Projektowanie Aplikacji Internetowych

Wykład 1

Wprowadzenie do Frontend Developmentu

Mateusz Pawełkiewicz

1. Czym jest frontend i jego rola w tworzeniu aplikacji webowych

1.1 Definicja frontendu

Frontend to część aplikacji lub strony internetowej, która jest bezpośrednio widoczna i interaktywna dla użytkownika. Obejmuje wszystkie elementy, z którymi użytkownik może wchodzić w interakcję za pomocą przeglądarki internetowej, takie jak tekst, obrazy, przyciski, formularze, animacje i inne elementy wizualne.

1.2 Rola frontendu w aplikacjach webowych

- **Interfejs użytkownika (UI)**: Frontend odpowiada za wygląd i układ elementów na stronie. UI ma na celu zapewnienie estetycznego i spójnego wyglądu aplikacji, który jest zgodny z oczekiwaniami użytkowników.
- **Doświadczenie użytkownika (UX)**: Frontend wpływa na to, jak użytkownik odbiera interakcję z aplikacją. Dobre UX oznacza, że strona jest intuicyjna, łatwa w nawigacji i dostarcza satysfakcjonujących doświadczeń.
- **Interaktywność**: Dzięki zastosowaniu języka JavaScript i jego bibliotek, frontend umożliwia dynamiczne reagowanie na działania użytkownika, takie jak kliknięcia, wprowadzanie danych czy nawigacja.
- **Komunikacja z backendem**: Frontend często współpracuje z backendem poprzez interfejsy API, wysyłając i odbierając dane w celu aktualizacji treści bez konieczności przeładowywania strony.

1.3 Znaczenie frontendu w biznesie

- **Pierwsze wrażenie**: Strona internetowa często jest pierwszym punktem kontaktu klienta z firmą. Profesjonalny i atrakcyjny frontend może przyciągnąć użytkowników i zachęcić ich do dalszej interakcji.
- **Dostępność i inkluzywność**: Poprzez odpowiednie praktyki kodowania, frontend może zapewnić dostęp do treści dla osób z niepełnosprawnościami, co zwiększa zasięg i pozytywny odbiór marki.
- **Konkurencyjność**: Nowoczesne technologie frontendowe pozwalają tworzyć innowacyjne rozwiązania, które wyróżniają firmę na tle konkurencji.

---

2. Różnice między frontendem a backendem

2.1 Frontend (strona klienta)

- **Definicja**: Frontend to warstwa prezentacji aplikacji, z którą użytkownik bezpośrednio interaguje.
- **Technologie**:
    - **HTML** (HyperText Markup Language): Odpowiada za strukturę strony.
    - **CSS** (Cascading Style Sheets): Odpowiada za stylizację i wygląd strony.
    - **JavaScript**: Odpowiada za interaktywność i logikę po stronie klienta.

- **Zadania**:
    - Tworzenie responsywnych interfejsów użytkownika.
    - Zapewnienie płynnej nawigacji i interakcji.
    - Walidacja danych po stronie klienta przed ich wysłaniem do serwera.

2.2 Backend (strona serwera)

- **Definicja**: Backend to część aplikacji działająca po stronie serwera, odpowiedzialna za logikę biznesową, przetwarzanie danych i komunikację z bazami danych.
- **Technologie**:
    - Języki programowania: **Java**, **Python**, **Ruby**, **PHP**, **Node.js**.
    - Bazy danych: **MySQL**, **PostgreSQL**, **MongoDB**.
- **Zadania**:
    - Przechowywanie i zarządzanie danymi.
    - Uwierzytelnianie i autoryzacja użytkowników.
    - Logika biznesowa aplikacji (np. obliczenia, przetwarzanie danych).

2.3 Współpraca między frontendem a backendem

- **API (Application Programming Interface)**: Interfejsy, które umożliwiają komunikację między frontendem a backendem poprzez wymianę danych w formacie JSON lub XML.
- **Asynchroniczność**: Frontend może wysyłać żądania do backendu i otrzymywać odpowiedzi bez konieczności przeładowywania strony, dzięki technologiom takim jak AJAX czy Fetch API.
- **Bezpieczeństwo**: Ważne jest, aby zarówno frontend, jak i backend odpowiednio walidowały dane oraz stosowały środki bezpieczeństwa, takie jak szyfrowanie danych czy tokeny uwierzytelniające.

---

3. Przegląd podstawowych technologii frontendowych

3.1 HTML (HyperText Markup Language)

3.1.1 Struktura dokumentu HTML

- **Deklaracja doctype**: <!DOCTYPE html> – informuje przeglądarkę o wersji HTML.
- **Elementy główne**:
    - <html>: Element nadrzędny zawierający cały dokument.
    - <head>: Sekcja zawierająca meta informacje, tytuł strony (<title>), odwołania do arkuszy stylów i skryptów.
    - <body>: Sekcja zawierająca treść strony, która jest widoczna dla użytkownika.

3.1.2 Elementy semantyczne w HTML5

- **Znaczenie semantyki**: Używanie odpowiednich znaczników poprawia dostępność strony oraz jej pozycjonowanie w wyszukiwarkach.

- **Przykłady elementów**:
    - <header>: Nagłówek strony lub sekcji.
    - <nav>: Nawigacja główna.
    - <main>: Główna treść strony.
    - <section>: Sekcja tematyczna.
    - <article>: Samodzielny fragment treści.
    - <aside>: Treści poboczne, np. paski boczne.
    - <footer>: Stopka strony lub sekcji.

3.1.3 Tworzenie formularzy i elementów interaktywnych

- **Formularze**: Pozwalają na zbieranie danych od użytkowników.
    - <form>: Element kontenerowy dla formularza.
    - <input>: Pole wprowadzania danych (różne typy: tekst, email, hasło, plik, itd.).
    - <label>: Etykieta dla pola formularza.
    - <textarea>: Wieloliniowe pole tekstowe.
    - <button>: Przyciski akcji.
- **Nowe elementy w HTML5**:
    - Elementy multimedialne: <audio>, <video>, <canvas>.
    - Elementy interaktywne: <details>, <summary>, <dialog>.

3.1.4 Dobre praktyki

- **Walidacja kodu**: Używanie narzędzi takich jak W3C Validator do sprawdzania poprawności kodu.
- **Dostępność**: Używanie atrybutów alt dla obrazów, aria-labels dla elementów interaktywnych.
- **SEO (Search Engine Optimization)**: Poprawna struktura i semantyka pomagają w lepszym pozycjonowaniu strony w wynikach wyszukiwania.

3.2 CSS (Cascading Style Sheets)

3.2.1 Podstawy składni CSS

- **Reguła CSS**: Składa się z selektora i deklaracji.
    - **Selektor**: Określa, do jakich elementów zostaną zastosowane style (np. .klasa, #id, tag).
    - **Deklaracja**: Para właściwość-wartość umieszczona w nawiasach klamrowych { } (np. color: red;).

3.2.2 Modele układu strony

- **Model pudełkowy (Box Model)**:
    - Określa, jak przeglądarka oblicza wymiary elementów.
    - Składa się z: zawartości (content), paddingu, obramowania (border) i marginesu (margin).
- **Pozycjonowanie elementów**:
    - **Static**: Domyślne położenie w dokumencie.
    - **Relative**: Pozycja względem swojego normalnego położenia.
    - **Absolute**: Pozycja względem najbliższego przodka z pozycją inną niż static.

- **Fixed**: Pozycja względem okna przeglądarki.
- **Sticky**: Połączenie relative i fixed, element przykleja się po osiągnięciu określonej pozycji.

3.2.3 Techniki układu strony

- **Flexbox**:
  - Zaprojektowany do układu w jednym wymiarze (rząd lub kolumna).
  - Właściwości rodzica: display: flex;, flex-direction, justify-content, align-items.
  - Właściwości dzieci: flex-grow, flex-shrink, flex-basis, order.
- **CSS Grid**:
  - Umożliwia tworzenie złożonych układów dwuwymiarowych.
  - Definiowanie siatki: display: grid;, grid-template-columns, grid-template-rows, grid-gap.
  - Pozycjonowanie elementów: grid-column, grid-row.

3.2.4 Responsywność i projektowanie mobilne

- **Media Queries**:
  - Umożliwiają stosowanie stylów w zależności od właściwości urządzenia.
  - Przykład: @media (max-width: 768px) { ... } – style dla urządzeń o szerokości ekranu mniejszej niż 768px.
- **Mobile First**:
  - Strategia projektowania najpierw dla urządzeń mobilnych, a następnie rozszerzanie stylów dla większych ekranów.

3.2.5 Animacje i przejścia

- **Przejścia (Transitions)**:
  - Umożliwiają płynne przejście między wartościami właściwości.
  - Właściwości: transition-property, transition-duration, transition-timing-function, transition-delay.
- **Animacje (Animations)**:
  - Definiowanie klatek kluczowych: @keyframes nazwaAnimacji { ... }.
  - Właściwości: animation-name, animation-duration, animation-iteration-count, animation-direction.

3.3 JavaScript

3.3.1 Wprowadzenie do języka JavaScript

- **Historia**:
  - Stworzony w 1995 roku przez Brendana Eicha.
  - Początkowo służył do prostych interakcji na stronach WWW.
  - Obecnie pełnoprawny język programowania używany zarówno na frontendzie, jak i backendzie (Node.js).

3.3.2 Zmienne, typy danych i operatory

- **Deklaracja zmiennych**:
  - var – zasięg funkcji lub globalny.
  - let – zasięg bloku, wprowadzony w ES6.
  - const – stała, zasięg bloku, wartość nie może być ponownie przypisana.
- **Typy danych**:
  - **Prymitywne**: string, number, boolean, null, undefined, symbol.
  - **Obiekty**: Tablice, funkcje, obiekty użytkownika.
- **Operatory**:
  - **Arytmetyczne**: +, -, *, /, %, ** (potęgowanie).
  - **Porównania**: ==, ===, !=, !==, <, >, <=, >=.
  - **Logiczne**: && (i), || (lub), ! (nie).

3.3.3 Struktury kontrolne

- **Instrukcje warunkowe**:
  - if (warunek) { ... } else { ... }
  - switch (wyrażenie) { case wartość: ... break; }
- **Pętle**:
  - for (inicjalizacja; warunek; inkrementacja) { ... }
  - while (warunek) { ... }
  - do { ... } while (warunek);
  - Iteracja po kolekcjach:
    - for...of – iteracja po wartościach (tablice, stringi).
    - for...in – iteracja po kluczach obiektu.

3.3.4 Funkcje i zakresy zmiennych

- **Deklaracja funkcji**:
  - Funkcje nazwane: function nazwaFunkcji(parametry) { ... }
  - Funkcje anonimowe: const funkcja = function(parametry) { ... };
  - Funkcje strzałkowe (ES6): const funkcja = (parametry) => { ... };
- **Zakresy zmiennych**:
  - **Globalny**: Zmienne dostępne w całym skrypcie.
  - **Funkcyjny**: Zmienne dostępne tylko wewnątrz funkcji.
  - **Blokowy**: Zmienne zadeklarowane za pomocą let lub const dostępne tylko w obrębie bloku { }.

3.3.5 Manipulacja Document Object Model (DOM)

- **Wybór elementów**:
  - document.getElementById('id') – wybiera element o danym id.
  - document.getElementsByClassName('klasa') – wybiera wszystkie elementy z daną klasą.
  - document.querySelector('selektor') – wybiera pierwszy element pasujący do selektora CSS.
  - document.querySelectorAll('selektor') – wybiera wszystkie elementy pasujące do selektora CSS.
- **Modyfikacja treści i atrybutów**:
  - element.textContent – ustawianie lub pobieranie tekstu wewnątrz elementu.
  - element.innerHTML – ustawianie lub pobieranie kodu HTML wewnątrz elementu.

    - element.setAttribute('nazwaAtrybutu', 'wartość') – ustawianie atrybutu.
    - element.getAttribute('nazwaAtrybutu') – pobieranie wartości atrybutu.
- **Stylowanie elementów**:
    - element.style.property – ustawianie stylu bezpośrednio na elemencie (np. element.style.color = 'blue';).
- **Tworzenie i usuwanie elementów**:
    - document.createElement('tag') – tworzenie nowego elementu.
    - element.appendChild(nowyElement) – dodawanie elementu do DOM.
    - element.removeChild(childElement) – usuwanie elementu z DOM.

3.3.6 Obsługa zdarzeń w przeglądarce

- **Dodawanie nasłuchiwaczy zdarzeń**:
    - element.addEventListener('typZdarzenia', funkcja);
    - Przykład: button.addEventListener('click', () => { alert('Kliknięto przycisk'); });
- **Typy zdarzeń**:
    - **Myszka**: click, dblclick, mouseover, mouseout, mousedown, mouseup.
    - **Klawiatura**: keydown, keyup, keypress.
    - **Formularze**: submit, focus, blur, change.
    - **Zdarzenia okna**: load, resize, scroll, unload.
- **Obiekt zdarzenia**:
    - Dostarcza informacje o zdarzeniu.
    - Dostępny jako parametr funkcji obsługującej zdarzenie (np. event lub e).
    - Przykład: element.addEventListener('click', (event) => { console.log(event.target); });

---

4. Przeglądarki internetowe i ich znaczenie

4.1 Silniki renderujące

- **Funkcje silników renderujących**:
    - Parsowanie kodu HTML i CSS.
    - Budowanie modelu DOM i CSSOM.
    - Tworzenie drzewa renderowania i wyświetlanie strony.
- **Główne silniki**:
    - **Blink**: Używany przez Google Chrome, Opera, Microsoft Edge.
    - **Gecko**: Używany przez Mozilla Firefox.
    - **WebKit**: Używany przez Safari.

4.2 Standardy webowe

- **Organizacje standaryzujące**:
    - **W3C (World Wide Web Consortium)**: Ustala standardy dla HTML, CSS i innych technologii webowych.
    - **ECMA International**: Standaryzuje JavaScript jako ECMAScript.
- **Znaczenie standardów**:
    - Zapewniają spójność działania stron w różnych przeglądarkach.
    - Ułatwiają rozwój sieci poprzez definiowanie wspólnych specyfikacji.

4.3 Niezgodności między przeglądarkami

- **Przyczyny**:
  - Różne implementacje standardów.
  - Wsparcie dla nowych funkcji w różnym czasie.
- **Konsekwencje**:
  - Elementy mogą wyglądać lub działać inaczej w różnych przeglądarkach.
  - Może to prowadzić do błędów lub nieprawidłowego wyświetlania strony.
- **Strategie radzenia sobie z niezgodnościami**:
  - **Feature detection**: Sprawdzanie, czy przeglądarka obsługuje daną funkcję przed jej użyciem.
  - **Polyfille**: Biblioteki dodające wsparcie dla brakujących funkcji w starszych przeglądarkach.
  - **Progressive enhancement**: Budowanie strony w taki sposób, aby działała w podstawowej formie w każdej przeglądarce, z dodatkowymi funkcjami dla tych, które je obsługują.

4.4 Narzędzia deweloperskie przeglądarek

4.4.1 Debugowanie JavaScript

- **Konsola JavaScript**:
  - Wyświetlanie błędów i ostrzeżeń.
  - Wykonywanie poleceń w czasie rzeczywistym.
- **Debugger**:
  - Ustawianie punktów przerwania (breakpoints).
  - Śledzenie wartości zmiennych i stosu wywołań.

4.4.2 Inspektor elementów

- **Podgląd struktury DOM**:
  - Hierarchia elementów na stronie.
  - Możliwość edycji elementów i atrybutów w locie.
- **Style CSS**:
  - Podgląd i modyfikacja stylów zastosowanych do elementów.
  - Narzędzia do analizy kaskady stylów i dziedziczenia.

4.4.3 Analiza sieci

- **Monitorowanie żądań HTTP**:
  - Lista wszystkich żądań wykonywanych przez stronę.
  - Szczegóły żądań: URL, metoda, status, czas trwania.
- **Optymalizacja**:
  - Identyfikacja zasobów powodujących długie czasy ładowania.
  - Analiza rozmiaru plików i możliwości kompresji.

4.4.4 Emulacja urządzeń i testowanie responsywności

- **Tryb responsywny**:
  - Symulacja różnych rozdzielczości ekranu.
  - Testowanie zachowania strony na urządzeniach mobilnych.
- **Symulacja funkcji urządzeń**:
  - Dotyk, orientacja ekranu, geolokalizacja.

---

5. Narzędzia pracy frontend developera

5.1 Edytory kodu

5.1.1 Visual Studio Code

- **Funkcjonalności**:
  - **IntelliSense**: Inteligentne podpowiedzi kodu.
  - **Debugowanie**: Wbudowane narzędzia do debugowania.
  - **Terminal**: Wbudowany terminal do wykonywania poleceń.
- **Popularne rozszerzenia**:
  - **Emmet**: Skróty kodu do szybkiego tworzenia struktur HTML i CSS.
  - **ESLint**: Narzędzie do analizy statycznej kodu JavaScript pod kątem błędów i zgodności ze standardami.
  - **Prettier**: Automatyczne formatowanie kodu zgodnie z ustalonym stylem.

5.1.2 Atom

- **Cechy charakterystyczne**:
  - **Open Source**: Darmowy i otwarty kod źródłowy.
  - **Pakiety**: Możliwość instalacji tysięcy pakietów rozszerzających funkcjonalność.
- **Funkcje**:
  - **Podzielony widok**: Możliwość pracy na wielu plikach jednocześnie.
  - **Wyszukiwanie i zamiana**: Zaawansowane funkcje wyszukiwania w projekcie.

5.2 Systemy kontroli wersji

5.2.1 Git

- **Podstawowe komendy**:
  - `git init`: Inicjalizacja nowego repozytorium.
  - `git clone`: Klonowanie istniejącego repozytorium.
  - `git status`: Sprawdzenie statusu zmian.
  - `git add`: Dodawanie zmian do indeksu.
  - `git commit`: Zatwierdzanie zmian z komunikatem.
  - `git push`: Wysyłanie zmian do zdalnego repozytorium.
  - `git pull`: Pobieranie i scalanie zmian z zdalnego repozytorium.

- **Praca zespołowa**:
    - **Gałęzie (branches)**: Umożliwiają równoległą pracę nad różnymi funkcjonalnościami bez wpływu na główną wersję kodu.
    - **Scalanie (merging)**: Łączenie zmian z różnych gałęzi.
    - **Rozwiązywanie konfliktów**: Proces ręcznego łączenia sprzecznych zmian w kodzie.

5.2.2 Platformy hostingowe

- **GitHub**:
    - Hostowanie kodu źródłowego.
    - Narzędzia do zarządzania projektami: Issues, Pull Requests.
    - Integracje z innymi narzędziami i usługami.
- **GitLab**:
    - Wbudowane narzędzia CI/CD.
    - Możliwość tworzenia prywatnych repozytoriów bez opłat.
- **Bitbucket**:
    - Obsługa zarówno Git, jak i Mercurial.
    - Integracja z narzędziami Atlassian, takimi jak Jira.

5.3 Konsola deweloperska

5.3.1 Debugowanie i analiza

- **Konsola**:
    - Wykonywanie poleceń JavaScript w czasie rzeczywistym.
    - Wyświetlanie wartości zmiennych i wyników funkcji.
- **Debuger**:
    - Śledzenie wykonania skryptu krok po kroku.
    - Inspekcja wartości zmiennych w różnych punktach programu.

5.3.2 Narzędzia do testowania wydajności

- **Performance**:
    - Nagrywanie i analiza czasu ładowania strony.
    - Identyfikacja wąskich gardeł w wydajności.
- **Memory**:
    - Monitorowanie zużycia pamięci.
    - Wykrywanie i analiza wycieków pamięci.

5.3.3 Inne funkcje

- **Application**:
    - Zarządzanie lokalnymi danymi aplikacji: Local Storage, Session Storage, Cookies.
    - Inspekcja cache i manifestów aplikacji PWA.
- **Security**:
    - Sprawdzanie bezpieczeństwa strony, np. ważności certyfikatów SSL.