Zaawansowany Programista JS

Dzień 5

Plan

- Podstawy asynchroniczności + instrukcje asynchroniczne `setTimeout` i `setInterval`
- Podstawy HTTP
  - Czym jest internet
  - Czym jest domena i hosting
  - Czym jest client i serwer
  - Nagłówki HTTP
  - Metody HTTP
  - Headery HTTP
  - Zasady pracy z HTTP
- Narzędzie `fetch`
  - Pobieranie danych za pomocą `fetch`
  - Wysyłanie danych za pomocą `fetch`
- API
  - Podstawy endpointów i API
  - Rest API
- Tworzenie aplikacji przy uzyciu fetch i public Rest API

Podstawy asynchroniczności + instrukcje asynchroniczne `setTimeout` i `setInterval`

Asynchroniczność w JavaScript odnosi się do sposobu, w jaki język ten obsługuje operacje wykonywane w tle, takie jak żądania sieciowe czy odczyty plików. Dzięki asynchronicznej obsłudze takich operacji, program nie musi czekać na zakończenie ich wykonania i może kontynuować pracę z innymi częściami aplikacji.

W JavaScript asynchroniczność jest realizowana za pomocą mechanizmu zwanych funkcjami zwrotnymi (callback functions) oraz obiektami Promise. Funkcje zwrotne to funkcje, które są przekazywane do innych funkcji jako argumenty i są wywoływane z powrotem po zakończeniu określonej operacji. Obiekty Promise natomiast są obiektami, które reprezentują wartość, która może być dostępna teraz, w przyszłości lub nigdy. W praktyce obiekt Promise zwraca wartość z funkcji asynchronicznej, która może być dostępna dopiero po wykonaniu operacji w tle.

Mechanizm asynchroniczności pozwala na wykonywanie operacji w tle, takich jak żądania sieciowe lub odczyty plików, bez blokowania interfejsu użytkownika. Dzięki temu użytkownik może nadal korzystać z aplikacji i wykonywać inne czynności, podczas gdy aplikacja zajmuje się operacjami w tle. W JavaScript asynchroniczność jest szczególnie przydatna w przypadku operacji, które wymagają pobrania danych z zewnętrznych źródeł, takich jak serwery internetowe lub bazy danych, ponieważ pobieranie takich danych może zająć dużo czasu i spowolnić działanie aplikacji, gdyby operacje były wykonywane synchronicznie.

Metody `setTimeout()` i `setInterval()` w JavaScript to funkcje umożliwiające wykonywanie innych funkcji w sposób asynchroniczny, tzn. w tle, po upływie określonego czasu lub z określoną częstotliwością.

Metoda `setTimeout()` umożliwia wywołanie określonej funkcji po upływie określonego czasu. Funkcja ta przyjmuje dwa argumenty: pierwszy to funkcja, która ma zostać wywołana, a drugi to czas w milisekundach, po którym ma nastąpić wywołanie tej funkcji. Przykładowe użycie tej metody wygląda następująco:

```
setTimeout(() => {
  console.log('Funkcja została wywołana po upływie 3 sekund');
}, 3000);
```

W tym przykładzie funkcja `console.log()` zostanie wywołana po upływie 3 sekund.

Metoda `setInterval()` umożliwia wywoływanie funkcji w regularnych odstępach czasu. Funkcja ta przyjmuje dwa argumenty: pierwszy to funkcja, która ma zostać wywołana, a drugi to czas w milisekundach, po którym ma nastąpić kolejne wywołanie tej funkcji. Przykładowe użycie tej metody wygląda następująco:

```
let count = 0;

const interval = setInterval(() => {
  count++;
  console.log('Funkcja została wywołana ' + count + ' razy');
}, 1000);
```

W tym przykładzie funkcja `console.log()` będzie wywoływana co sekundę, zwiększając wartość zmiennej count o jeden przy każdym wywołaniu.

Obie te metody zwracają identyfikator wywołania, który może być użyty do anulowania wykonania funkcji za pomocą metody `clearTimeout()` lub `clearInterval()`.

Podstawy HTTP

HTTP (ang. Hypertext Transfer Protocol) to protokół sieciowy, który służy do przesyłania danych między klientem (np. przeglądarką internetową) a serwerem w Internecie. Jest to protokół bezstanowy, co oznacza, że każde żądanie jest niezależne od poprzednich żądań, a serwer nie przechowuje informacji o poprzednich żądaniach.

HTTP został zaprojektowany do przesyłania dokumentów hipertekstowych (np. stron internetowych), ale może być wykorzystywany również do przesyłania innych rodzajów danych, takich jak pliki multimedialne, aplikacje internetowe czy serwisy API.

W protokole HTTP wyróżnia się dwie podstawowe metody, jakie może wykonać klient: GET i POST. Metoda GET służy do pobierania danych z serwera, natomiast metoda POST służy do

przesyłania danych z klienta do serwera, np. poprzez formularz internetowy.

Współcześnie protokół HTTP jest używany w wersji 1.1, która wprowadza wiele usprawnień, w tym m.in. możliwość korzystania z nagłówków Keep-Alive do utrzymania połączenia między klientem a serwerem oraz z kompresji danych w celu zwiększenia szybkości przesyłania. Obecnie wdrażana jest również wersja HTTP/2, która wprowadza jeszcze większe usprawnienia w przesyłaniu danych w Internecie.

Czym jest internet

Internet to globalna sieć komputerowa, która łączy ze sobą miliony komputerów na całym świecie. Jest to sieć otwarta, zdecentralizowana i niezależna od jakiejkolwiek instytucji czy organizacji. Internet umożliwia przesyłanie informacji w postaci tekstów, zdjęć, plików multimedialnych, a także pozwala na komunikację w czasie rzeczywistym, np. poprzez czaty, wideokonferencje czy rozmowy głosowe.

Internet składa się z wielu sieci, które są ze sobą połączone. Do przesyłania danych wykorzystywane są różne protokoły, m.in. protokół HTTP, FTP, SMTP czy POP3. Internet pozwala na dostęp do różnego rodzaju informacji i usług, takich jak strony internetowe, serwisy społecznościowe, sklepy internetowe czy aplikacje internetowe.

Czym jest domena i hosting

Domena to unikalny adres internetowy, pod którym znajduje się strona internetowa lub inna usługa dostępna w Internecie. Składa się z nazwy i rozszerzenia, np. "google.com". Domena umożliwia łatwe zapamiętanie adresu strony lub usługi, a także ułatwia wyszukiwanie w Internecie. Domeny można rejestrować za pośrednictwem firm specjalizujących się w rejestracji domen, a ich koszt zależy od popularności danego adresu oraz od wybranej końcówki (tzw. TLD - Top Level Domain), np. ".com", ".pl", ".org".

Hosting to usługa polegająca na udostępnianiu miejsca na serwerze, na którym można umieścić swoją stronę internetową lub aplikację internetową, tak aby była dostępna dla użytkowników z całego świata. Hosting może oferować różne funkcjonalności, takie jak bazy danych, poczta e-mail, zaplecze techniczne czy wsparcie dla różnych języków programowania. Koszt usługi hostingowej zależy od wybranego planu, mocy serwera, dostępnych zasobów, a także od wybranej firmy hostingowej. Hosting można zakupić u wielu dostawców na rynku, którzy oferują różne pakiety usług w zależności od potrzeb klienta.

Czym jest client i serwer

Client i server to dwa podstawowe pojęcia związane z architekturą sieciową i systemami rozproszonymi.

Client (klient) to komputer lub inny urządzenie końcowe, które korzysta z usług udostępnianych przez serwer. Przykładowymi klientami są przeglądarki internetowe, programy pocztowe czy aplikacje mobilne. Klient wysyła zapytania do serwera, oczekując na odpowiedź, która może przyjąć różne formy, np. plik, stronę internetową, dane z bazy danych czy usługę sieciową.

Server (serwer) to komputer lub inny urządzenie, które udostępnia usługi klientom poprzez sieć. Serwer może oferować różnego rodzaju usługi, np. hosting stron internetowych, udostępnianie plików, przetwarzanie danych, udostępnianie baz danych

czy usługi sieciowe. Serwer odbiera zapytania od klientów i przetwarza je, wysyłając w odpowiedzi żądane informacje.

Współpraca między klientem a serwerem odbywa się na zasadzie komunikacji sieciowej, z wykorzystaniem różnych protokołów, np. HTTP, FTP, SMTP czy POP3. Client i server mogą działać na różnych platformach sprzętowych i oprogramowaniowych, a ich rola może być wymieniana w zależności od konkretnych potrzeb aplikacji lub usługi.

Nagłówki HTTP

Nagłówki HTTP to dane przesyłane wraz z żądaniem lub odpowiedzią HTTP, które zawierają dodatkowe informacje o transakcji sieciowej. Nagłówki HTTP składają się z pól nagłówka, które zawierają wartości, opisujące poszczególne aspekty transakcji.

Nagłówki HTTP umożliwiają m.in. przesyłanie informacji o:

- Typie MIME przesyłanych danych
- Wersji protokołu HTTP
- Kodowaniu treści
- Daty i czasie transakcji
- Przeglądarce lub aplikacji, z której zostało wysłane żądanie
- Parametrach żądania, takich jak dane formularza czy dane sesji
- Kodzie odpowiedzi HTTP, takim jak kod 404 (strona nie znaleziona) czy 200 (ok)

Nagłówki HTTP pozwalają na ustanowienie i kontrolowanie sesji między klientem a serwerem oraz na dostarczanie informacji, które nie są bezpośrednio związane z treścią przesyłaną w żądaniu lub odpowiedzi. Nagłówki HTTP mogą być dodawane, modyfikowane lub usuwane przez przeglądarki, serwery lub aplikacje internetowe w zależności od konkretnych potrzeb.

Metody HTTP

Metody HTTP to standardowe sposoby przekazywania żądań i odpowiedzi między klientem a serwerem w protokole HTTP. Najczęściej wykorzystywanymi metodami HTTP są

- GET - pobranie zasobu z serwera. Metoda ta przesyła żądanie pobrania określonego zasobu, np. strony internetowej lub pliku, znajdującego się na serwerze. Dane są przekazywane w nagłówkach żądania.
- POST - przesłanie danych do serwera. Metoda ta przesyła żądanie przesłania danych do serwera, np. danych formularza, które mają zostać zapisane w bazie danych. Dane są przekazywane w treści żądania.
- PUT - aktualizacja istniejącego zasobu na serwerze. Metoda ta przesyła żądanie aktualizacji określonego zasobu, np. pliku lub danych w bazie danych. Dane są przekazywane w treści żądania.
- DELETE - usunięcie zasobu z serwera. Metoda ta przesyła żądanie usunięcia określonego zasobu, np. pliku lub rekordu w bazie danych.

Kody HTTP

Kody odpowiedzi HTTP to numeryczne wartości zwracane przez serwer w odpowiedzi na żądanie HTTP wysłane przez klienta. Najważniejsze kody odpowiedzi HTTP to

- 1xx - informacyjne - informacja dla klienta o statusie żądania, np. 100 (Continue) oznacza, że żądanie zostało przyjęte, a klient powinien kontynuować wysyłanie ciała żądania.
- 2xx - sukces - żądanie zostało pomyślnie przetworzone przez serwer, np. 200 (OK) oznacza, że żądanie zostało pomyślnie przetworzone i odpowiedź zawiera żądane informacje.
- 3xx - przekierowanie - klient musi podjąć dodatkowe działania w celu ukończenia żądania, np. 301 (Moved Permanently) oznacza, że żądany zasób został przeniesiony na inny adres.
- 4xx - błąd klienta - serwer otrzymał nieprawidłowe żądanie od klienta, np. 404 (Not Found) oznacza, że żądany zasób nie został znaleziony na serwerze.
- 5xx - błąd serwera - serwer napotkał problem podczas przetwarzania żądania, np. 500 (Internal Server Error) oznacza, że serwer napotkał nieoczekiwany błąd.

Niektóre inne przykładowe kody odpowiedzi HTTP to

- 201 (Created) - nowy zasób został utworzony na serwerze
- 204 (No Content) - serwer przetworzył żądanie, ale nie zwrócił żadnej treści
- 400 (Bad Request) - żądanie jest nieprawidłowe lub niekompletne
- 401 (Unauthorized) - klient nie ma uprawnień do wykonania żądanego działania
- 403 (Forbidden) - klient ma uprawnienia do wykonania żądanego działania, ale serwer odmawia dostępu
- 503 (Service Unavailable) - serwer jest tymczasowo niedostępny Kody odpowiedzi HTTP pozwalają na zrozumienie wyniku żądania i wskazują na potencjalne problemy, które mogą wystąpić w trakcie przetwarzania żądania.

Zasady pracy z HTTP

Dobre praktyki pracy z HTTP

- Używaj bezpiecznych protokołów - zawsze używaj protokołów bezpiecznych, takich jak HTTPS, aby chronić dane przesyłane między klientem a serwerem.
- Używaj prawidłowych metod HTTP - używaj odpowiednich metod HTTP w zależności od rodzaju działania, np. GET do pobierania danych, POST do przesyłania danych, DELETE do usuwania danych itp.
- Używaj nagłówków HTTP w sposób właściwy - używaj nagłówków HTTP, aby przekazywać dodatkowe informacje o żądaniu i odpowiedzi, np. Content-Type do określania typu danych, Cache-Control do określania sposobu cache'owania itp.
- Weryfikuj dane wejściowe - upewnij się, że dane przesyłane przez klienta są prawidłowe i nie stanowią zagrożenia dla serwera, np. poprzez walidację danych na stronie klienta i serwerze.
- Zwracaj odpowiednie kody odpowiedzi - używaj odpowiednich kodów odpowiedzi HTTP, aby informować klienta o wyniku żądania, np. 200 (OK) w przypadku pomyślnego żądania, 404 (Not Found) w przypadku braku zasobu itp.

Narzędzie `fetch` i Promise

Fetch to wbudowany w JavaScript interfejs umożliwiający wysyłanie żądań sieciowych (np. pobieranie danych z serwera) i obsługę odpowiedzi. Fetch używa obietnic, dzięki czemu może być używany w asynchroniczny sposób.

Fetch pozwala na łatwe wysyłanie żądań HTTP, a także obsługę odpowiedzi, które mogą być w formacie JSON, XML lub innych. Obsługuje wiele różnych typów żądań HTTP, w tym GET, POST, PUT, DELETE i inne. Fetch używa też tzw. Promise, które umożliwiają łatwe i czytelne programowanie asynchroniczne.

Promise to mechanizm w JavaScript, który umożliwia asynchroniczne programowanie. Promise jest obiektem, który reprezentuje wartość, która może nie być dostępna jeszcze, ale zostanie dostarczona w przyszłości. Promise może mieć trzy stany:

- Oczekujący (pending): stan początkowy, kiedy obietnica nie została jeszcze spełniona ani odrzucona
- Spełniony (fulfilled): stan, w którym obietnica została spełniona i zwrócona wartość jest dostępna
- Odrzucony (rejected): stan, w którym obietnica nie została spełniona i zwracany jest błąd

Pobieranie danych za pomocą `fetch`

Przykład użycia fetch do pobrania danych w formacie JSON z serwera wygląda następująco

```
fetch('https://example.com/data.json')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error(error));
```

W tym przykładzie, wywołujemy funkcję `fetch()` z adresem URL naszego pliku JSON. Następnie wykorzystujemy metodę `json()` obiektu odpowiedzi, aby przetworzyć odpowiedź jako dane w formacie JSON. W końcu, wykorzystujemy Promise `.then()` do obsługi danych i `.catch()` do obsługi błędów.

Wysyłanie danych za pomocą `fetch`

Aby wysłać zapytanie POST za pomocą fetch() w JavaScript, należy utworzyć obiekt opcji i przekazać go jako drugi argument do metody fetch(). Opcje te powinny zawierać informacje o żądaniu, takie jak adres URL, nagłówki i ciało zapytania. Aby przekazać dane jako ciało zapytania, należy użyć opcji body.

```
fetch(url, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(data)
})
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error(error));
```

W tym przykładzie, przekazujemy adres URL i opcje, które zawierają metodę POST, nagłówki z informacją o formacie danych oraz dane, które zostaną przesłane jako ciało

zapytania. W tym przypadku, dane są w formacie JSON i dlatego używamy metody `JSON.stringify()` do przetworzenia obiektu data na ciąg znaków JSON.

Następnie, wywołujemy metodę `.then()` na wyniku zapytania, aby przetworzyć odpowiedź, która przychodzi z serwera. W tym przypadku, używamy metody `.json()` do przekształcenia odpowiedzi w formacie JSON na obiekt JavaScript. Na końcu, wyświetlamy przetworzone dane w konsoli.

W przypadku wystąpienia błędu, używamy metody `.catch()` do jego obsługi i wyświetlenia komunikatu o błędzie.

API

API (Application Programming Interface) to interfejs programistyczny, który udostępnia zestaw narzędzi, funkcji i protokołów komunikacyjnych, które pozwalają programom na komunikację i współpracę z innymi programami, systemami operacyjnymi, aplikacjami internetowymi i urządzeniami.

API umożliwiają programistom na integrację i wykorzystanie gotowych usług, funkcjonalności i zasobów udostępnianych przez różne aplikacje i serwisy. Dzięki temu, programy mogą działać razem i wymieniać się informacjami bez potrzeby tworzenia wszystkiego od zera.

API mogą przyjmować różne formy, na przykład interfejsy programistyczne w języku JavaScript, biblioteki i frameworki, protokoły sieciowe, interfejsy użytkownika i wiele innych. Przykładami popularnych API są np. API Google Maps, Facebook API czy Twitter API.

Podstawy endpointów i API

Endpointy to adresy URL, pod którymi dostępne są konkretne zasoby, usługi lub funkcjonalności w ramach danego API. Endpointy umożliwiają programistom na komunikację i wymianę danych z serwerami za pomocą standardowych metod HTTP, takich jak GET, POST, PUT, PATCH, DELETE itp.

Na przykład, API do zarządzania użytkownikami może udostępniać endpoint /users, pod którym dostępna jest lista wszystkich użytkowników, oraz endpointy /users/{id}, gdzie {id} to identyfikator konkretnego użytkownika, które pozwalają na uzyskanie, edycję lub usunięcie danych użytkownika.

Dzięki endpointom, programiści mogą łatwo i bezpiecznie uzyskiwać i przetwarzać dane z różnych źródeł, korzystając z już istniejących usług i aplikacji. Endpointy są jednym z podstawowych elementów API i ich poprawne definiowanie i dokumentowanie jest kluczowe dla wydajnej i bezpiecznej pracy z API.

Rest API

REST (Representational State Transfer) API to styl architektury oprogramowania, który określa zasady i protokoły komunikacji pomiędzy aplikacjami, w których kluczową rolę odgrywają zasoby i operacje na tych zasobach. REST jest jednym z najczęściej stosowanych stylów architektury dla tworzenia API, zarówno wewnątrz jak i na zewnątrz organizacji.

REST API to interfejs programistyczny, który umożliwia programistom na przesyłanie i odbieranie danych między klientem (aplikacją lub serwisem) a serwerem za pomocą

standardowych protokołów HTTP, takich jak GET, POST, PUT, DELETE itp. Dane przesyłane w formacie JSON, XML lub innych.

REST API opiera się na sześciu głównych zasadach, które określają, jakie powinny być endpointy, jakie metody HTTP powinny być używane do manipulowania zasobami oraz jakie powinny być sposoby autoryzacji i uwierzytelniania dostępu do API. Te zasady to

- Klient-serwer: Separacja interfejsu użytkownika i logiki serwera
- Bezstanowość: Każde zapytanie od klienta do serwera musi zawierać wszystkie informacje potrzebne do zrozumienia zapytania. Stan sesji przechowywany jest po stronie klienta.
- Protokół warstwowy: Warstwy architektury powinny działać niezależnie, z wykorzystaniem różnych technologii.
- Interfejs jednorodny: Jednoznacznie zdefiniowane zasoby i operacje na nich.
- Możliwość buforowania: Odpowiedzi powinny mieć metadane, które pozwalają na określenie, czy można je buforować czy też nie.
- Bezpieczeństwo: Mechanizmy zabezpieczające przed atakami oraz kontrola dostępu.

REST API ma wiele zalet, takich jak prostota, skalowalność, niezależność od platformy i języka programowania, a także możliwość korzystania z gotowych narzędzi i bibliotek, co znacznie przyspiesza i ułatwia pracę programistów.

Tworzenie aplikacji przy uzyciu fetch i public Rest API

```
// Napisz aplikacje w JavaScript z wykorzystaniem biblioteki fetch, która spełnia następujące kryteria

// Korzystając z endpointa public API np https://openlibrary.org/works/OL15626917W.json wyswietl na stronie poszczegolne informacje.

// 1. Tytul
// 2. Tematyke
// 3. Opis

// Korzystając z drugiego endpointa https://openlibrary.org/authors/OL33421A.json dopisz informacje o autorze

// 1. Nazwa
```
