznika w sekcji <header>:

```
<button id="theme-toggle">Toggle Theme</button>
```

7. Dodaj style dla ciemnego motywu:

```
body.dark-mode {
  background-color: #121212;
  color: #ffffff;
}

.dark-mode header {
  background-color: #1e1e1e;
}

.dark-mode .header-link:hover {
  border-bottom: 1px solid #ffffff;
}
```

8. Dodajmy funkcję do obsługi przełączania motywów w pliku router.js:

```
document.getElementById('theme-toggle').addEventListener('click', () => {
  document.body.classList.toggle('dark-mode');
});
```

9. Spróbujmy wdrożyć aplikację serverlessową w środowisku chmurowym:

Azure Static Web:
Azure Static Web oferuje darmową usługę hostingową pojedynczych stron internetowych. Możemy tworzyć aplikacje internetowe za pomocą platform i bibliotek języka JavaScript, takich jak Angular, React, Vue. Publikować witryny statyczne przy użyciu platform takich jak Gatsby, Hugo i VuePress. Wdrażać aplikacje internetowe za pomocą platform takich jakNext.js i Nuxt.js. Te aplikacje obejmują zasoby HTML, CSS, JavaScript i obrazy, które tworzą aplikację. Po utworzeniu zasobu Azure Static Web Apps platforma Azure współdziała bezpośrednio z usługą GitHub lub Azure DevOps.

Wdrażanie usługi w Azure Static Web App

1. Zaloguj się w serwisie https://portal.azure.com oraz znajdź usługę Static Wb Apps.
2. Wybierz utwórz statyczną aplikację sieci Web. Podaj szczegóły projektu takie jak subskrypcja i grupa zasobów, nazwę statycznej aplikacji oraz darmowy plan hostingu.
3. Wybierz dowolny region europejski do wdrożenia strony internetowej.
4. W szczegółach wdrożenia wybierz GitHub oraz zaloguj się za pomocą konta GitHub. Podaj organizację,repozytorium i gałąź projektu.
5. Następnie utwórz statyczną aplikację za pomocą przycisku przeglądanie + tworzenie.
6. Sprawdź czy strona internetowa wczytuje się.
7. Więcej : https://azure.microsoft.com/pl-pl/services/app-service/static/

GitHub Pages:
GitHub Pages to usługa hostingu witryn statycznych, która pobiera pliki HTML, CSS i JavaScript bezpośrednio z repozytorium na GitHub, opcjonalnie uruchamia je w procesie kompilacji i publikuje witrynę internetową. Można hostować swoją witrynę w domenie github.io lub we własnej domenie niestandardowej. Witryny GitHub Pages są domyślnie publicznie dostępne w Internecie, nawet jeśli repozytorium witryny jest prywatne lub wewnętrzne.

Wdrażanie usługi w Github Pages

1. Github Pages umożliwia stworzenie strony internetowej na bazie repozytorium oraz konta Github.
2. Utwórz w serwisie GitHub nowe repozytorium publiczne o nazwie nazwa użytkownika.github.io, gdzie nazwa użytkownika to Twoja nazwa użytkownika (lub nazwa organizacji) w serwisie GitHub.
3. Wyślij do repozytorium kod źródłowy swojej strony wraz z plikiem index.html.
   Kroki wdrożenia aplikacji SPA na GitHub Pages:
   Utwórz repozytorium na GitHubie:
   - Zaloguj się na GitHubie i utwórz nowe repozytorium o nazwie np. "my-spa".

   Inicjalizacja projektu lokalnie:
   - Otwórz terminal i przejdź do katalogu projektu.
   - Zainicjuj repozytorium git:
     git init
   - Dodaj wszystkie pliki do repozytorium:
     git add .
   - Zatwierdź zmiany:
     git commit -m "Initial commit"

   Wypchnięcie kodu na GitHub:
   - Dodaj zdalne repozytorium:
     git remote add origin https://github.com/TWOJ_UZYTKOWNIK/my-spa.git
   - Wypchnij kod na GitHuba:
     git push -u origin master
4. Konfiguracja GitHub Pages:
   - Przejdź do ustawień repozytorium na GitHubie.
   - W sekcji "GitHub Pages" wybierz gałąź, którą chcesz użyć jako źródło (najczęściej main lub master) i zapisz zmiany.
   - Po wykonaniu tych kroków Twoja aplikacja SPA będzie dostępna online pod adresem https://TWOJ_UZYTKOWNIK.github.io/my-spa.
5. Sprawdź czy strona internetowa jest dostępna pod utworzonym przez Ciebie adresem w serwisie github.io Więcej : https://pages.github.com/

10. Utwórz nową stronę "Gallery" i dodaj do niej galerię zdjęć. wyświetl je w postaci miniatur w układzie 3x3. Upewnij się, że obrazy są ładowane asynchronicznie jako obiekty BLOB. Dodaj funkcjonalność lazy loading, aby obrazy były ładowane dopiero wtedy, gdy użytkownik przewinie stronę do ich widoczności. Dodaj możliwość powiększenia zdjęcia w modalnym oknie po kliknięciu na miniaturkę. Upewnij się, że modal można zamknąć klikając na przycisk "zamknij" lub poza obrazem.

11. Rozbuduj stronę kontaktową o formularz z polami imię, e-mail i wiadomość. Zaimplementuj walidację formularza przed jego wysłaniem oraz mechanizm reCAPTCHA.

12. Przeprowadź testy funkcjonalności aplikacji wykorzystując narzędzia developerskie ChromeDevTools.
    - Sprawdź poprawność działania nawigacji między stronami (About, Contact, Gallery).
    - Zweryfikuj poprawność ładowania obrazów w galerii (lazy loading i modal).
    - Jak możesz wykorzystać zakładkę Lighthouse w testach aplikacji SPA?
    - Przetestuj walidację formularza kontaktowego oraz działanie reCAPTCHA.
    - Przetestuj aplikację na różnych przeglądarkach i rozmiarach urządzeń aby upewnić się, że działa poprawnie w różnych środowiskach.
    - Jakie są główne wyzwania techniczne związane z budową aplikacji SPA?