



Wykład nr 2

Temat: Wprowadzenie do programowania aplikacji internetowych - historia i rozwój technologii webowych

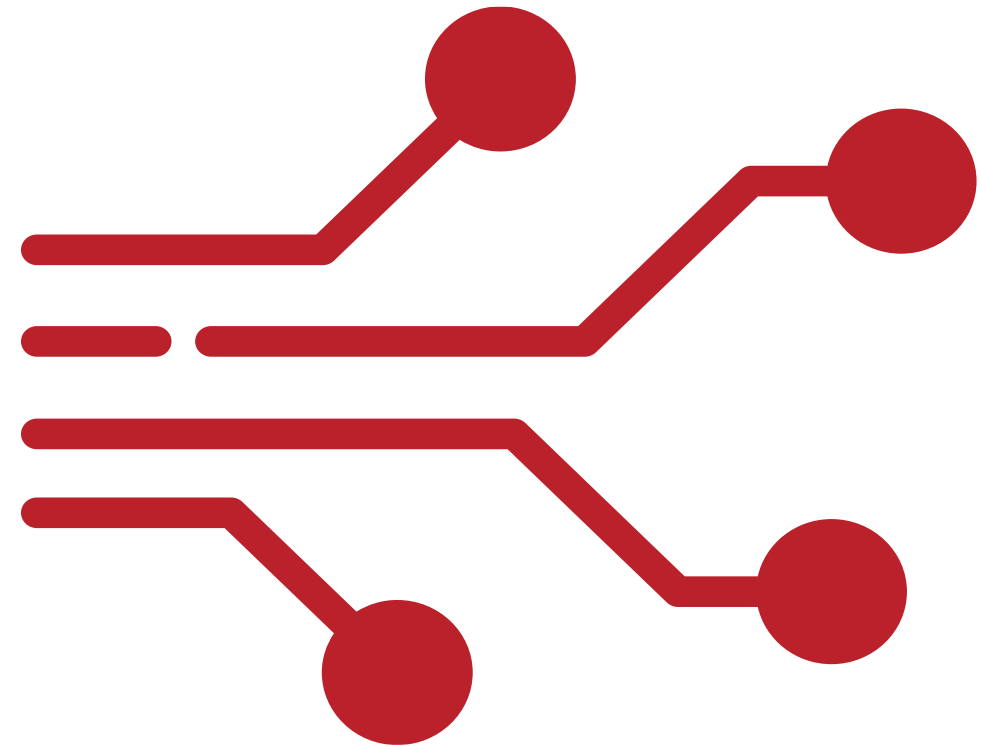
dr hab. inż. Michał Wydra, prof. Uczelni
Katedra Informatyki Stosowanej

Plan prezentacji

- Czym są aplikacje internetowe?
- Ewolucja technologii webowych
- Architektura aplikacji webowych
- Współczesne zastosowania i przyszłość Webu



Czym są aplikacje internetowe?



3

Czym jest programowanie?

Programowanie to proces projektowania i zapisywania algorytmów, czyli logicznych sekwencji działań, które prowadzą do osiągnięcia konkretnego celu. Polega ono na analizie problemu, rozbiciu go na mniejsze, zrozumiałe etapy oraz opisaniu tych etapów w postaci jednoznacznych instrukcji. Instrukcje te są zapisywane w językach programowania, które pełnią rolę pośrednika między sposobem myślenia człowieka a technicznym sposobem działania komputera. Dzięki programowaniu możliwe jest tworzenie aplikacji, stron internetowych, systemów operacyjnych czy gier, a także automatyzowanie powtarzalnych czynności i przetwarzanie dużych ilości danych. W praktyce programowanie łączy w sobie logiczne myślenie, kreatywność oraz umiejętność precyzyjnego formułowania poleceń, tak aby komputer mógł je wykonać dokładnie zgodnie z zamierzeniem autora.

Program i algorytm

Algorytm jest precyzyjnym opisem kolejnych czynności, które należy wykonać, aby rozwiązać określony problem lub osiągnąć zamierzony rezultat. Niezależnie od tego, czy dotyczy obliczeń matematycznych, sortowania danych czy przygotowania potrawy, algorytm zawsze opiera się na logicznej kolejności kroków, które mogą być jednoznacznie zrozumiane i powtórzone.

Program natomiast stanowi praktyczną realizację algorytmu – jest jego zapisem w konkretnym języku programowania, dostosowanym do wymagań komputera. Oznacza to, że ten sam algorytm może zostać zaimplementowany w różnych językach i wciąż prowadzić do tego samego efektu. Porównanie do przepisu kulinarnego i gotowego dania dobrze oddaje tę zależność: algorytm określa, co i w jakiej kolejności należy zrobić, natomiast program jest efektem zastosowania tych instrukcji w praktyce, który komputer potrafi wykonać.

Języki programowania

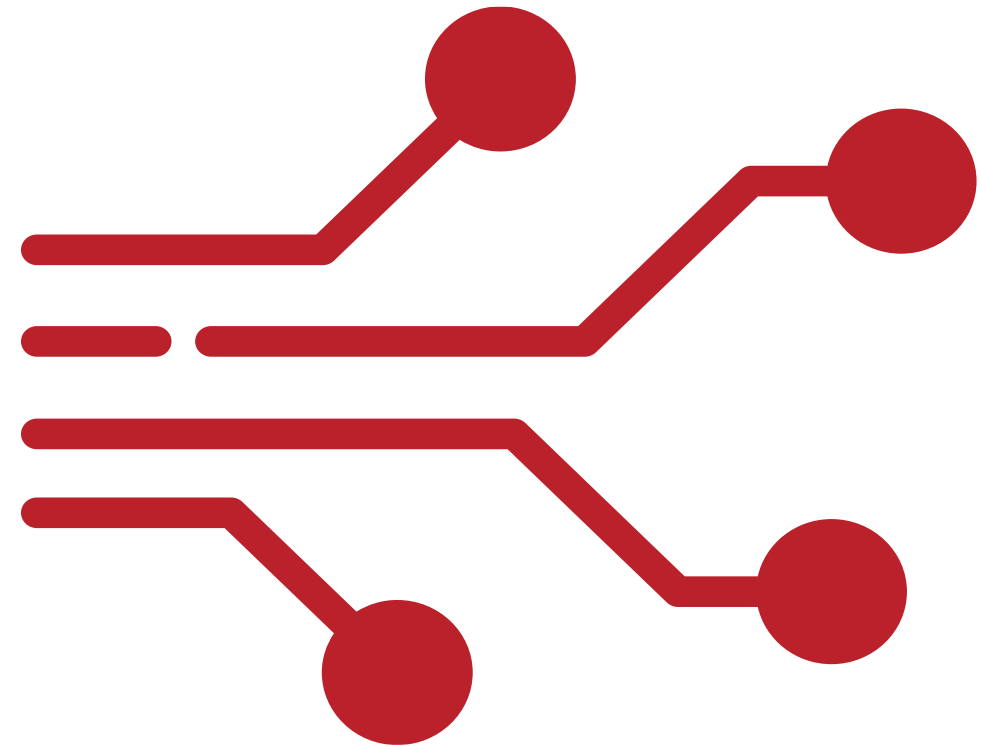
Języki programowania są narzędziami, które umożliwiają człowiekowi przekazywanie komputerowi poleceń w sposób dla niego zrozumiały. Ponieważ komputer operuje wyłącznie na danych binarnych, języki programowania pełnią rolę pośrednika – pozwalają programiście opisywać algorytmy i zasady działania systemów w formie zbliżonej do języka naturalnego, a następnie są tłumaczone na postać możliwą do wykonania przez maszynę. Dzięki temu tworzenie oprogramowania staje się bardziej czytelne, uporządkowane i dostępne dla ludzi.

Ze względu na poziom abstrakcji języki programowania dzieli się m.in. na:

- **niskopoziomowe**, takie jak assembler, są bardzo bliskie sprzętowi i pozwalają na bezpośrednią kontrolę nad działaniem procesora oraz pamięci, jednak ich użycie jest trudniejsze i bardziej czasochłonne
- **wysokopoziomowe**, takie jak JavaScript, oferują większą abstrakcję, prostszą składnię i bogate biblioteki, co ułatwia pisanie, czytanie i utrzymanie kodu. Dzięki nim programista może skupić się na logice rozwiązania problemu, a nie na szczegółach technicznych działania sprzętu

6

Ewolucja technologii webowych



Kompilacja i interpretacja

Kompilacja i interpretacja to dwa różne sposoby przetwarzania kodu źródłowego na instrukcje zrozumiałe dla komputera. W przypadku **języków kompilowanych** cały program jest najpierw tłumaczony przez kompilator do postaci maszynowej, czyli pliku wykonywalnego, zanim zostanie uruchomiony. Oznacza to, że ewentualne błędy składniowe są wykrywane już na etapie kompilacji, a sam program po poprawnym skompilowaniu działa zazwyczaj szybciej, ponieważ komputer wykonuje gotowy kod maszynowy. Przykładami takich języków są C oraz C++.

Języki interpretowane działają w inny sposób – kod nie jest wcześniej tłumaczony w całości, lecz wykonywany na bieżąco, linia po linii, przez interpreter w trakcie działania programu. Dzięki temu programista może szybciej testować zmiany w kodzie, ponieważ nie ma potrzeby każdorazowej kompilacji całego programu. Wadą tego podejścia może być wolniejsze działanie aplikacji oraz wykrywanie części błędów dopiero w momencie wykonywania kodu. Do języków interpretowanych zalicza się między innymi JavaScript, który jest powszechnie wykorzystywany w aplikacjach internetowych.

8

Początki Internetu

Początki Internetu wiązały się z bardzo prostą formą prezentowania informacji. Pierwsze strony internetowe miały charakter statyczny i były tworzone wyłącznie w języku HTML, który służył do opisu struktury dokumentu oraz rozmieszczenia tekstu i obrazów. Strony te pełniły głównie funkcję informacyjną – przypominały cyfrowe dokumenty lub broszury, które użytkownik mógł jedynie odczytać.

W tamtym okresie nie istniały mechanizmy umożliwiające aktywną interakcję z użytkownikiem, takie jak formularze, animacje czy dynamicznie zmieniające się treści. Każda zmiana zawartości strony wymagała ręcznej edycji pliku przez autora i ponownego jej opublikowania. Internet był więc medium jednokierunkowym, w którym użytkownik pełnił rolę biernego odbiorcy informacji, a nie aktywnego uczestnika, jak ma to miejsce we współczesnych aplikacjach internetowych.

Moja strona

[Strona Główna](#) | [O mnie](#)

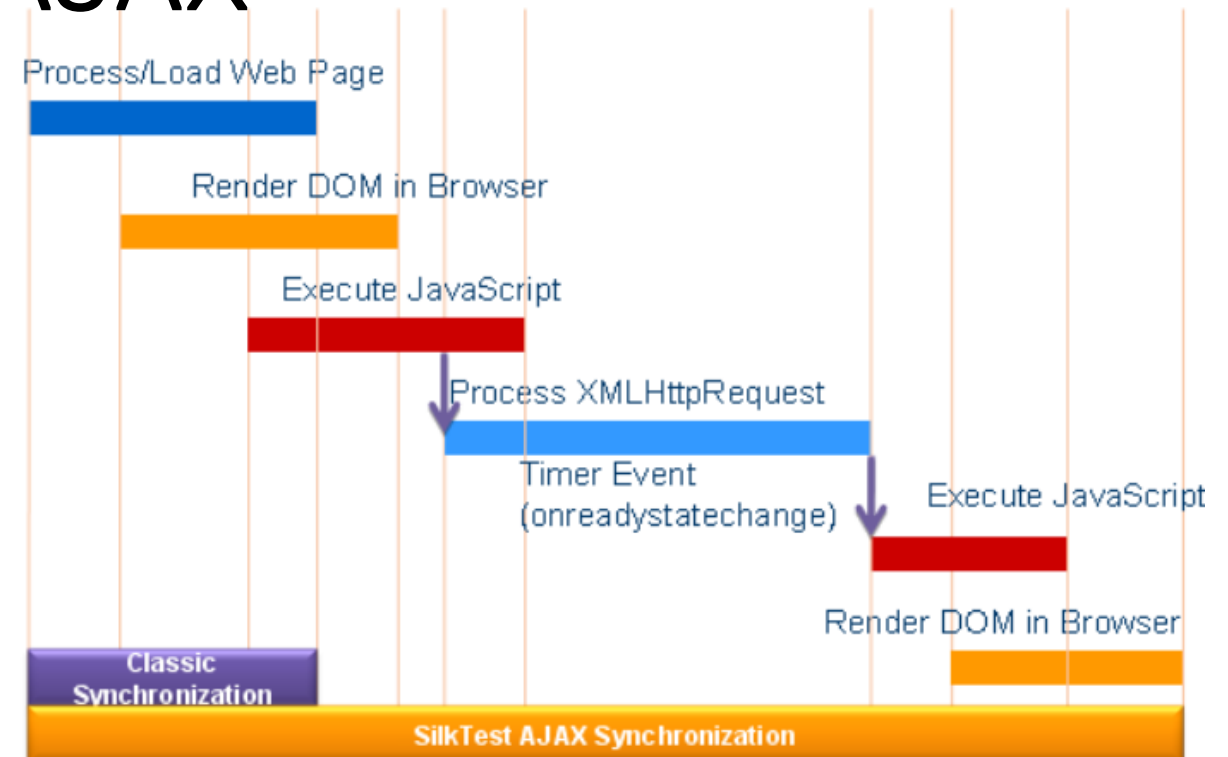
Witaj na mojej stronie

Lorem ipsum dolor sit amet consectetur adipisicing elit. Dolor atque eveniet id quasi autem quis porro molestiae dolorem laudantium qui corrupti, perspiciatis exercitationem blanditiis optio aliquam dolore necessitatibus aliquid recusandae placeat ducimus libero tenetur doloribus. Eligendi quibusdam architecto quisquam a dolorum temporibus illum quod quo reprehenderit, perspiciatis provident nemo quam harum laborum ipsum, aut nobis voluptatibus accusantium assumenda eos ullam alias repellendus! Reprehenderit optio expedita quo voluptate ad magni eos ratione temporibus architecto, necessitatibus assumenda. Consequatur amet, quod quibusdam quas corrupti facilis soluta nemo, culpa consectetur repellat mollitia sed quo quam provident fugit, tempore vero eaque! Sunt provident quisquam iure, similique obcaecati voluptas distinctio quidem autem corrupti dolore nihil tempora consequuntur esse nulla nostrum ea maiores eum eius. Sapiente provident corrupti non distinctio dicta ipsum. Nihil, a quae libero fugit aliquid odit! Neque voluptates, architecto illo quod dolore impedit? Possimus a corporis quia! Dolore, eligendi consequatur ea explicabo illo quidem quos est, natus provident error inventore consectetur nesciunt eum, culpa corporis deserunt vel adipisci quia sint omnis recusandae maxime unde! Totam fuga neque mollitia sed sunt debitis sequi id ratione! Officia quas optio eligendi harum assumenda ad distinctio nulla delectus voluptatibus veritatis placeat debitis at voluptates, ratione nam, culpa possimus ullam! Eligendi, laboriosam, tempora vel quam repellat ipsa necessitatibus totam laudantium natus accusantium, laborum numquam quod sequi libero enim voluptatem autem itaque! Quam laborum sapiente maxime placeat consequuntur, cupiditate, eos molestiae nihil odio tempora perspiciatis culpa quas. Voluptates rerum harum, nam voluptatum dolore, recusandae, a aliquam natus illum minus assumenda. Dolore, libero? Necessitatibus sit praesentium vel suscipit preferendis obcaecati quasi. Inventore, quasi repellendus eligendi magnam facere mollitia fugiat perspiciatis quaerat voluptate dolorum distinctio nisi, quisquam, nesciunt sed aliquid quia. Repellat nobis natus minima beatae debitis ab sunt, quia necessitatibus itaque voluptatum iure quod maiores harum! Animi odit aliquam eos a?

Rys. 1. Przykład prostej, statycznej strony opartej na HTML [źródło: https://webporady.pl/prosta-strona-internetowa-w-html/#google_vignette]

Dynamiczne strony i AJAX

Dynamiczne strony internetowe powstały jako odpowiedź na ograniczenia statycznych witryn. Dzięki technologii AJAX przeglądarka może komunikować się z serwerem w tle, bez konieczności przeładowywania całej strony. Umożliwia to asynchroniczne pobieranie danych oraz aktualizowanie tylko wybranych elementów strony, takich jak listy czy formularze. W efekcie aplikacje internetowe stały się szybsze, bardziej responsywne i wygodniejsze w obsłudze dla użytkownika.



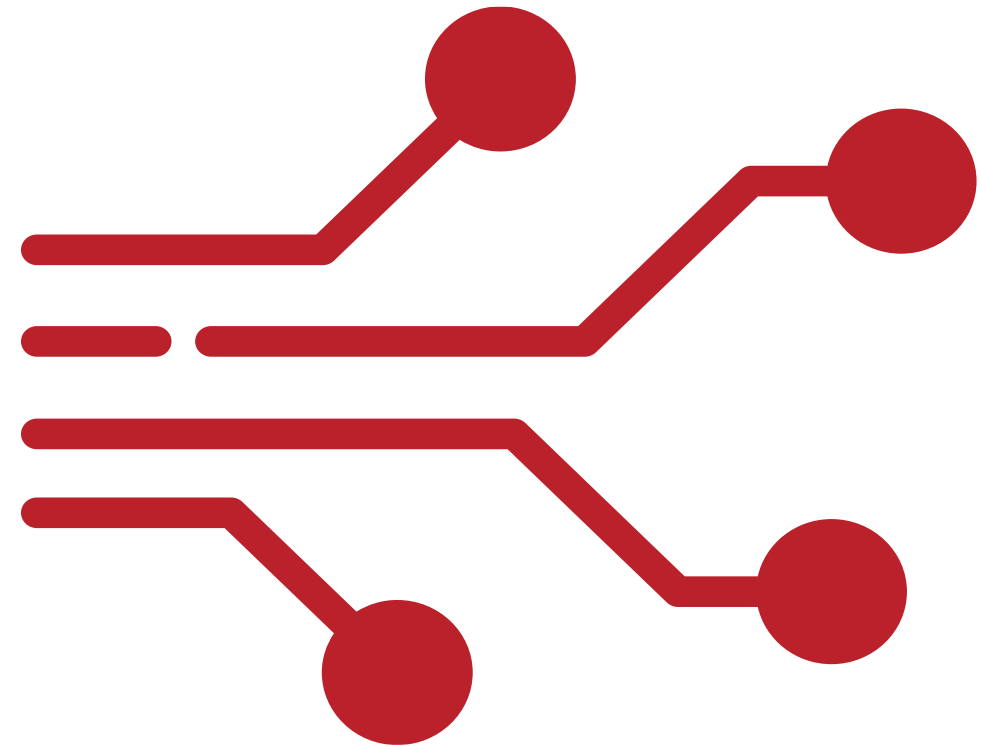
Rys. 2. Przepływ przetwarzania AJAX [źródło: 11
<https://www.microfocus.com/documentation/silk-performer/195/en/silkperformer-195-webhelp-en/SILKPERF-7ED3AAA1-BDWLT-AJAXPROCESSFLOW-CON.html>]

Strona internetowa a aplikacja webowa

Strona internetowa służy głównie do prezentowania informacji i ma zazwyczaj charakter statyczny lub tylko w niewielkim stopniu dynamiczny. Jej podstawowym celem jest przekazanie treści użytkownikowi, który pełni rolę odbiorcy i ma ograniczone możliwości interakcji. Takie strony często zawierają teksty, obrazy oraz proste elementy nawigacyjne.

Aplikacja webowa jest znacznie bardziej rozbudowana i interaktywna. Działa w przeglądarce, ale swoim zachowaniem przypomina klasyczny program desktopowy, umożliwiając użytkownikowi wykonywanie konkretnych operacji, takich jak edycja danych, zarządzanie plikami czy praca w czasie rzeczywistym. Dzięki temu aplikacje webowe oferują większą funkcjonalność i aktywne zaangażowanie użytkownika.

Architektura aplikacji webowych



13

Rozwój frontendu

Frontend to część aplikacji internetowej, która działa bezpośrednio w przeglądarce użytkownika i odpowiada za to, co użytkownik widzi oraz w jaki sposób może wchodzić w interakcję ze stroną. Na początku rozwoju Internetu frontend był ograniczony głównie do statycznych dokumentów, jednak wraz z rozwojem technologii stał się dynamiczny i coraz bardziej rozbudowany. Jego zadaniem jest nie tylko prezentacja treści, ale także zapewnienie intuicyjnej i wygodnej obsługi aplikacji.

Podstawą frontendu są trzy kluczowe technologie. HTML odpowiada za strukturę strony, czyli układ elementów i ich znaczenie. CSS służy do stylizacji, definiując wygląd strony, kolory, czcionki oraz responsywność na różnych urządzeniach. JavaScript natomiast umożliwia implementację logiki aplikacji po stronie klienta, obsługę zdarzeń oraz dynamiczne modyfikowanie treści. Wspólne wykorzystanie tych technologii pozwoliło na rozwój nowoczesnych, interaktywnych aplikacji webowych.

14

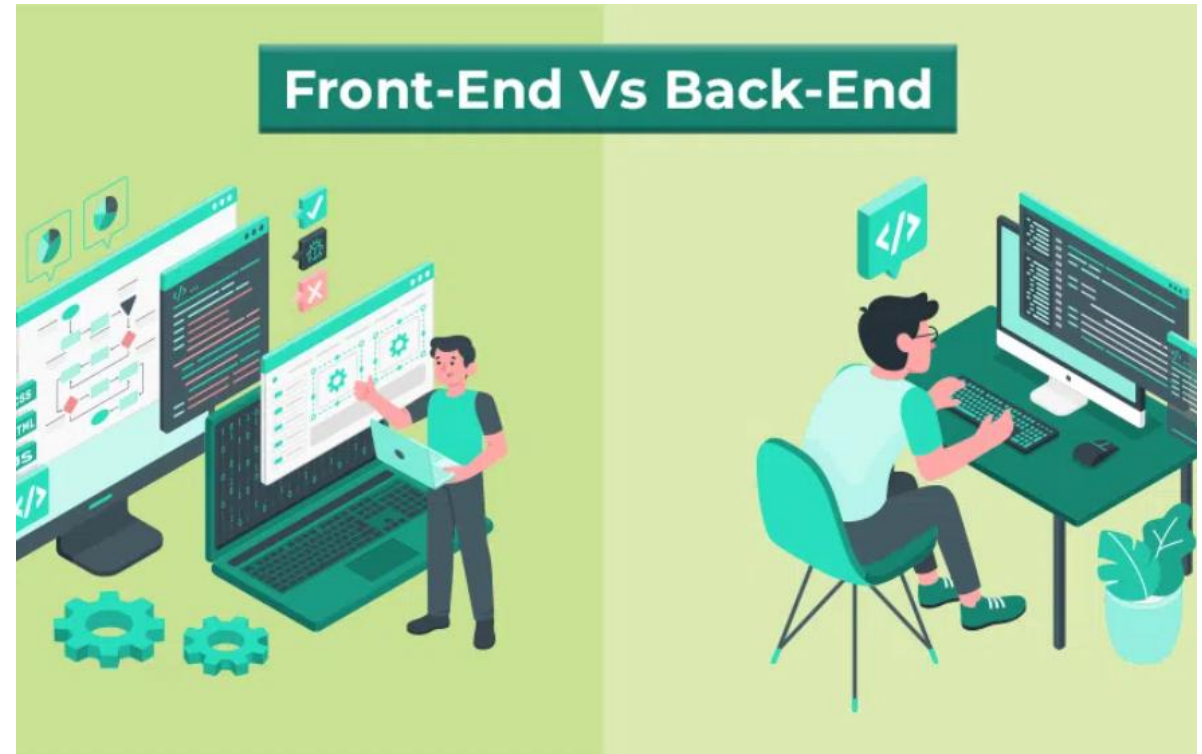
Frontend a backend

Frontend i backend to dwa podstawowe elementy nowoczesnych aplikacji webowych.

Frontend odpowiada za interfejs użytkownika i wszystkie elementy widoczne w przeglądarce, umożliwiając interakcję z aplikacją.

Backend działa po stronie serwera i zajmuje się logiką aplikacji, przetwarzaniem danych oraz obsługą baz danych.

Oba obszary są ze sobą ściśle powiązane i komunikują się, aby zapewnić poprawne i dynamiczne działanie aplikacji.

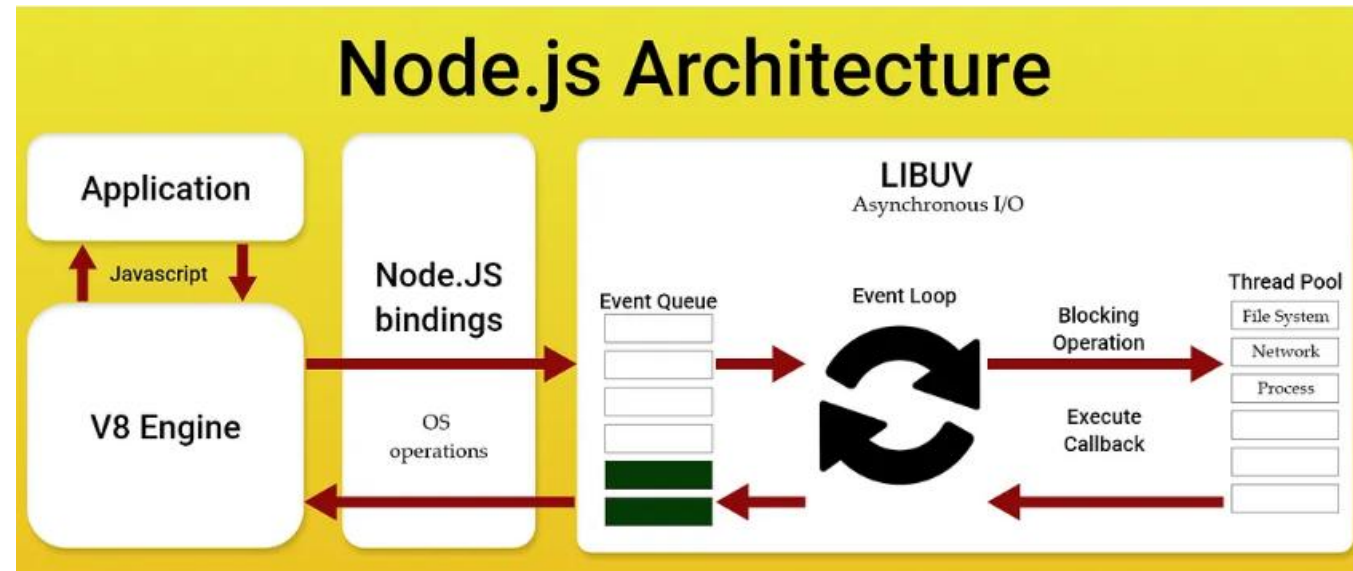


Rys. 3. Frontend vs backend [źródło: <https://wardriving.pl/frontend-vs-backend-roznice-i-podobienstwa/>]

15

JavaScript po stronie serwera

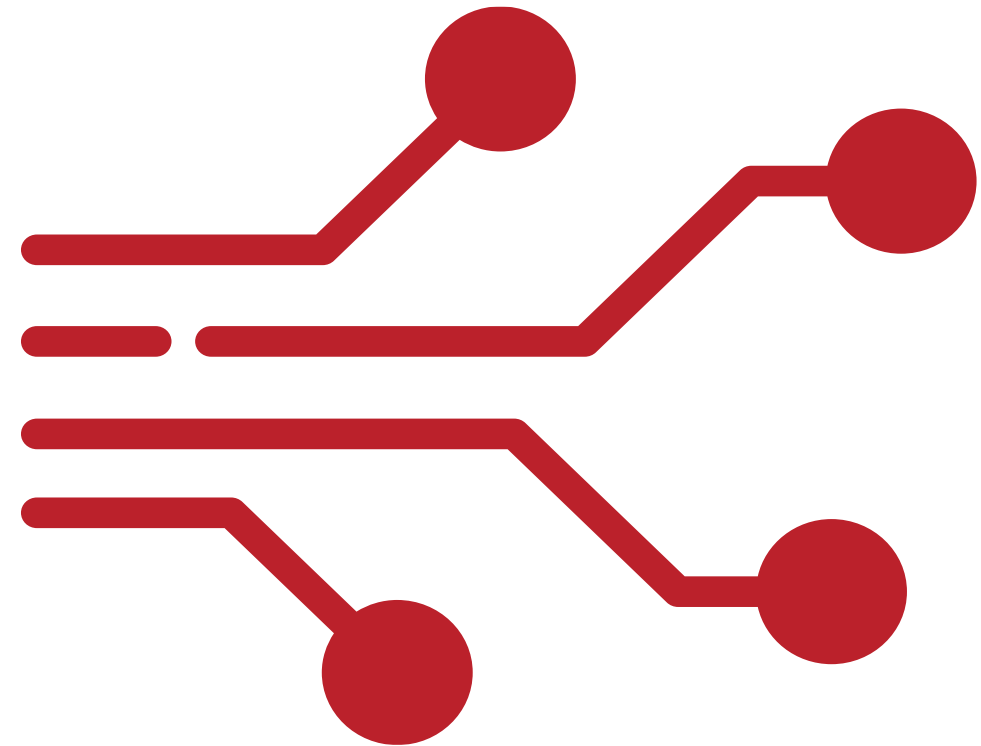
JavaScript początkowo był wykorzystywany wyłącznie po stronie przeglądarki, jednak wraz z pojawieniem się środowiska Node.js zyskał możliwość działania także po stronie serwera. Dzięki temu JavaScript może obsługiwać logikę aplikacji, komunikację z bazami danych oraz obsługę żądań sieciowych. Umożliwia to tworzenie pełnych aplikacji webowych z wykorzystaniem jednego języka programowania zarówno w frontendzie, jak i backendzie, co upraszcza rozwój i utrzymanie oprogramowania.



Rys. 4. Node.js Architecture [źródło: <https://medium.com/@abeythilakeudara3/nodejs-architecture-42a1d0efad8f>]

16

Współczesne zastosowania i przyszłość Webu



17

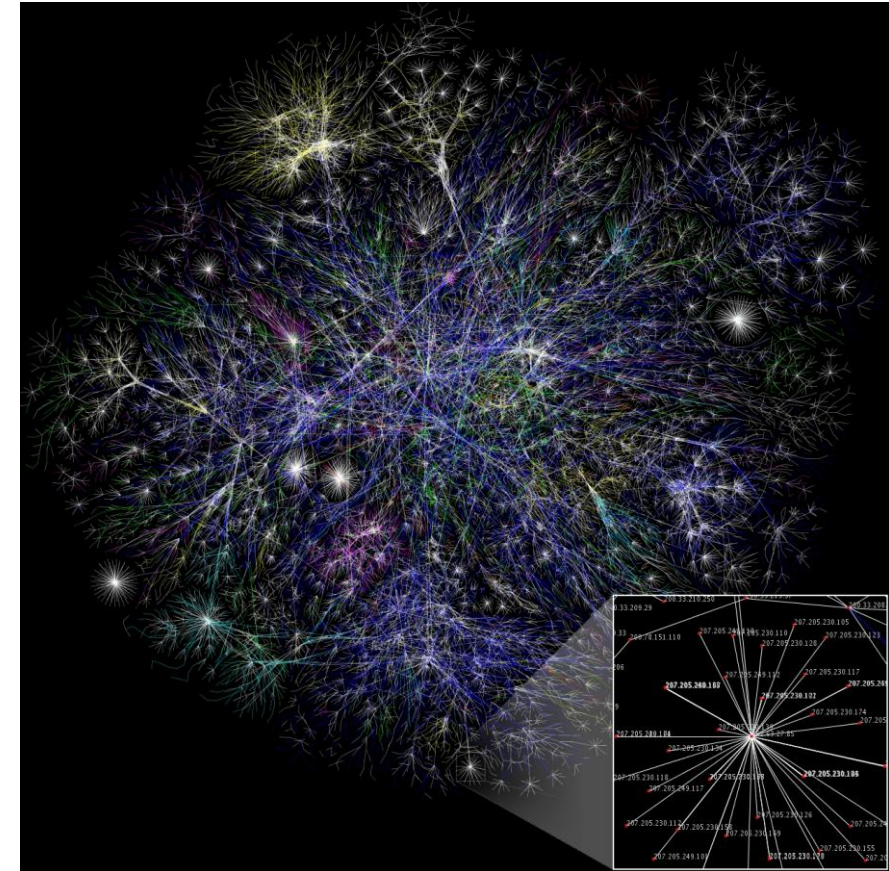
Aplikacje mobilne i desktopowe

JavaScript nie jest ograniczony wyłącznie do aplikacji webowych, ale znajduje zastosowanie także w tworzeniu aplikacji mobilnych i desktopowych. Dzięki frameworkowi React Native możliwe jest budowanie aplikacji mobilnych na systemy Android i iOS z wykorzystaniem jednego wspólnego kodu w JavaScriptcie. Pozwala to znacząco skrócić czas tworzenia aplikacji i ułatwia ich utrzymanie.

Również aplikacje desktopowe mogą być tworzone przy użyciu JavaScriptu, m.in. za pomocą technologii takich jak Electron czy NW.js. Umożliwiają one budowanie programów działających na różnych systemach operacyjnych z wykorzystaniem technologii webowych. Dzięki temu JavaScript stał się językiem uniwersalnym, wykorzystywanym w wielu typach aplikacji i na różnych platformach.

Internet Rzeczy (IoT)

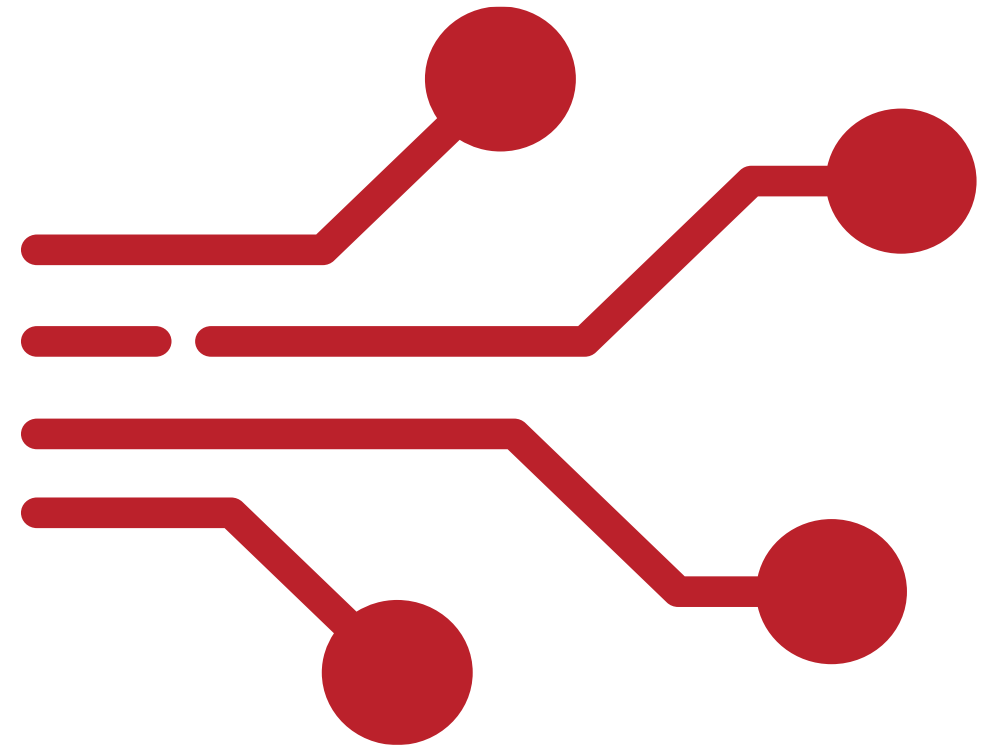
Internet Rzeczy (IoT) to koncepcja łączenia urządzeń z siecią w celu wymiany danych i automatyzacji działań. JavaScript jest wykorzystywany w inteligentnych urządzeniach, czujnikach oraz systemach smart home, gdzie umożliwia sterowanie sprzętem i obsługę komunikacji. Dzięki temu możliwa jest analiza danych w czasie rzeczywistym oraz szybkie reagowanie systemu na zmieniające się warunki.



Rys. 5. Frontend vs backend [źródło:
https://pl.wikipedia.org/wiki/Plik:Internet_map_1024.jpg]

19

Podsumowanie



20

Technologie webowe przeszły długą i dynamiczną drogę rozwoju – od prostych, statycznych dokumentów HTML do rozbudowanych, interaktywnych aplikacji działających w przeglądarkach, na urządzeniach mobilnych i komputerach desktopowych. Wraz z tą ewolucją zmieniała się rola języków programowania oraz sposobów tworzenia aplikacji. JavaScript odegrał w tym procesie kluczową rolę, stopniowo rozszerzając swoje zastosowanie z prostych skryptów po pełnoprawny język wykorzystywany zarówno w frontendzie, jak i backendzie. Obecnie stanowi on jeden z najważniejszych fundamentów nowoczesnego programowania aplikacji internetowych i wieloplatformowych.

Literatura:

- [1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.
- [2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.
- [3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.
- [4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.
- [5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Dziękuję za uwagę

Wykład nr 3

Temat: Struktura aplikacji internetowej. Podział aplikacji na backend i frontend

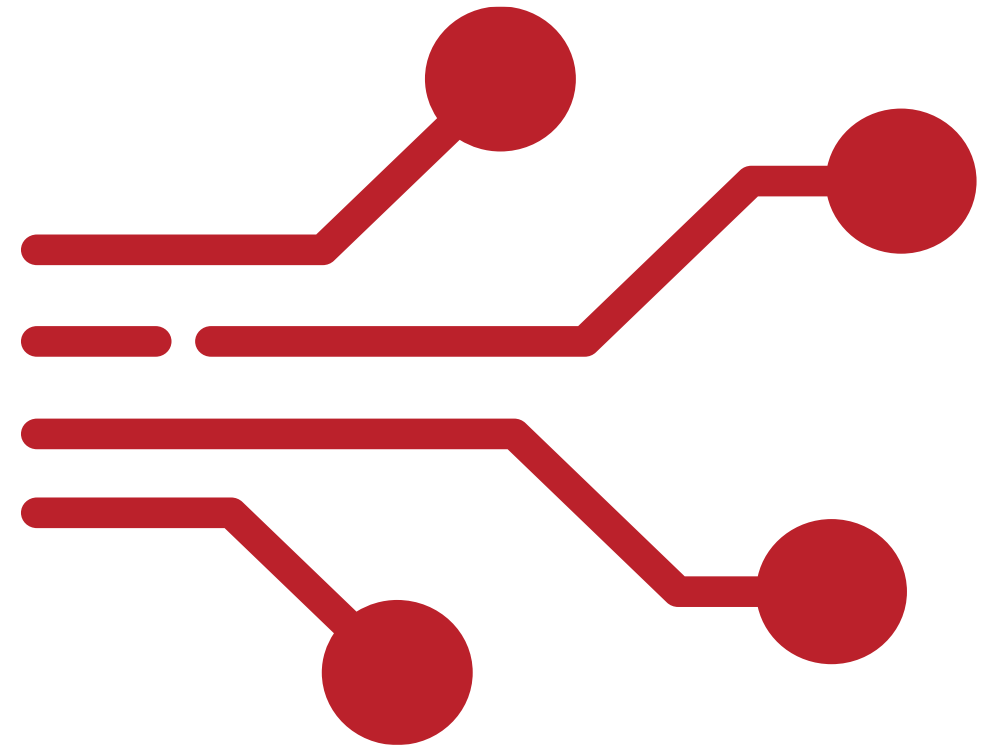
dr hab. inż. Michał Wydra, prof. Uczelni
Katedra Informatyki Stosowanej

Plan prezentacji

- Czym jest aplikacja internetowa i jak działa
- Podział aplikacji na frontend i backend
- Warstwowa struktura aplikacji webowej
- Architektura klient–serwer i jej warianty
- Nowoczesne podejścia architektoniczne



Czym jest aplikacja internetowa i jak działa



3

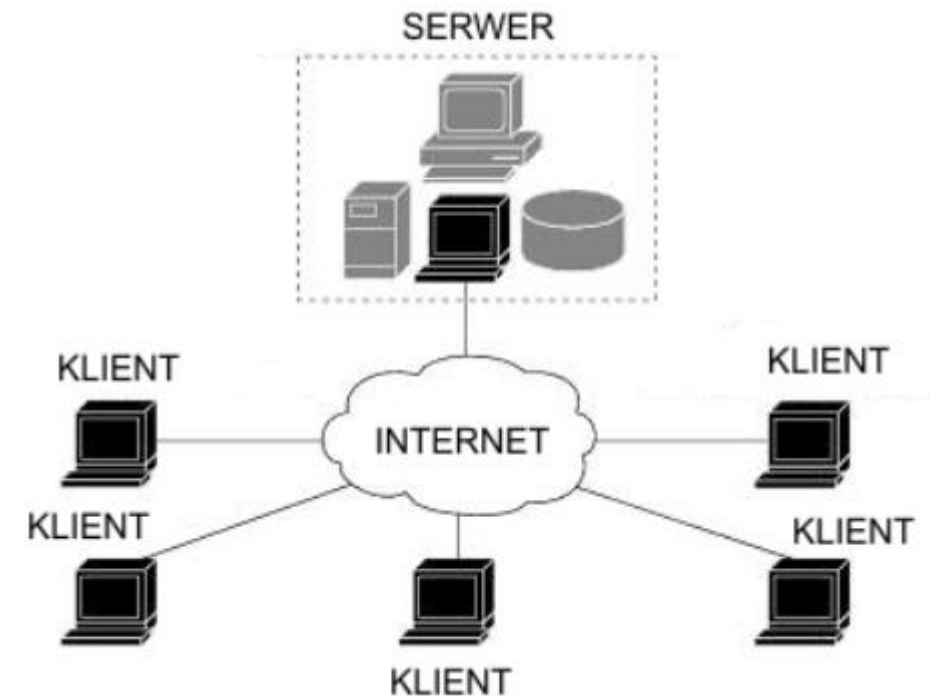
Aplikacja internetowa

Aplikacja internetowa to rozbudowany system informatyczny dostępny za pośrednictwem przeglądarki internetowej, który umożliwia użytkownikowi aktywną interakcję z danymi oraz funkcjonalnościami systemu w czasie rzeczywistym. W przeciwieństwie do tradycyjnych stron WWW, które pełnią głównie funkcję informacyjną, aplikacje webowe pozwalają na wykonywanie konkretnych operacji, takich jak wprowadzanie, edytowanie czy przetwarzanie danych.

Charakterystyczną cechą aplikacji internetowych jest obecność logiki biznesowej, obsługa formularzy oraz mechanizmy walidacji danych, które zapewniają poprawność i bezpieczeństwo wprowadzanych informacji. Dodatkowo dynamiczne aktualizacje treści, realizowane bez konieczności przeładowywania strony, sprawiają, że korzystanie z aplikacji jest płynne i wygodne, a jej działanie zbliżone do klasycznych programów desktopowych.

Model klient-serwer

Model klient-serwer jest podstawą działania aplikacji internetowych. Klientem jest zazwyczaj przeglądarka internetowa, która wyświetla interfejs użytkownika i wysyła żądania do systemu. Serwer odpowiada za logikę aplikacji oraz udostępnianie danych. Komunikacja pomiędzy klientem a serwerem odbywa się za pomocą protokołu HTTP, który umożliwia wymianę informacji między tymi elementami.



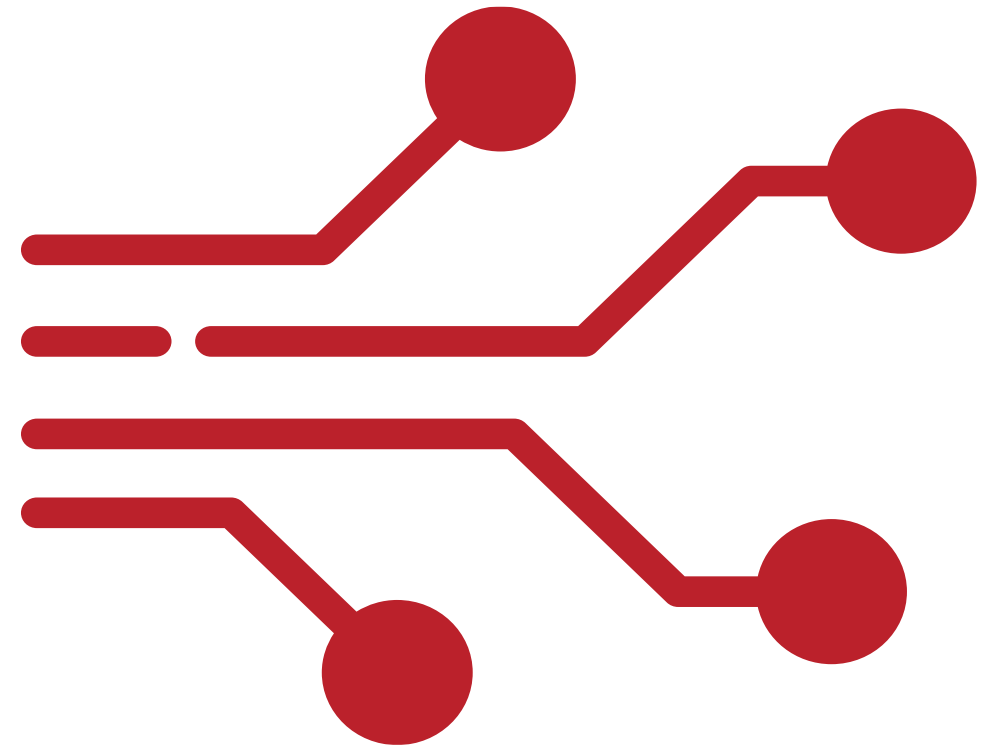
Rys. 1. Model klient-serwer [źródło:
https://eduinf.waw.pl/inf/utills/010_2010/0111.php]

Cykl żądanie-odpowieź

Cykl żądanie-odpowieź opisuje podstawowy sposób działania aplikacji internetowych i wymiany danych pomiędzy przeglądarką a serwerem. Proces rozpoczyna się w momencie, gdy użytkownik wpisuje adres URL w przeglądarce, co powoduje wysłanie żądania HTTP do serwera. Serwer odbiera to żądanie, analizuje je i wykonuje odpowiednie operacje, takie jak pobranie danych lub uruchomienie logiki aplikacji.

Po przetworzeniu żądania serwer odsyła odpowiedź, która może zawierać dokument HTML, dane w formacie JSON lub inne zasoby. Następnie przeglądarka interpretuje otrzymaną odpowiedź i renderuje wynik, wyświetlając użytkownikowi stronę lub aktualizując jej zawartość. Dzięki temu cyklowi użytkownik może wchodzić w interakcję z aplikacją internetową w sposób płynny i zrozumiały.

Podział aplikacji na frontend i backend



Frontend

Frontend to część aplikacji internetowej działająca po stronie klienta, odpowiedzialna za interfejs użytkownika, prezentację danych oraz obsługę interakcji. To ona decyduje o wyglądzie i sposobie korzystania z aplikacji. Do tworzenia frontendu wykorzystuje się głównie HTML do budowy struktury, CSS do stylizacji oraz JavaScript do realizacji logiki interakcji i dynamicznego działania strony.

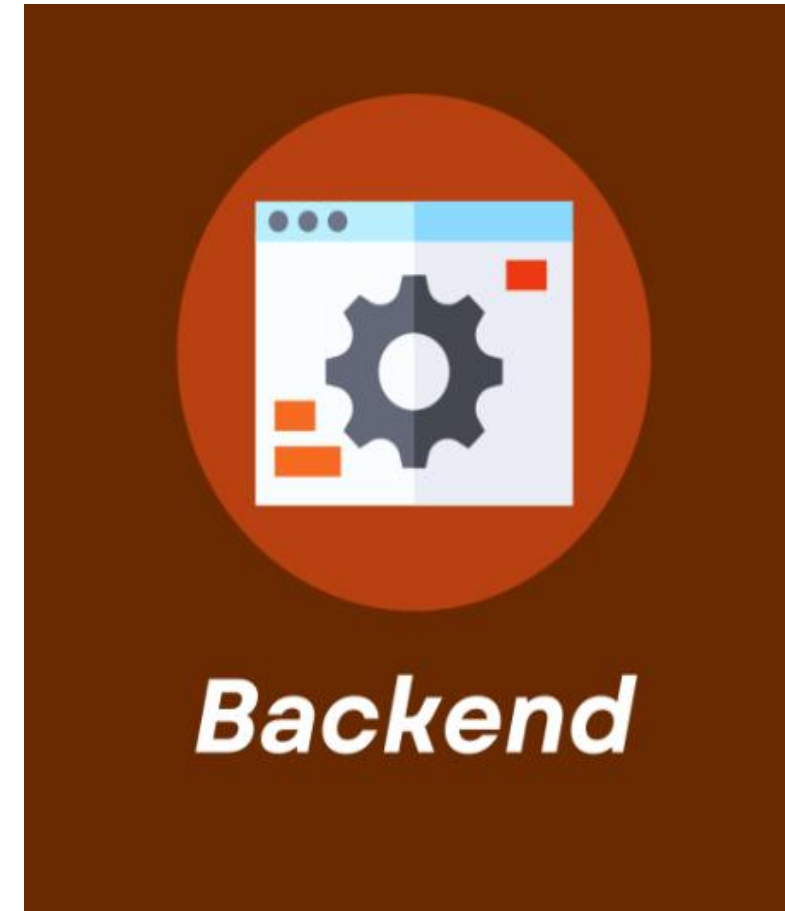


8

Rys. 2. Frontend [źródło:
<https://www.lcusale.shop/?path=page/ggitem&ggpid=2292419>]

Backend

Backend to część aplikacji internetowej działająca po stronie serwera, niewidoczna bezpośrednio dla użytkownika. Odpowiada za realizację logiki biznesowej, przetwarzanie danych oraz komunikację z bazą danych. Do jego zadań należy również zapewnienie bezpieczeństwa systemu, w tym obsługa autoryzacji i kontroli dostępu. Dzięki backendowi aplikacja może poprawnie reagować na działania użytkownika i przechowywać oraz udostępniać dane w bezpieczny sposób.



Rys. 3. Backend [źródło:
<https://www.lcusale.shop/?path=page/ggitem&ggpid=2292419>]

Frontend vs Backend

Frontend:

- działa w przeglądarce,
- skupia się na użytkowniku,
- reaguje na akcje interfejsu.

Backend:

- działa na serwerze,
- przetwarza dane,
- obsługuje logikę aplikacji.

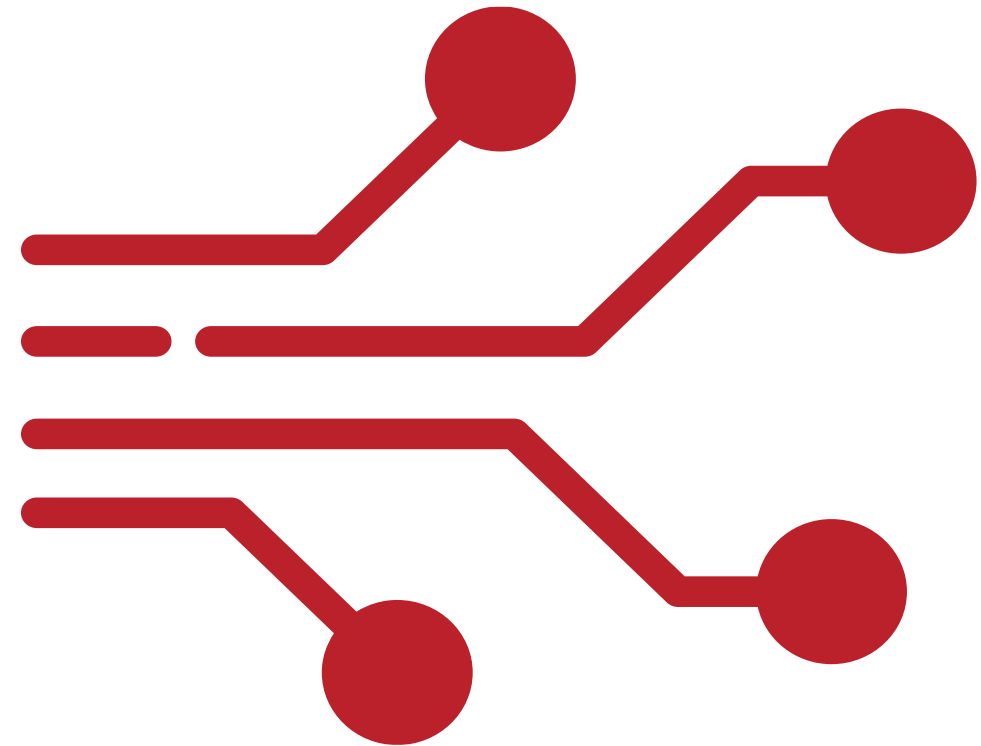
**Obie części muszą ze sobą
współpracować.**

Aplikacje statyczne i dynamiczne

Aplikacje statyczne i dynamiczne różnią się sposobem generowania i udostępniania treści użytkownikowi. W przypadku aplikacji statycznych serwer przechowuje i udostępnia gotowe pliki, takie jak HTML, CSS czy obrazy, które są zawsze takie same niezależnie od użytkownika. Tego typu aplikacje są proste w budowie, szybkie w działaniu, ale oferują bardzo ograniczoną interakcję.

Aplikacje dynamiczne działają w inny sposób – serwer generuje treść w czasie rzeczywistym w odpowiedzi na żądania użytkownika. Oznacza to, że wyświetlane dane mogą się zmieniać w zależności od kontekstu, na przykład zalogowanego użytkownika, przesłanych formularzy czy danych zapisanych w bazie. Dzięki temu dynamiczne aplikacje umożliwiają personalizację, obsługę procesów biznesowych i interakcję w czasie rzeczywistym, co sprawia, że stanowią podstawę nowoczesnych systemów webowych.

Warstwowa struktura aplikacji webowej



12

Architektura warstwowa

Architektura warstwowa polega na podziale aplikacji na logiczne warstwy, z których każda odpowiada za inny aspekt działania systemu, na przykład prezentację, logikę biznesową czy dostęp do danych. Taki podział pozwala na lepsze uporządkowanie kodu i wyraźne oddzielenie odpowiedzialności poszczególnych elementów aplikacji.

Dzięki architekturze warstwowej kod staje się bardziej czytelny i łatwiejszy w utrzymaniu, a poszczególne warstwy można testować niezależnie od siebie. Umożliwia to również rozwój lub modyfikację jednej części systemu bez konieczności ingerowania w pozostałe, co jest szczególnie istotne w większych i długoterminowych projektach webowych.

Warstwa prezentacji

Warstwa prezentacji odpowiada za:

- wygląd interfejsu,
- sposób prezentacji danych,
- obsługę interakcji użytkownika.

W aplikacjach webowych realizowana jest głównie przez frontend.



Rys. 4. Architektura warstwowa aplikacji [źródło: <https://nullpointerexception.pl/architektura-warstwowa-sposob-na-organizacje-kodu/>]

14

Warstwa logiki biznesowej

Warstwa logiki biznesowej:

- realizuje zasady działania aplikacji,
- przetwarza dane,
- podejmuje decyzje logiczne.

Jest kluczowym elementem backendu.



Rys. 4. Architektura warstwowa aplikacji [źródło: <https://nullpointerexception.pl/architektura-warstwowa-sposob-na-organizacje-kodu/>]

15

Warstwa danych

Warstwa danych:

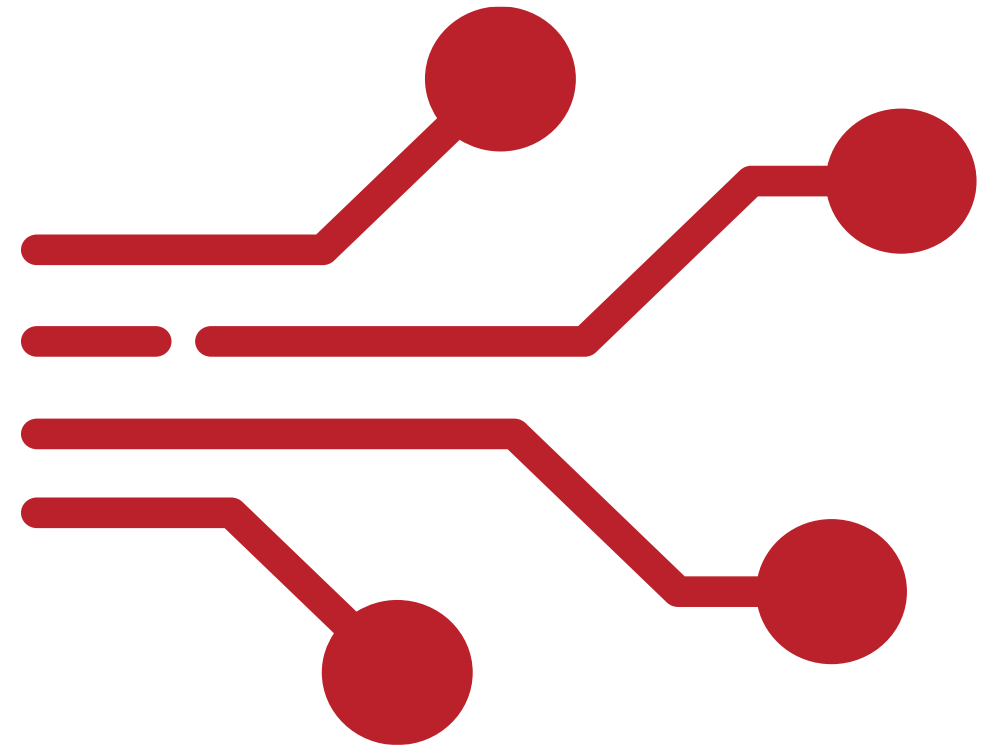
- odpowiada za zapis i odczyt danych,
- komunikuje się z bazą danych,
- izoluje logikę aplikacji od szczegółów przechowywania danych.



Rys. 4. Architektura warstwowa aplikacji [źródło: <https://nullpointerexception.pl/architektura-warstwowa-sposob-na-organizacje-kodu/>]

16

Architektura klient-serwer i jej warianty



17

Architektura dwuwarstwowa

Architektura dwuwarstwowa, nazywana również modelem client–server, polega na podziale aplikacji na dwie główne części. Warstwą kliencką jest frontend, który działa w przeglądarce użytkownika i odpowiada za interfejs oraz interakcję z systemem. Warstwą serwerową jest backend, który realizuje logikę aplikacji, przetwarza dane i komunikuje się z bazą danych.

Jest to podstawowy i najczęściej spotykany model aplikacji webowych, szczególnie w mniejszych i średnich systemach. Prosta struktura architektury dwuwarstwowej ułatwia zrozumienie działania aplikacji oraz jej rozwój, dlatego stanowi punkt wyjścia do nauki bardziej zaawansowanych architektur.

Architektura wielowarstwowa

Architektura wielowarstwowa, określana także jako architektura N-tier, stanowi rozwinięcie klasycznego modelu client–server poprzez wprowadzenie dodatkowych, wyspecjalizowanych warstw. Oprócz warstwy klienckiej i serwerowej system może zostać podzielony m.in. na warstwę usług, odpowiedzialną za logikę biznesową, warstwę API, która obsługuje komunikację między komponentami, oraz warstwę danych, zajmującą się przechowywaniem i dostępem do informacji.

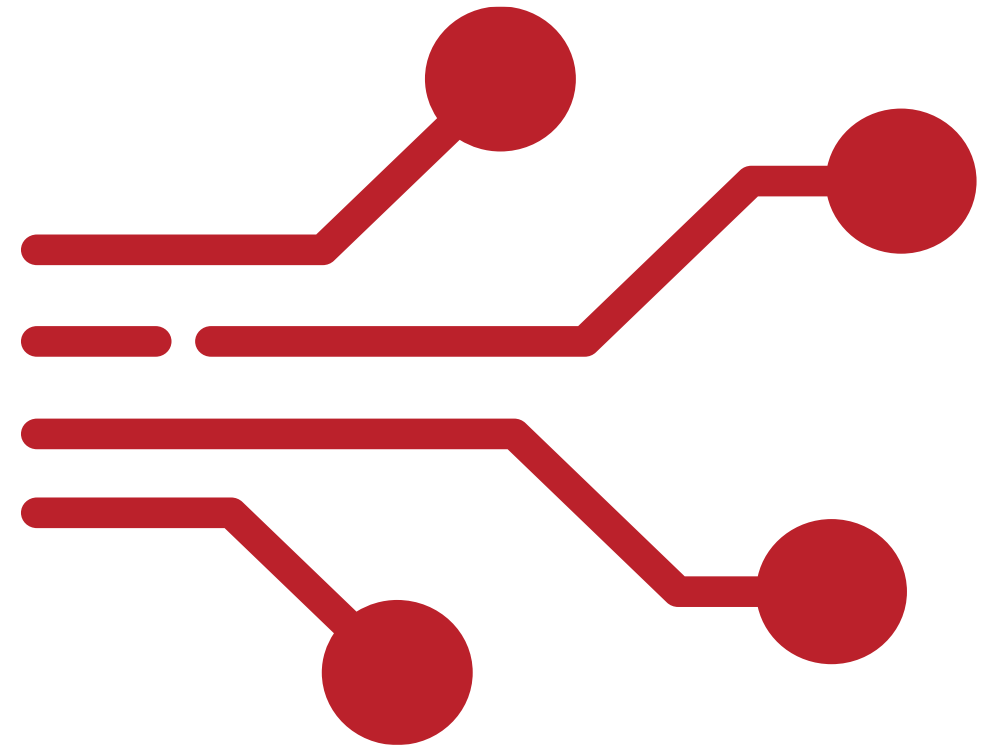
Taki podział pozwala na lepszą organizację kodu, większą elastyczność oraz łatwiejsze skalowanie aplikacji. Architektura N-tier jest szczególnie często stosowana w dużych i złożonych systemach, gdzie istotne są niezawodność, możliwość rozwoju oraz praca wielu zespołów nad różnymi elementami aplikacji jednocześnie.

Single Page Applications

Single Page Applications (SPA) to nowoczesny model tworzenia aplikacji webowych, w którym cała aplikacja działa w obrębie jednej strony ładowanej do przeglądarki. Interfejs użytkownika jest generowany po stronie klienta, a jego zmiany odbywają się dynamicznie, bez konieczności ponownego przeładowywania strony. Dzięki temu korzystanie z aplikacji jest płynne i szybkie, a doświadczenie użytkownika zbliżone do pracy z aplikacją desktopową.

W architekturze SPA backend pełni głównie rolę dostawcy danych i udostępnia je za pomocą interfejsów API. Komunikacja pomiędzy frontendem a backendem odbywa się asynchronicznie, najczęściej w formacie JSON, co pozwala na bieżące pobieranie i aktualizowanie danych. Ze względu na wysoką wydajność, elastyczność i wygodę użytkowania, aplikacje typu SPA stanowią obecnie dominujący model nowoczesnych aplikacji webowych.

Nowoczesne podejścia architektoniczne



21

Architektura monolityczna

Architektura monolityczna polega na budowie całej aplikacji jako jednego, spójnego systemu, w którym wszystkie funkcjonalności są ze sobą ściśle powiązane i wdrażane razem. W początkowej fazie projektu takie podejście jest stosunkowo proste w implementacji, testowaniu i uruchamianiu, ponieważ cała logika aplikacji znajduje się w jednym miejscu. Ułatwia to szybkie rozpoczęcie pracy nad projektem i sprawne wprowadzanie pierwszych funkcjonalności.

Wraz ze wzrostem skali aplikacji architektura monolityczna może jednak stawać się problematyczna. Rozbudowa jednego elementu często wymaga modyfikacji i ponownego wdrożenia całego systemu, co zwiększa ryzyko błędów i utrudnia rozwój. Dobrym przykładem monolitu jest tradycyjna platforma e-commerce, w której katalog produktów, koszyk zakupowy, obsługa płatności i zarządzanie zamówieniami funkcjonują jako jedna całość. Silne powiązania między tymi modułami sprawiają, że z czasem utrzymanie i skalowanie takiego systemu staje się coraz trudniejsze.

22

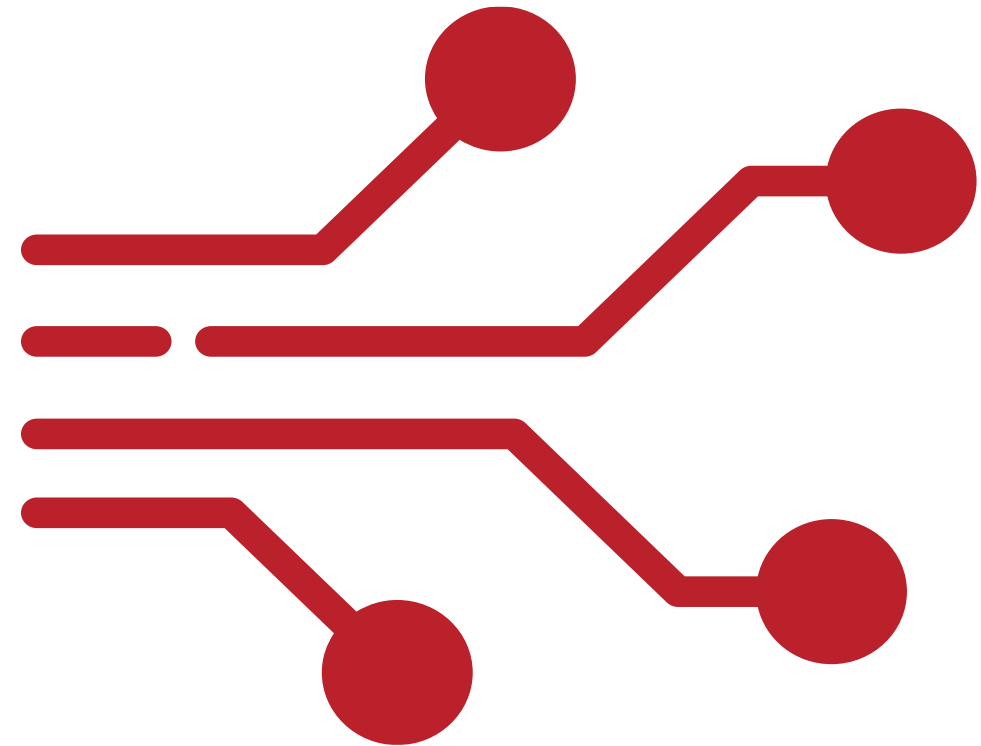
Architektura usługowa i mikroserwisy

Architektura usługowa oraz mikroserwisy to podejścia, w których aplikacja składa się z wielu niezależnych komponentów backendu, zwanych usługami. Każda usługa realizuje ściśle określoną funkcję i komunikuje się z innymi elementami systemu za pomocą dobrze zdefiniowanych interfejsów. Takie podejście zwiększa elastyczność systemu oraz ułatwia jego rozwój, utrzymanie i skalowanie, ponieważ poszczególne usługi mogą być modyfikowane lub wdrażane niezależnie.

Dobrym przykładem zastosowania tej architektury jest system bankowy. Usługa walidacji klienta może być wykorzystywana jednocześnie przez system kredytowy, system otwierania kont oraz system ubezpieczeń, bez konieczności powielania tej samej logiki w różnych częściach systemu. Dodatkowo często stosuje się system ESB (Enterprise Service Bus), który pełni rolę pośrednika i umożliwia komunikację pomiędzy różnymi systemami, na przykład nowoczesną aplikacją mobilną a starszym systemem mainframe. Dzięki temu cała infrastruktura jest bardziej spójna, skalowalna i odporna na zmiany.

23

Podsumowanie



24

Aplikacje internetowe opierają się na modelu klient–serwer, w którym przeglądarka użytkownika komunikuje się z backendem za pomocą protokołu HTTP, wymieniając żądania i odpowiedzi. Kluczowym elementem takiego podejścia jest wyraźny podział aplikacji na frontend, odpowiedzialny za interfejs użytkownika i obsługę interakcji, oraz backend, który realizuje logikę biznesową, przetwarzanie danych, komunikację z bazami danych oraz kwestie bezpieczeństwa. Aby ułatwić rozwój, testowanie i utrzymanie systemu, aplikacje projektuje się zgodnie z architekturą warstwową, oddzielając warstwę prezentacji, logiki oraz dostępu do danych. Współczesne aplikacje webowe coraz częściej wykorzystują rozbudowany frontend oraz backend udostępniający funkcjonalności w postaci usług i API, co pozwala na większą skalowalność, elastyczność oraz łatwiejszą integrację z innymi systemami i platformami.

Literatura:

- [1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.
- [2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.
- [3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.
- [4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.
- [5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Dziękuję za uwagę

Wykład nr 4

Temat: Struktura dokumentów i podstawy tworzenia stron internetowych. Protokół HTTP. Obiektowy model dokumentów

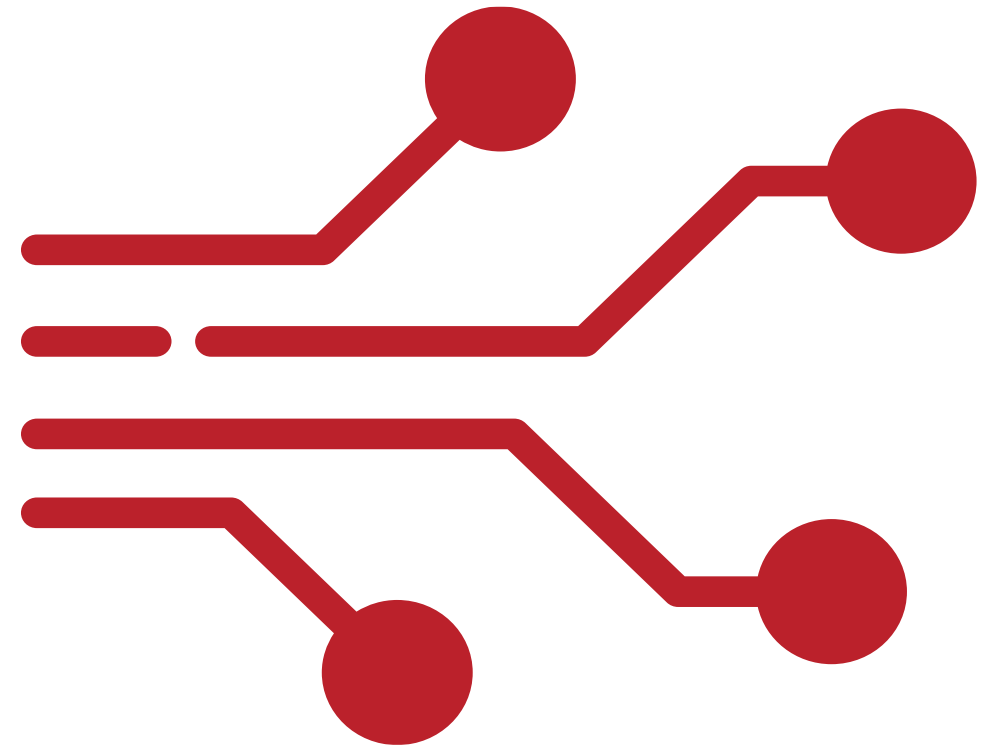
dr hab. inż. Michał Wydra, prof. Uczelni
Katedra Informatyki Stosowanej

Plan prezentacji

- Dokument HTML jako podstawa strony internetowej
- Struktura i elementy języka HTML
- Protokół HTTP i komunikacja klient-serwer
- Obiektowy Model Dokumentów (DOM)
- Znaczenie struktury dokumentu w aplikacjach webowych



Dokument HTML jako podstawa strony internetowej



Czym jest HTML?

HTML (HyperText Markup Language) to podstawowy język znaczników służący do definiowania struktury oraz semantyki stron internetowych. Określa on, które elementy treści są nagłówkami, akapitami, listami czy formularzami, dzięki czemu przeglądarka może poprawnie je interpretować. HTML, obok CSS i JavaScript, stanowi jeden z filarów frontendu, odpowiadając wyłącznie za logiczną strukturę treści, a nie za jej wygląd czy zachowanie.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="UTF-8">
5      <title>Title goes here</title>
6    </head>
7    <body>
8
9    </body>
10 </html>
```

Rys. 1. Przykład kodu HTML [źródło: <https://technologia-swiat.pl/wprowadzenie-do-html-co-trzeba-wiedziec-na-poczatek/>]

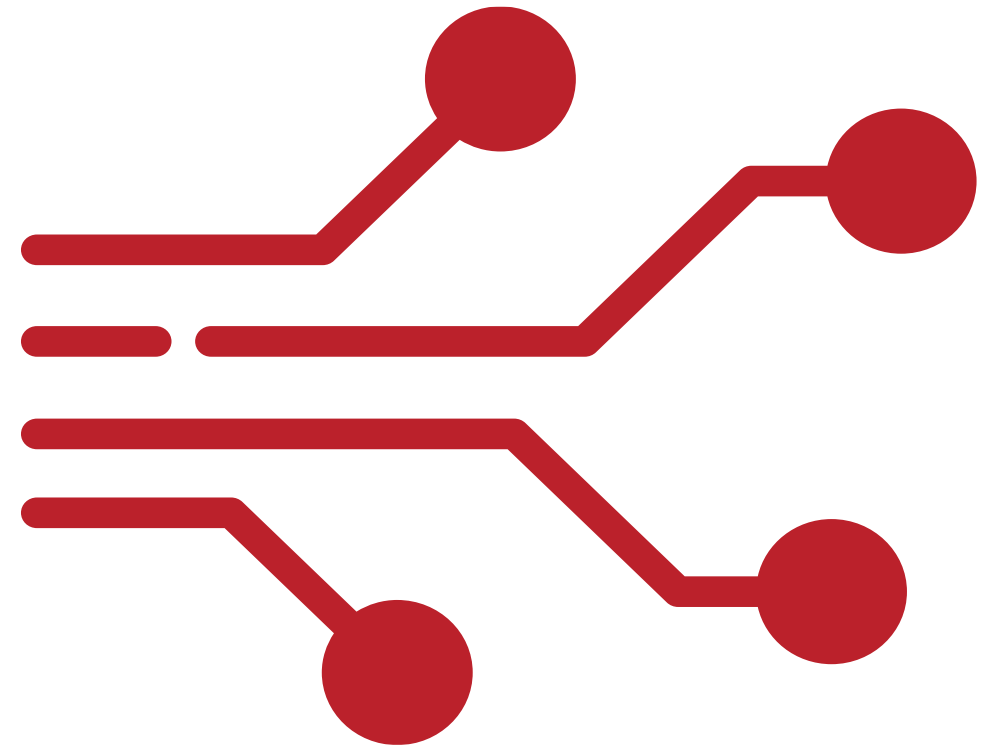
4

Dokument HTML jako plik

Dokument HTML jest zwykłym plikiem tekstowym, który najczęściej zapisywany jest z rozszerzeniem `.html` lub `.htm`. Oznacza to, że jego zawartość można swobodnie przeglądać i edytować w dowolnym edytorze tekstu, takim jak Notatnik, Visual Studio Code czy inne narzędzia przeznaczone do pracy z kodem. Plik HTML składa się z czytelnych znaczników oraz tekstu, co sprawia, że jego struktura jest zrozumiała zarówno dla przeglądarki, jak i dla człowieka.

Do uruchomienia dokumentu HTML nie jest potrzebna kompilacja ani specjalistyczne oprogramowanie. Wystarczy otworzyć plik bezpośrednio w przeglądarce internetowej, która interpretuje znaczniki HTML i renderuje stronę w formie graficznej. Dzięki temu proces tworzenia i testowania stron jest szybki i prosty, a każda wprowadzona zmiana może być natychmiast sprawdzona poprzez ponowne odświeżenie strony.

Struktura i elementy języka HTML



Podstawowa struktura HTML

Każdy dokument HTML składa się z kilku podstawowych elementów: deklaracji doctype, znacznika `<html>`, sekcji `<head>` oraz sekcji `<body>`. Sekcja `<head>` zawiera informacje techniczne i metadane, natomiast `<body>` obejmuje treści widoczne dla użytkownika.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="UTF-8">
5      <title>Title goes here</title>
6    </head>
7    <body>
8
9    </body>
10 </html>
```

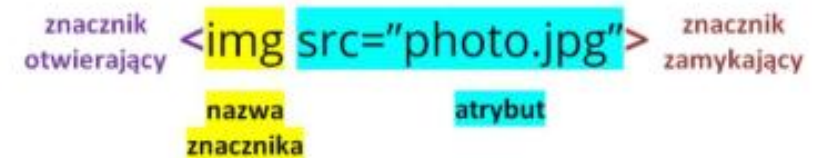
Rys. 1. Przykład kodu HTML [źródło: <https://technologia-swiat.pl/wprowadzenie-do-html-co-trzeba-wiedziec-na-poczatek/>]

7

Elementy i znaczniki

Elementy HTML składają się z tagu otwierającego, treści oraz tagu zamykającego. Przykładem są nagłówki, akapity czy listy. Niektóre elementy, takie jak `<br>` lub `<img>`, są elementami pustymi i nie zawierają treści.

Znacznik pojedynczy



Znacznik podwójny



Rys. 2. Znaczniki [źródło: <https://e-pasje.pl/html-podstawy-co-to-jest-kod-html/>]

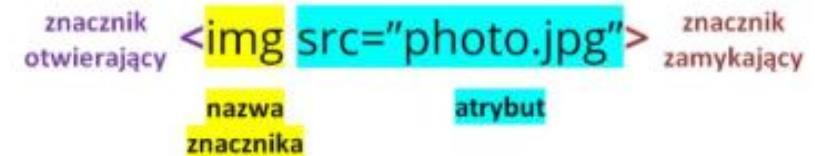
Atrybuty elementów

Atrybuty umożliwiają przekazywanie dodatkowych informacji do elementów HTML.

Składają się z nazwy i wartości i są umieszczane w tagu otwierającym.

Przykładem jest atrybut `href` w znaczniku `<a>`, który określa adres docelowy odsyłacza.

Znacznik pojedynczy



Znacznik podwójny



Rys. 2. Znaczniki [źródło: <https://e-pasje.pl/html-podstawy-co-to-jest-kod-html/>]

Popularne znaczniki HTML

Popularne znaczniki HTML	
Znacznik (tag)	Opis
<p>	paragraf
	pogrubienie (bold)
<html>	znacznik otwierający dokument
<head>	znacznik otwierający sekcję HEAD
<body>	znacznik otwierający sekcję BODY
	grafika
<h1>	nagłówek pierwszego stopnia
<h2>	nagłówek drugiego stopnia
<h3>	nagłówek trzeciego stopnia
 	zakończenie linii
<center>	wycentrowanie
<form>	formularz
<hr>	linia pozioma
<i>	pochylenie tekstu
<u>	podkreślenie tekstu
<table>	tabela
<tr>	wiersz w tabeli
<td>	komórka w tabeli
	lista
	czcionka
	pogrubienie tekstu
<small>	pomniejszenie tekstu

Rys. 3. Popularne znaczniki HTML [źródło: <https://lepiej-widoczni.pl/wp-content/uploads/2020/06/znacznikihtml.png>]

Najważniejsze elementy tekstowe

HTML oferuje elementy do strukturyzowania treści tekstowych, takie jak nagłówki `<h1>`–`<h6>`, akapity `<p>` oraz elementy wyróżnień tekstu.

Poprawne użycie tych elementów wpływa na czytelność strony i jej dostępność.

Widok w HTML

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4   <h1>Nagłówek 1</h1>
5   <h2>Nagłówek 2</h2>
6   <h3>Nagłówek 3</h3>
7   <h4>Nagłówek 4</h4>
8   <h5>Nagłówek 5</h5>
9   <h6>Nagłówek 6</h6>
10 </body>
11 </html>
```

Widok w przeglądarce

Nagłówek 1

Nagłówek 2

Nagłówek 3

Nagłówek 4

Nagłówek 5

Nagłówek 6

Rys. 4. Nagłówki [źródło: <https://strategiczni.pl/podstawowe-znaczniki-html-czy-je-znasz>]

11

Listy, obrazy, tabele

Listy, obrazy i tabele należą do najczęściej wykorzystywanych elementów na stronach internetowych, ponieważ umożliwiają czytelne i logiczne prezentowanie informacji. HTML udostępnia specjalne znaczniki do tworzenia list uporządkowanych i nieuporządkowanych, które pozwalają grupować treści w przejrzysty sposób, na przykład w postaci punktów lub numerowanych kroków. Dzięki temu użytkownik łatwiej przyswaja informacje i szybciej odnajduje interesujące go dane.

Obrazy są kolejnym istotnym elementem stron internetowych, wzbogacającym treść wizualnie i pomagającym w przekazywaniu informacji. HTML umożliwia ich osadzanie wraz z opisami alternatywnymi, co ma znaczenie zarówno dla dostępności, jak i pozycjonowania stron. Z kolei tabele służą do prezentowania danych w postaci wierszy i kolumn, co sprawdza się szczególnie w przypadku zestawień, wyników czy porównań. Wykorzystanie dedykowanych elementów HTML pozwala zachować porządek w strukturze dokumentu i poprawia czytelność prezentowanych treści.

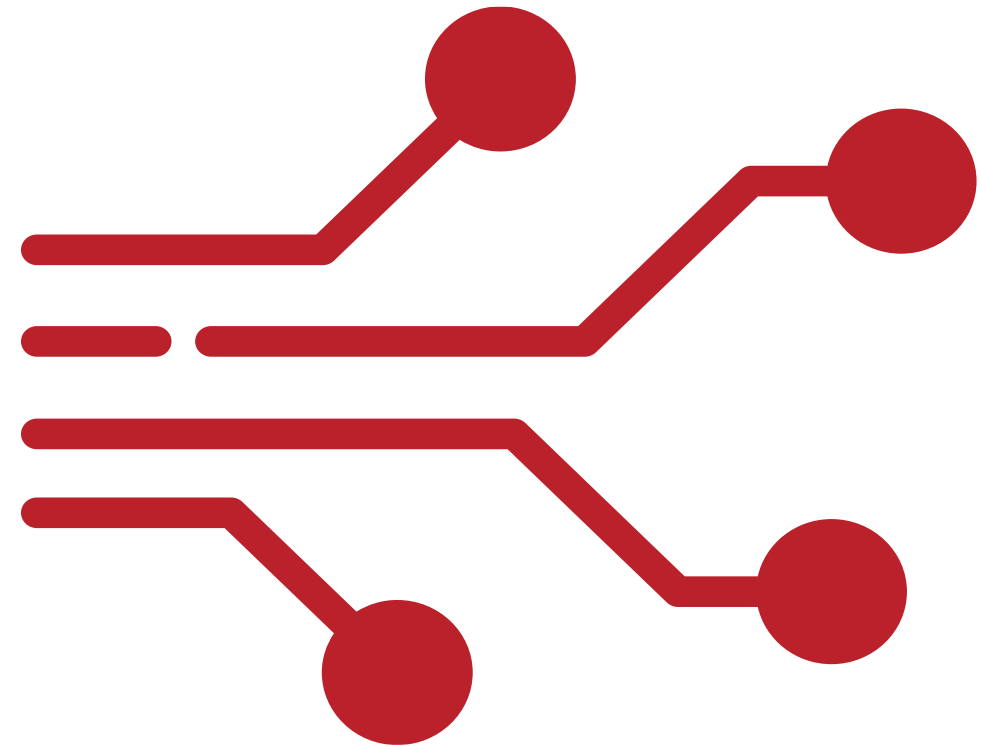
12

Formularze

Formularze są jednym z najważniejszych elementów stron i aplikacji internetowych, ponieważ umożliwiają użytkownikowi wprowadzanie danych oraz przesyłanie ich do serwera. To właśnie za pomocą formularzy użytkownik może się zalogować, zarejestrować, wypełnić ankietę czy wysłać wiadomość. Formularz stanowi więc podstawowy sposób komunikacji użytkownika z aplikacją webową.

Formularze składają się z różnych typów pól, takich jak pola tekstowe, przyciski, listy wyboru, pola wyboru czy pola do wprowadzania haseł. HTML pozwala na precyzyjne określenie rodzaju danych, jakie mają zostać wprowadzone, co ułatwia ich późniejsze przetwarzanie. Dzięki formularzom aplikacje webowe mogą zbierać dane, reagować na działania użytkownika i realizować kluczowe funkcjonalności systemu.

Protokół HTTP i komunikacja klient-serwer



14

Protokół HTTP

HTTP (HyperText Transfer Protocol) jest podstawowym protokołem komunikacyjnym wykorzystywanym w Internecie do wymiany danych pomiędzy przeglądarką użytkownika a serwerem. Określa on, w jaki sposób klient wysyła żądania (np. o pobranie strony lub danych), a serwer przygotowuje i odsyła odpowiedzi.

Protokół HTTP definiuje strukturę komunikatów, dostępne metody żądań oraz zasady przesyłania informacji, takich jak nagłówki i treść wiadomości. Dzięki temu możliwa jest uporządkowana i jednolita komunikacja między różnymi systemami, co stanowi fundament działania stron i aplikacji internetowych.

Żądania i odpowiedzi HTTP

Komunikacja w sieci opiera się na mechanizmie żądań i odpowiedzi HTTP. Przeglądarka, działając jako klient, wysyła do serwera żądanie HTTP, w którym określa, jakiego zasobu lub danych oczekuje. Serwer odbiera to żądanie, przetwarza je, a następnie odsyła odpowiedź zawierającą dokument HTML, dane (np. w formacie JSON) lub odpowiedni kod statusu informujący o wyniku operacji.

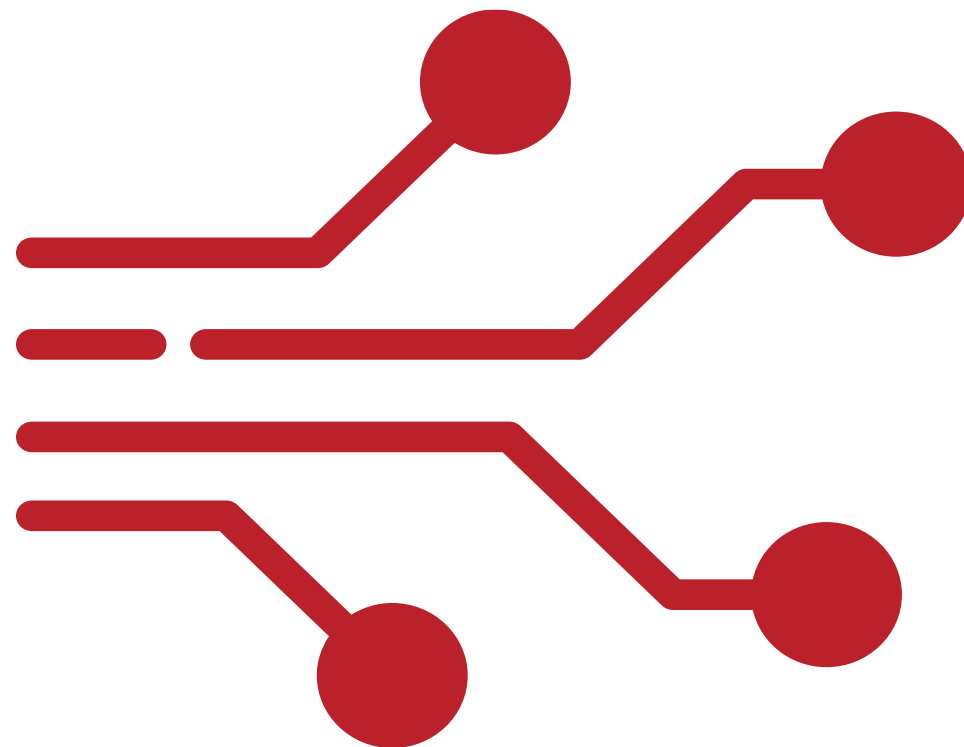
Ten prosty, ale skuteczny mechanizm stanowi fundament działania stron i aplikacji internetowych, umożliwiając pobieranie treści, przesyłanie danych oraz realizację interakcji użytkownika z systemem.

Metody HTTP

Metody HTTP określają, jaki rodzaj operacji ma zostać wykonany na zasobie po stronie serwera. Najczęściej wykorzystywane są metody **GET** i **POST**. Metoda GET służy do pobierania danych i zwykle przekazuje parametry w adresie URL, co sprawia, że jest odpowiednia do odczytu informacji, ale nie do przesyłania danych wrażliwych.

Metoda POST wykorzystywana jest do wysyłania danych do serwera, na przykład podczas przesyłania formularzy czy tworzenia nowych zasobów. Dane nie są wówczas widoczne w adresie strony, co zwiększa bezpieczeństwo i czytelność komunikacji. Wybór odpowiedniej metody HTTP ma istotne znaczenie zarówno dla bezpieczeństwa aplikacji, jak i dla poprawnego znaczenia wykonywanych operacji.

Obiektowy Model Dokumentów (DOM)



18

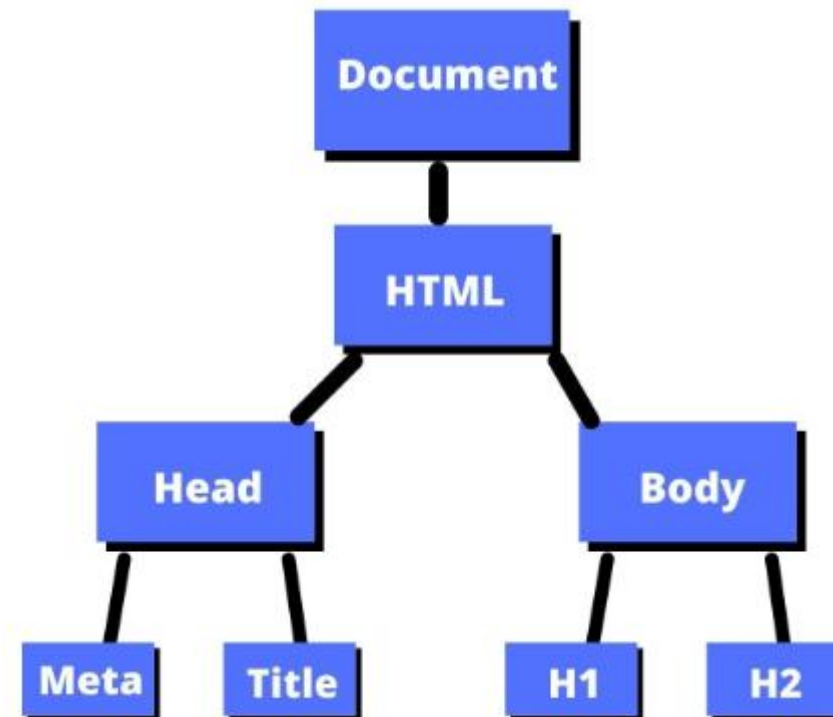
Czym jest DOM?

DOM (Document Object Model) to wewnętrzna reprezentacja dokumentu HTML w postaci hierarchicznej struktury obiektów, nazywanej drzewem DOM. Po wczytaniu strony przeglądarka interpretuje kod HTML i przekształca każdy element, atrybut oraz fragment tekstu w obiekty, które są ze sobą powiązane relacjami nadrzędności i podrzędności. Struktura ta dokładnie odzwierciedla budowę strony internetowej.

Dzięki DOM możliwa jest dynamiczna manipulacja zawartością strony za pomocą JavaScriptu. Programista może odczytywać, modyfikować lub usuwać elementy, zmieniać ich atrybuty i style oraz reagować na działania użytkownika. DOM stanowi więc kluczowe połączenie pomiędzy strukturą HTML a logiką aplikacji, umożliwiając tworzenie interaktywnych i dynamicznych stron internetowych.

Drzewo DOM

Drzewo DOM przedstawia dokument HTML jako hierarchiczną strukturę węzłów, w której każdy element strony jest osobnym węzłem. Węzły te pozostają ze sobą w relacjach rodzinnych – mogą mieć elementy nadrzędne, podrzędne oraz elementy znajdujące się na tym samym poziomie, zwane rodzeństwem. Taki układ dokładnie odzwierciedla strukturę dokumentu HTML zapisaną w kodzie strony.



Rys. 5. Wizualizacja drzewa DOM [źródło: <https://www.freecodecamp.org/news/what-is-the-dom-document-object-model-meaning-in-javascript/>]

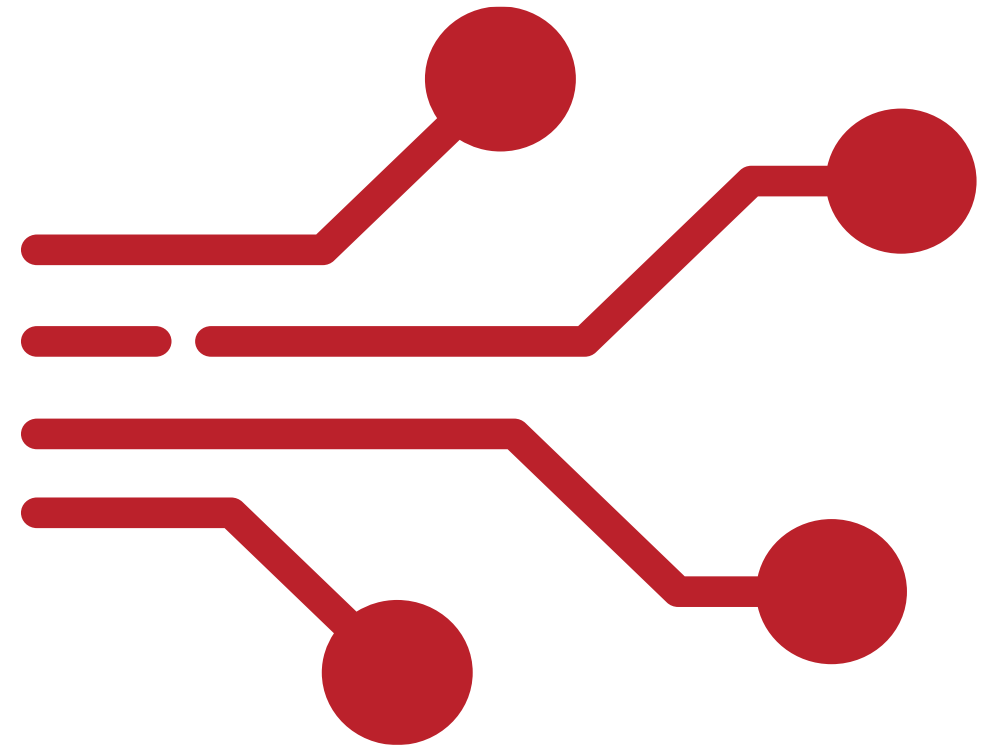
20

DOM a JavaScript

JavaScript umożliwia bezpośrednią i dynamiczną manipulację obiektami DOM, czyli elementami reprezentującymi strukturę dokumentu HTML. Programista może za jego pomocą zmieniać treść elementów, modyfikować style, dodawać lub usuwać elementy oraz reagować na zdarzenia, takie jak kliknięcia czy wprowadzanie danych. Wszystkie te operacje odbywają się na już załadowanej stronie.

Dzięki połączeniu JavaScriptu z DOM strony internetowe mogą reagować na działania użytkownika w czasie rzeczywistym, bez konieczności ponownego ładowania całego dokumentu. Pozwala to tworzyć interaktywne interfejsy, dynamiczne formularze czy animacje, które znacząco poprawiają komfort i płynność korzystania z aplikacji webowych.

Znaczenie struktury dokumentu w aplikacjach webowych



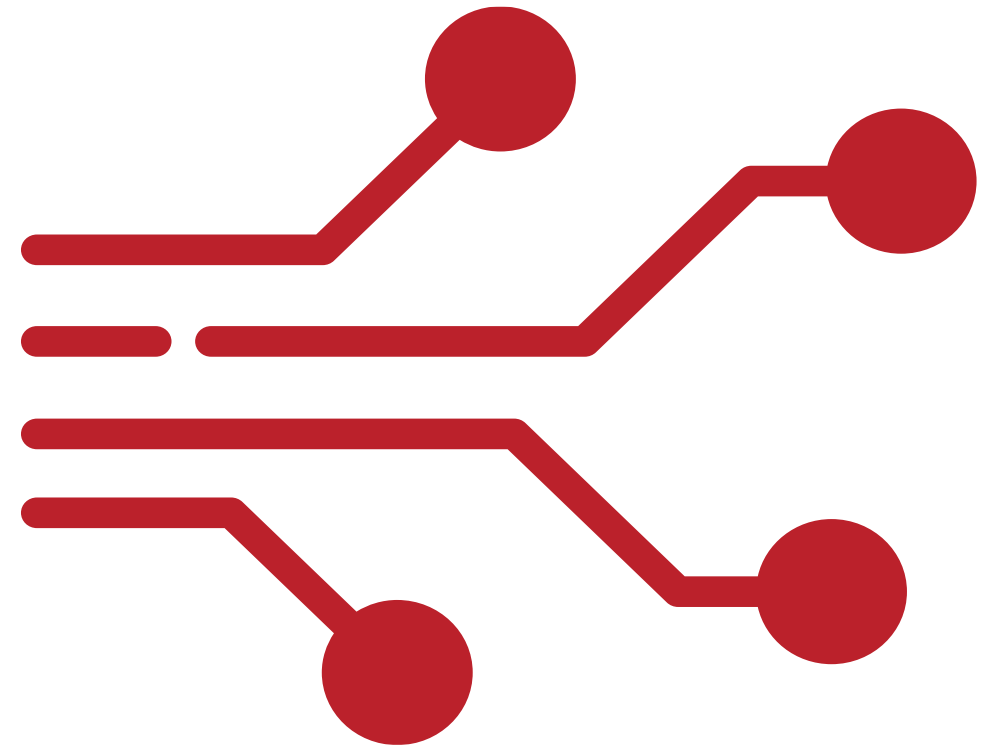
22

Znaczenie poprawnej struktury

Poprawna struktura dokumentu HTML ma kluczowe znaczenie dla jakości i funkcjonowania strony internetowej. Odpowiednie użycie znaczników i semantyki wpływa na dostępność, umożliwiając korzystanie ze strony osobom z niepełnosprawnościami oraz poprawną interpretację treści przez technologie wspomagające. Dobrze zorganizowany kod jest także czytelniejszy i łatwiejszy w utrzymaniu dla programistów.

Struktura HTML ma również istotny wpływ na pozycjonowanie strony w wyszukiwarkach internetowych, ponieważ ułatwia robotom wyszukiwarek analizę i indeksowanie treści. Ponadto stanowi fundament poprawnego działania CSS i JavaScript, które opierają się na strukturze dokumentu do stylizacji i manipulacji elementami. Dzięki temu dobrze zaprojektowany HTML ułatwia dalszy rozwój i rozbudowę aplikacji webowej.

Podsumowanie



24

HTML odpowiada za zdefiniowanie struktury i semantyki strony internetowej, HTTP umożliwia komunikację pomiędzy przeglądarką a serwerem, natomiast DOM stanowi wewnętrzną reprezentację dokumentu HTML w przeglądarce. Elementy te współpracują ze sobą, tworząc podstawowy mechanizm działania stron i aplikacji webowych. Zrozumienie ich wzajemnych zależności jest kluczowe dla świadomego projektowania nowoczesnych rozwiązań internetowych oraz stanowi solidny fundament do dalszej nauki stylizacji za pomocą CSS i programowania interakcji w JavaScript.

Literatura:

- [1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.
- [2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.
- [3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.
- [4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.
- [5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Dziękuję za uwagę

Wykład nr 5

Temat: Formatowanie i stylizacja dokumentów - kaskadowe arkusze stylów

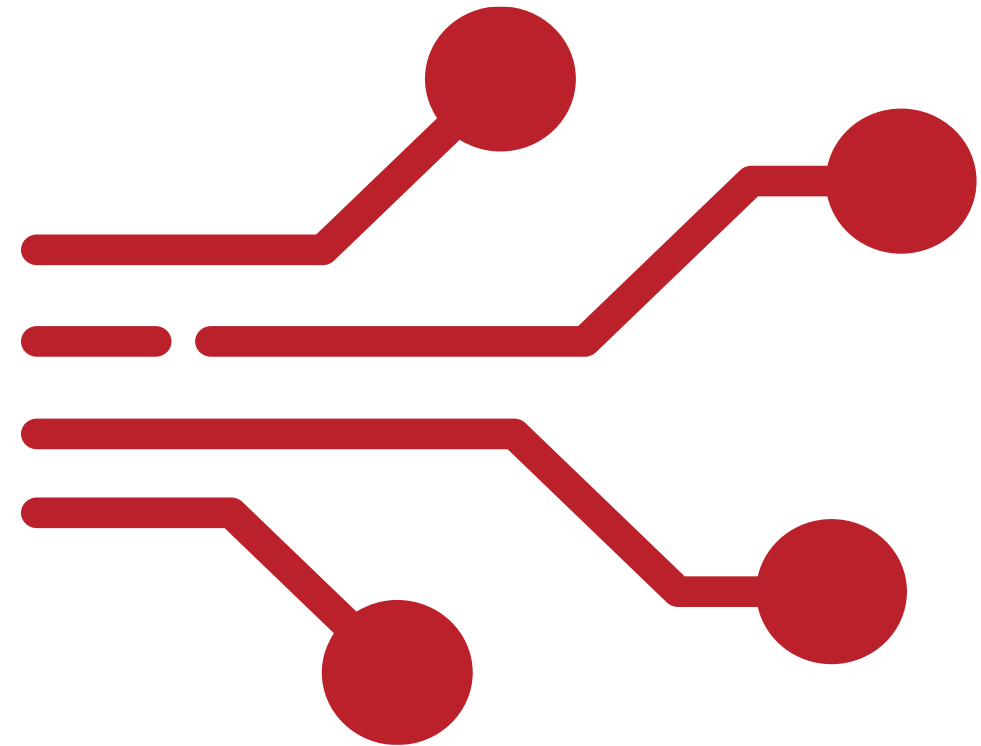
dr hab. inż. Michał Wydra, prof. Uczelni
Katedra Informatyki Stosowanej

Plan prezentacji

- Rola CSS w tworzeniu stron internetowych
- Budowa i sposoby stosowania arkuszy stylów
- Selektory, kaskada i dziedziczenie
- Formatowanie tekstu, kolorów i elementów
- Układ strony i nowoczesne systemy layoutu



Rola CSS w tworzeniu stron internetowych



Czym jest CSS?

CSS (Cascading Style Sheets) to język służący do opisywania wyglądu i sposobu prezentacji elementów HTML w przeglądarce internetowej. Jego głównym zadaniem jest oddzielenie warstwy wizualnej od struktury dokumentu, dzięki czemu HTML odpowiada za treść i semantykę, a CSS za estetykę strony. Za pomocą arkuszy stylów można kontrolować kolory, czcionki, rozmiary tekstu, odstępy między elementami, obramowania oraz ich rozmieszczenie na stronie.



Rys. 1. Zasada kodu CSS [źródło:
<https://how2html.pl/css-podstawy/>]

HTML a CSS

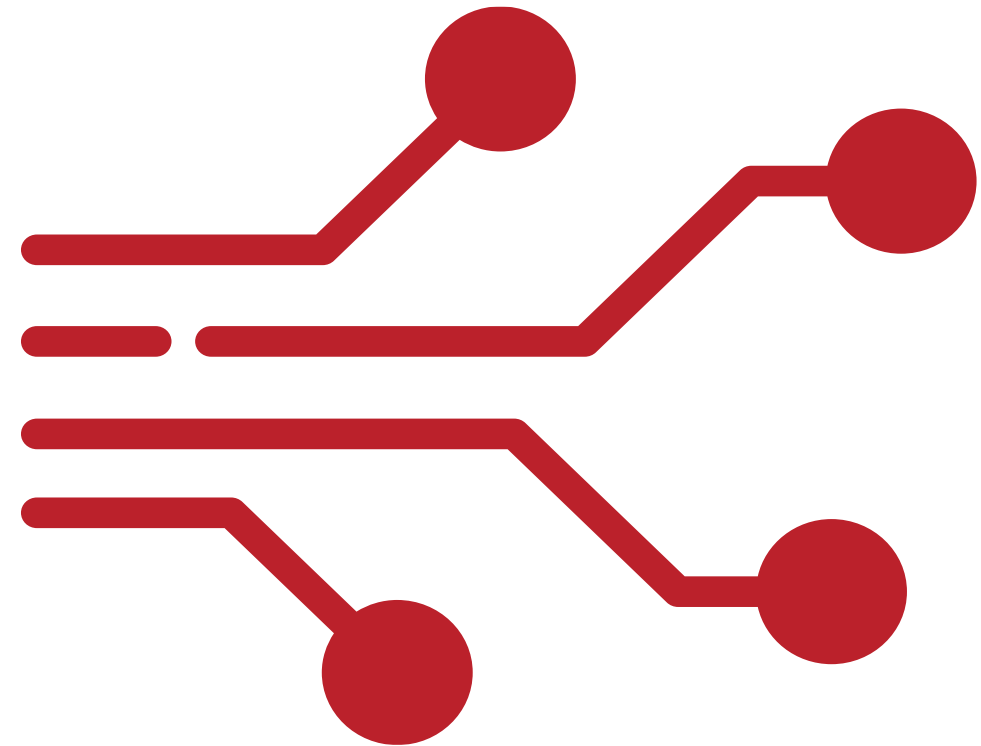
HTML i CSS pełnią w tworzeniu stron internetowych odmienne, ale uzupełniające się role. HTML odpowiada za strukturę oraz znaczenie treści, określając, które elementy są nagłówkami, akapitami, listami czy formularzami. CSS natomiast zajmuje się wyglądem tych elementów, definiując kolory, czcionki, układ, odstępy oraz sposób prezentacji treści w przeglądarce.

```
1  <!doctype html>
2  <html lang="pl">
3
4  <head>
5      <meta charset="utf-8">
6      <title>Richard Feynman</title>
7      <style>
8          p {margin-top: 5px;}
9          h1 {margin-bottom: 0; margin-top: 30px;}
10         h2 {margin-bottom: 0; margin-top: 30px;}
11         h3 {margin-bottom: 0; margin-top: 30px;}
12     </style>
13 </head>
14
15 <body>
16     <h1>Richard Feynman</h1>
```

Rys. 2. Praca z HTML i CSS [źródło:
<https://shebang.pl/html/podstawy-css/>]

5

Budowa i sposoby stosowania arkuszy stylów



6

Reguły CSS

Reguła CSS jest podstawowym elementem arkusza stylów i określa, w jaki sposób wybrane elementy HTML mają być wyświetlane w przeglądarce. Składa się ona z selektora, który wskazuje, do jakich elementów reguła ma zostać zastosowana, oraz z zestawu deklaracji zapisanych w nawiasach klamrowych. Dzięki selektorom CSS można precyzyjnie wybierać pojedyncze elementy, grupy elementów lub elementy o określonych cechach.



Rys. 3. Reguła CSS [źródło:
<https://delante.pl/definicje/css/>]

7

Zewnętrzne arkusze stylów

Zewnętrzne arkusze stylów to najczęściej stosowany sposób definiowania wyglądu stron internetowych. Polegają one na umieszczeniu reguł CSS w osobnym pliku, który następnie jest dołączany do dokumentów HTML. Dzięki temu kod odpowiedzialny za strukturę i treść strony pozostaje oddzielony od kodu odpowiedzialnego za jej wygląd.

Takie rozwiązanie umożliwia centralne zarządzanie stylami wielu stron jednocześnie – zmiana w jednym pliku CSS może automatycznie wpłynąć na wygląd całego serwisu. Zapewnia to spójność wizualną, ułatwia utrzymanie projektu oraz przyspiesza pracę nad większymi aplikacjami i serwisami internetowymi.

Wewnętrzne i liniowe style

Style CSS mogą być definiowane nie tylko w zewnętrznych arkuszach stylów, ale także bezpośrednio w dokumencie HTML. Wewnętrzne style umieszcza się w sekcji <head> strony, co pozwala na stylizowanie elementów danej strony bez tworzenia osobnego pliku CSS. Rozwiązanie to bywa przydatne w prostych projektach lub wtedy, gdy style dotyczą wyłącznie jednego dokumentu.

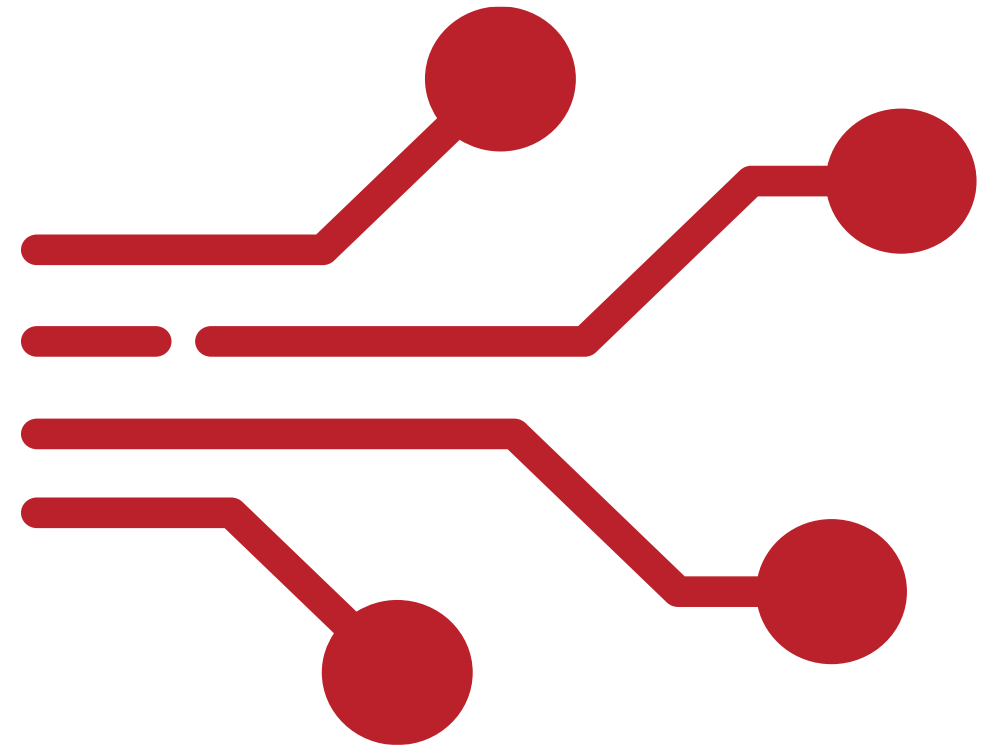
Style liniowe są zapisywane bezpośrednio w atrybucie konkretnego elementu HTML i wpływają tylko na ten jeden element. Choć dają one szybki efekt, znacząco obniżają czytelność kodu i utrudniają jego utrzymanie. Z tego powodu zarówno style wewnętrzne, jak i liniowe stosuje się rzadziej – głównie w wyjątkowych sytuacjach, do testów lub w celach dydaktycznych.

Importowanie arkuszy CSS

CSS umożliwia importowanie innych arkuszy stylów, co pozwala na dzielenie kodu na mniejsze, logiczne moduły. Dzięki temu style mogą być organizowane tematycznie, na przykład w osobnych plikach dla formularzy, tabel, typografii czy układu strony. Takie podejście zwiększa przejrzystość kodu i ułatwia jego zrozumienie.

Modularna struktura arkuszy CSS jest szczególnie przydatna w większych projektach, gdzie jeden plik stylów byłby trudny w utrzymaniu. Importowanie stylów pozwala na łatwiejsze zarządzanie zmianami, ponowne wykorzystywanie reguł oraz pracę wielu osób nad tym samym projektem bez ryzyka chaosu w kodzie.

Selektory, kaskada i dziedziczenie



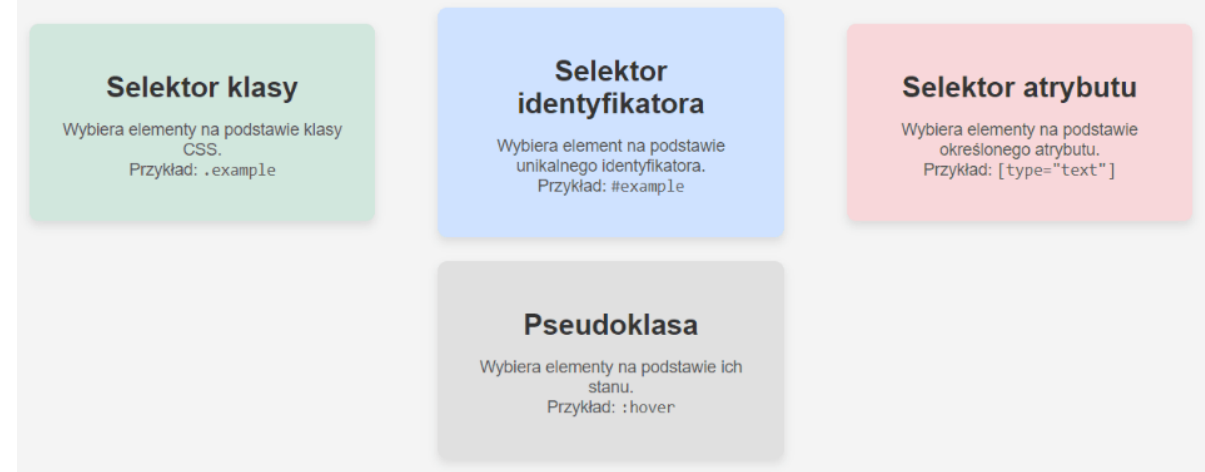
11

Selektory CSS

Selektory CSS służą do określania, do których elementów HTML mają zostać zastosowane reguły stylów. Dzięki nim przeglądarka wie, jakie elementy dokumentu powinny zostać objęte danym zestawem deklaracji. Selekcja może dotyczyć pojedynczych elementów lub całych grup elementów spełniających określone warunki.

CSS umożliwia wybieranie elementów na podstawie różnych kryteriów, takich jak typ znacznika, przypisana klasa, unikalny identyfikator, atrybuty czy relacje między elementami w strukturze dokumentu.

Diagram: Różnice między selektorami CSS



Rys. 4. Różnice między selektorami CSS [źródło: <https://devstockacademy.pl/blog/programowanie-i-technologie-webowe/selector-css-kluczowa-umiejetnosc-w-stylowaniu-stron/?srsId=AfmBOopttW5tI8qUIOYB-d8495VI8T2vPfBji8dCdJL4CN0cyQDvI8Lu>]

12

Łączenie selektorów

Łączenie selektorów w CSS pozwala tworzyć bardziej precyzyjne reguły stylów, które są stosowane tylko do elementów spełniających określone warunki. Dzięki temu możliwe jest stylizowanie elementów w konkretnym kontekście, na przykład tylko tych, które znajdują się wewnątrz danego kontenera, sekcji lub formularza. Takie podejście zwiększa kontrolę nad wyglądem strony i zapobiega niepożądanym efektom ubocznym.

Stosowanie złożonych selektorów jest szczególnie przydatne w większych projektach, gdzie te same typy elementów pojawiają się w wielu miejscach strony. Umożliwia to nadawanie im różnych stylów w zależności od ich położenia w strukturze dokumentu, co sprzyja porządkowi w kodzie oraz łatwiejszemu utrzymaniu i rozbudowie arkuszy CSS.

Kaskada i specyficzność

CSS działa zgodnie z zasadą kaskady, co oznacza, że do jednego elementu HTML może jednocześnie pasować wiele reguł stylów. Przeglądarka musi wtedy zdecydować, które właściwości zostaną ostatecznie zastosowane. Proces ten opiera się na jasno określonych regułach, które zapewniają przewidywalność działania arkuszy stylów.

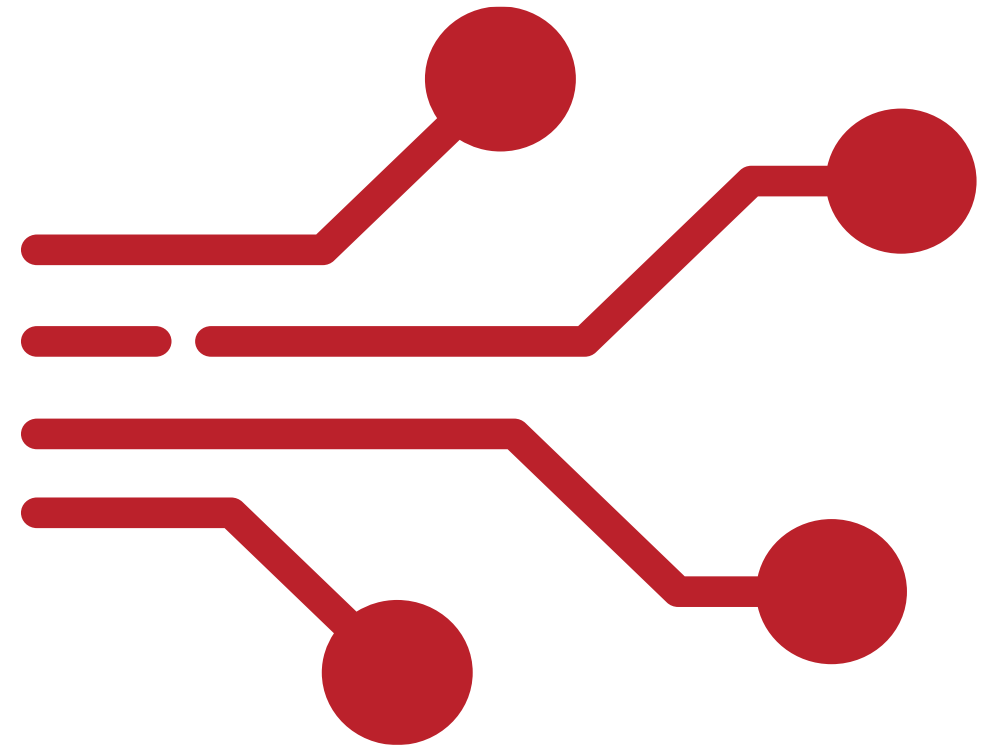
O wyborze reguły decydują przede wszystkim dwa czynniki: kolejność definicji oraz specyficzność selektora. Reguły zapisane później mogą nadpisywać wcześniejsze, ale większe znaczenie ma specyficzność, czyli „siła” selektora. Selektory bardziej szczegółowe, na przykład wykorzystujące identyfikatory lub złożone relacje, mają pierwszeństwo przed ogólnymi. Zrozumienie kaskady i specyficzności jest kluczowe dla poprawnego stylizowania elementów i unikania konfliktów w CSS.

Dziedziczenie

Dziedziczenie w CSS polega na tym, że niektóre właściwości stylów są automatycznie przekazywane z elementów nadrzędnych do ich elementów potomnych. Dotyczy to głównie właściwości związanych z tekstem, takich jak kolor, krój czcionki czy rozmiar pisma. Dzięki temu nie ma potrzeby definiowania tych samych stylów osobno dla każdego elementu, co upraszcza kod i zwiększa jego czytelność.

Nie wszystkie właściwości CSS podlegają jednak dziedziczeniu. Elementy takie jak obramowania, marginesy czy tła nie są dziedziczone, co pozwala uniknąć niepożądanych efektów wizualnych i zachować większą kontrolę nad wyglądem poszczególnych elementów. Świadome korzystanie z dziedziczenia ułatwia tworzenie spójnych i estetycznych układów stron internetowych.

Formatowanie tekstu, kolorów i elementów

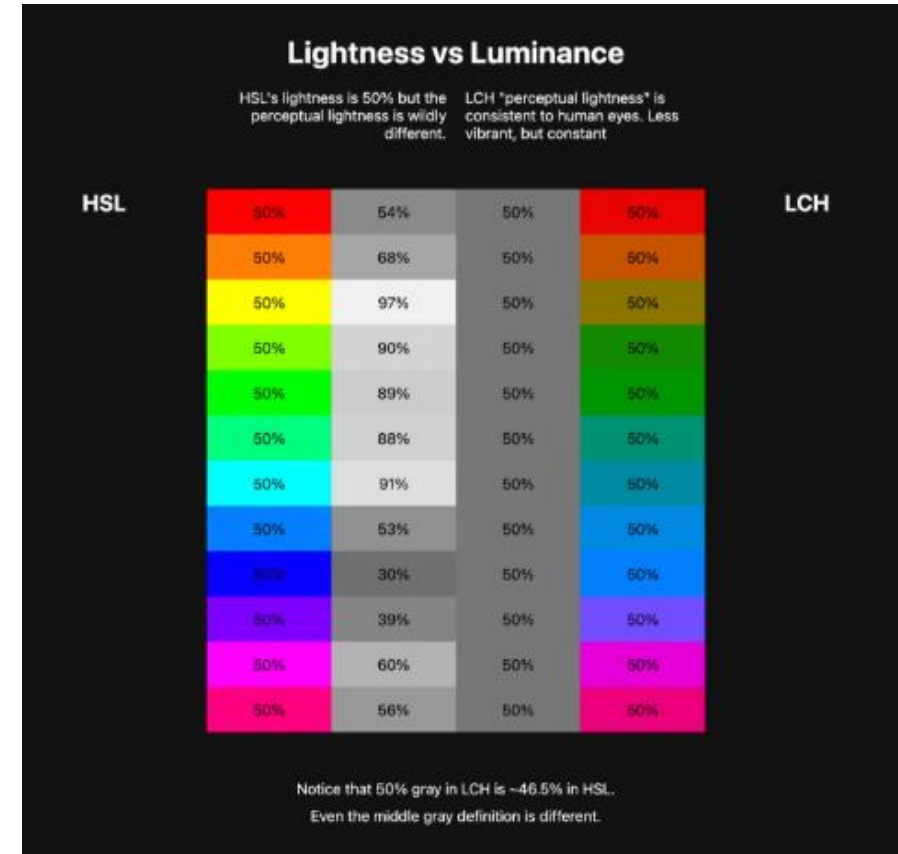


16

Kolory w CSS

CSS oferuje wiele sposobów definiowania kolorów. Najprostszą metodą jest użycie nazw kolorów, takich jak red, blue czy green, jednak ich zakres jest ograniczony. Bardziej elastyczne są zapisy liczbowe, które umożliwiają dokładne określenie barwy.

Do najczęściej stosowanych formatów należą wartości RGB oraz zapis szesnastkowy, które pozwalają definiować kolory na podstawie składowych czerwonej, zielonej i niebieskiej. Modele RGBA i HSL dodatkowo umożliwiają kontrolę przezroczystości lub manipulowanie jasnością i nasyceniem koloru.



Rys. 5. Precepcja jasności [źródło: <https://developer.chrome.com/docs/css-ui/high-definition-css-color-guide?hl=pl>]

17

Formatowanie tekstu

CSS umożliwia zaawansowane formatowanie tekstu, dzięki czemu można w pełni kontrolować jego wygląd na stronie internetowej. Za pomocą właściwości CSS określa się krój czcionki, jej rozmiar, grubość oraz styl, co pozwala dostosować tekst do charakteru strony i potrzeb użytkownika. Dodatkowo możliwe jest ustawianie odstępów między liniami, literami i wyrazami, co wpływa na estetykę i przejrzystość treści.

Nagłówek H1

Jan Kowalski, 01 02 2006

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent ornare ornare est. Fusce dignissim euismod sapien. Fusce ante nisi, accumsan pharetra, fermentum et, ullamcorper vitae, arcu. Sed tempor ligula ut urna. Nam bibendum, nibh ac tincidunt sollicitudin, enim tortor porta lacus, in laoreet nibh mi tempus metus. Nunc et elit. Sed magna nisi, tempor nec, cursus ac, mattis et, nunc. Duis tempus vulputate elit. Duis non sapien. Vestibulum vitae orci ac metus dictum rhoncus. Sed viverra, tellus quis nonummy accumsan, justo enim gravida metus, [eget consequat](#) nunc sem at risus. Curabitur porta. Etiam sagittis augue quis ligula. Duis tristique euismod nisi.

Nagłówek H2

Vestibulum neque dui, blandit sit amet, dignissim nec, scelerisque at, erat. Etiam in augue. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Integer varius, mauris et blandit laoreet, nulla enim tempor est, at condimentum arcu ligula et leo. Maecenas scelerisque, ipsum mollis nonummy placerat, lectus nibh tristique neque, eget pellentesque augue turpis vel risus.

[powrót](#)

Rys. 6. Formatowanie tekstu, czcionka Verdana [źródło:
<https://magazynt3.pl/okielznac-style-formatowanie-tekstu/>]

18

Stylizacja list i tabel

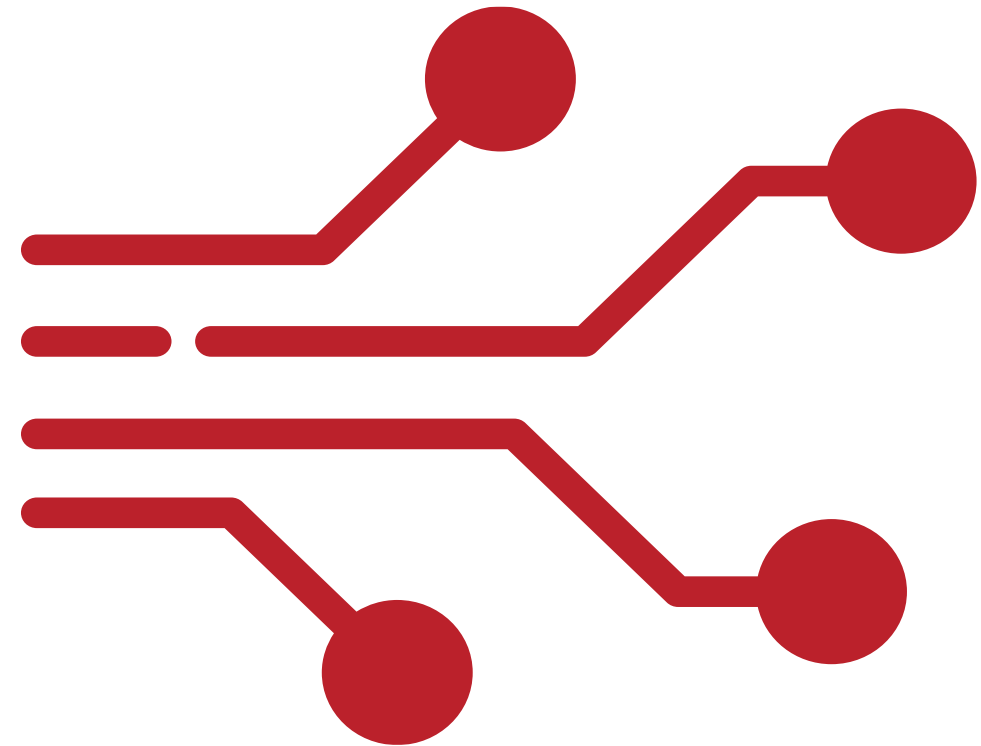
CSS umożliwia rozbudowaną stylizację list oraz tabel, dzięki czemu można dostosować ich wygląd do charakteru strony i poprawić czytelność prezentowanych danych. W przypadku list możliwa jest zmiana typu znaczników, ustawianie odstępów, wyrównania czy wyglądu elementów listy, co pozwala tworzyć zarówno proste, jak i bardziej zaawansowane układy nawigacyjne lub informacyjne.

Tabele mogą być stylizowane poprzez definiowanie obramowań, kolorów tła, odstępów między komórkami czy wyróżnianie poszczególnych wierszy i kolumn.

<i>armenian</i>	<i>circle</i>	<i>decimal</i>	<i>georgian</i>	<i>decimal-leading-zero</i>	<i>lower-alpha</i>	<i>lower-greek</i>	<i>lower-roman</i>	<i>square</i>	<i>upper-alpha</i>	<i>upper-roman</i>
Ա.	°	1.	ს.	01.	a.	α.	i.	■	A.	I.
Բ.	°	2.	ბ.	02.	b.	β.	ii.	■	B.	II.
Գ.	°	3.	გ.	03.	c.	γ.	iii.	■	C.	III.
Դ.	°	4.	დ.	04.	d.	δ.	iv.	■	D.	IV.
Ե.	°	5.	ე.	05.	e.	ε.	v.	■	E.	V.

Rys. 7. Stylizacja list [źródło: <https://mirosławzelent.pl/kurs-css/zmiana-wygladu-list-style-ul-ol/>]

Układ strony i nowoczesne systemy layoutu



20

Pozycjonowanie elementów

CSS oferuje różne sposoby pozycjonowania elementów HTML, które decydują o ich rozmieszczeniu na stronie oraz zachowaniu podczas przewijania. Domyślnym ustawieniem jest pozycjonowanie statyczne, w którym elementy układane są zgodnie z naturalnym przepływem dokumentu. Pozycjonowanie relatywne pozwala przesunąć element względem jego pierwotnego położenia, nie wpływając na układ pozostałych elementów.

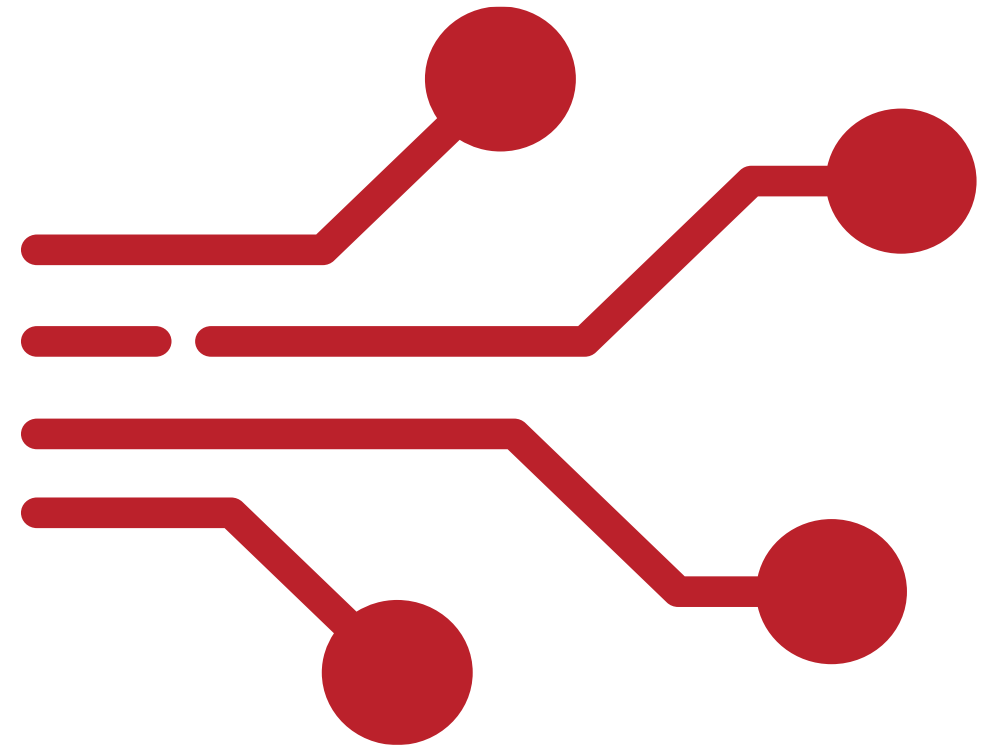
Pozycjonowanie absolutne umożliwia umieszczenie elementu w konkretnym miejscu względem najbliższego elementu nadrzędnego, który posiada pozycjonowanie, natomiast pozycjonowanie względem okna przeglądarki sprawia, że element pozostaje w stałym miejscu nawet podczas przewijania strony. Wybór odpowiedniego sposobu pozycjonowania ma kluczowe znaczenie dla poprawnego działania układu strony oraz komfortu użytkownika.

Systemy layoutu

Systemy layoutu w CSS służą do budowania układu elementów na stronie internetowej. W przeszłości powszechnie wykorzystywano właściwość `float`, która pozwalała ustawiać elementy obok siebie, jednak nie była przeznaczona bezpośrednio do tworzenia złożonych układów i często prowadziła do problemów z utrzymaniem kodu. Z tego powodu obecnie traktowana jest głównie jako rozwiązanie historyczne lub pomocnicze.

Współczesnym standardem są nowocześniejsze mechanizmy, takie jak Flexbox i Grid. Flexbox umożliwia łatwe zarządzanie układem elementów w jednym wymiarze, na przykład w wierszu lub kolumnie, natomiast Grid pozwala tworzyć rozbudowane, dwuwymiarowe siatki. Oba rozwiązania zapewniają dużą elastyczność, czytelniejszy kod oraz lepsze dopasowanie układu do różnych rozmiarów ekranów.

Podsumowanie



23

CSS stanowi kluczowe narzędzie do stylizacji i budowania układu stron internetowych, odpowiadając za ich wygląd i estetykę. Dzięki oddzieleniu warstwy prezentacji od struktury dokumentu HTML ułatwia organizację kodu oraz jego utrzymanie i rozwój. CSS umożliwia precyzyjne kontrolowanie kolorów, typografii i rozmieszczenia elementów, a także tworzenie nowoczesnych, czytelnych i responsywnych interfejsów użytkownika, dostosowanych do różnych urządzeń i rozmiarów ekranów.

Literatura:

- [1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.
- [2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.
- [3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.
- [4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.
- [5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Dziękuję za uwagę

Wykład nr 6
Temat: Podstawy
programowania po stronie
klienta. Wprowadzenie do
języków skryptowych.
Manipulacja strukturą
dokumentu. Obsługa zdarzeń
w aplikacjach internetowych

dr hab. inż. Michał Wydra, prof. Uczelni

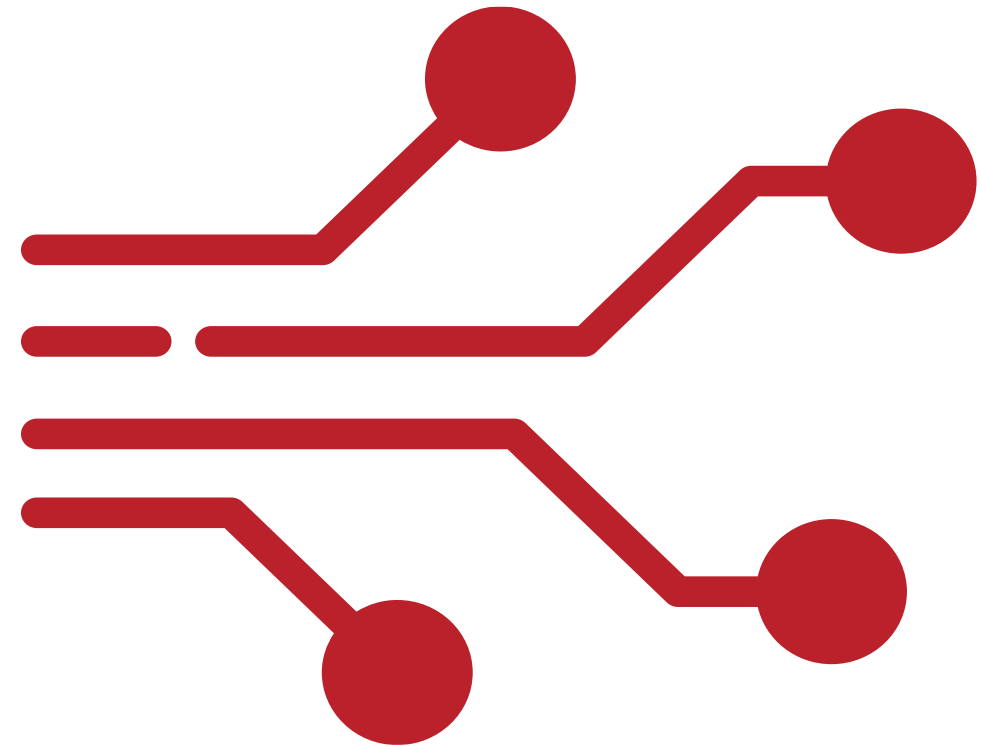
Katedra Informatyki Stosowanej

Plan prezentacji

- Programowanie po stronie klienta
- Języki skryptowe i rola JavaScript
- Podstawy języka JavaScript
- Manipulacja strukturą dokumentu (DOM)
- Obsługa zdarzeń w aplikacjach webowych



Programowanie po stronie klienta



3

Czym jest programowanie po stronie klienta?

Programowanie po stronie klienta odnosi się do kodu, który jest wykonywany bezpośrednio w przeglądarce internetowej użytkownika, bez konieczności każdorazowej komunikacji z serwerem. Dzięki temu możliwe jest szybkie reagowanie aplikacji na działania użytkownika, takie jak kliknięcia, wpisywanie danych czy poruszanie się po stronie. Mechanizm ten znacząco poprawia płynność działania i komfort korzystania ze stron internetowych.

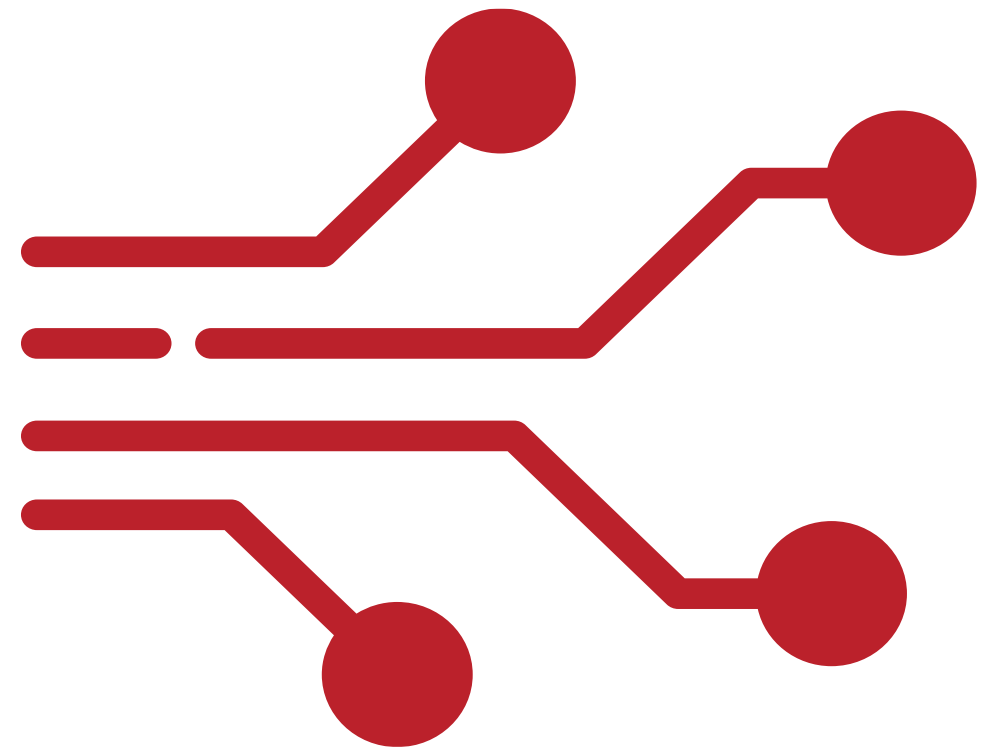
Głównym celem programowania po stronie klienta jest zapewnienie interaktywności oraz dynamicznej aktualizacji treści, na przykład walidacji formularzy, pokazywania lub ukrywania elementów czy aktualizacji danych bez przeładowania strony. Kluczową rolę odgrywa tu JavaScript, który współpracuje z HTML i CSS, umożliwiając tworzenie nowoczesnych, responsywnych i przyjaznych dla użytkownika interfejsów.

Rola przeglądarki

Przeglądarka internetowa pełni kluczową rolę w programowaniu po stronie klienta, ponieważ interpretuje i łączy ze sobą kod HTML, CSS oraz JavaScript. Na podstawie HTML tworzy strukturę strony, CSS odpowiada za jej wygląd, a JavaScript umożliwia dynamiczne działanie i interakcję. Przeglądarka zarządza także tworzeniem drzewa DOM oraz jego aktualizacją w trakcie działania aplikacji.

Dzięki temu możliwe jest wykonywanie wielu operacji bez konieczności ponownego ładowania dokumentu. Przeglądarka potrafi reagować na zdarzenia użytkownika, takie jak kliknięcia czy wprowadzanie danych, walidować formularze oraz dynamicznie zmieniać wygląd i treść strony. To właśnie te mechanizmy sprawiają, że nowoczesne strony internetowe są interaktywne i responsywne.

Języki skryptowe i rola JavaScript



Języki skryptowe

Języki skryptowe to języki programowania, które są interpretowane, a nie kompilowane, co oznacza, że kod jest wykonywany bezpośrednio przez interpreter w momencie uruchomienia. Dzięki temu proces tworzenia i testowania aplikacji jest szybszy, ponieważ zmiany w kodzie można sprawdzić natychmiast, bez dodatkowych etapów przygotowania programu do działania.

Języki skryptowe są powszechnie wykorzystywane do automatyzacji zadań oraz dynamicznego sterowania zachowaniem aplikacji. W kontekście aplikacji internetowych pozwalają reagować na działania użytkownika, modyfikować treść strony w czasie rzeczywistym oraz kontrolować logikę interfejsu. Ich elastyczność i prostota sprawiają, że stanowią ważny element nowoczesnych systemów informatycznych.

JavaScript jako język Webu

JavaScript jest podstawowym językiem skryptowym wykorzystywanym w przeglądarkach internetowych i stanowi jeden z filarów nowoczesnych technologii webowych. To właśnie on odpowiada za dynamiczne zachowanie stron, umożliwiając modyfikowanie treści i struktury dokumentu bez konieczności jego ponownego ładowania. Dzięki JavaScriptowi strony internetowe przestają być statyczne i mogą reagować na działania użytkownika w czasie rzeczywistym.

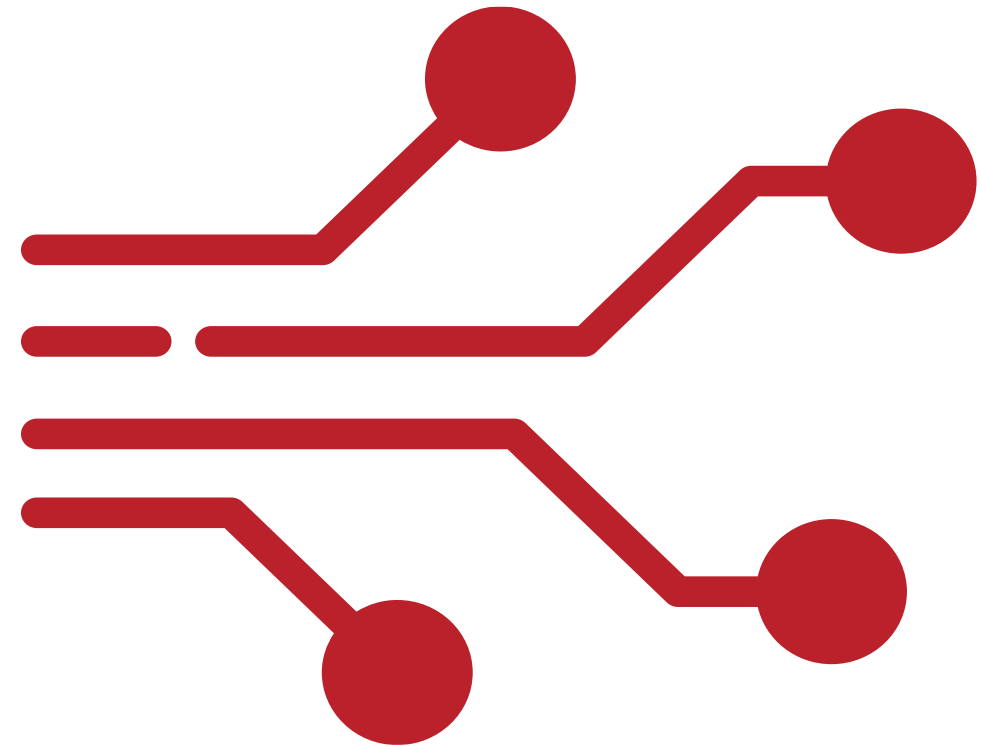
Język ten pozwala również na komunikację z serwerem, na przykład w celu pobierania lub wysyłania danych w tle, co jest podstawą działania współczesnych aplikacji webowych. Dodatkowo JavaScript obsługuje zdarzenia generowane przez użytkownika, takie jak kliknięcia, ruchy myszy czy wprowadzanie danych z klawiatury. Dzięki temu możliwe jest tworzenie interaktywnych, responsywnych i funkcjonalnych interfejsów użytkownika.

Osadzanie JavaScript w stronie

Kod JavaScript może być osadzany na stronie internetowej na kilka sposobów. Może znajdować się bezpośrednio w dokumencie HTML, na przykład wewnątrz odpowiedniego znacznika, lub w osobnych plikach z rozszerzeniem `.js`. Obie metody pozwalają na wykonywanie skryptów w przeglądarce, jednak różnią się organizacją i przeznaczeniem.

Najczęściej stosowanym rozwiązaniem są zewnętrzne pliki JavaScript. Takie podejście poprawia czytelność i porządek w kodzie, ponieważ oddziela logikę aplikacji od struktury dokumentu HTML. Dodatkowo umożliwia ponowne wykorzystanie tych samych skryptów na wielu stronach, ułatwia utrzymanie projektu oraz sprzyja lepszej organizacji większych aplikacji webowych.

Podstawy języka JavaScript



10

Zmienne i stałe

JavaScript umożliwia przechowywanie danych w zmiennych oraz stałych, które są podstawowymi elementami każdego programu. Zmienne służą do zapisywania informacji, które mogą ulegać zmianie w trakcie działania aplikacji, na przykład danych wprowadzanych przez użytkownika lub wyników obliczeń. Dzięki nim program może reagować na różne sytuacje i dynamicznie dostosowywać swoje działanie.

Stałe natomiast przechowują wartości, które po przypisaniu nie powinny być zmieniane. Stosuje się je do zapisu danych, które pozostają niezmiennie przez cały czas działania programu, takich jak ustawienia czy wartości konfiguracyjne. Rozróżnienie pomiędzy zmiennymi a stałymi pomaga tworzyć bardziej czytelny i bezpieczny kod, ograniczając ryzyko przypadkowych modyfikacji danych.

Typy danych

JavaScript obsługuje różne typy danych, które określają, jaki rodzaj informacji jest przechowywany w zmiennej oraz jakie operacje mogą być na niej wykonywane. Do podstawowych typów należą liczby, które służą do wykonywania obliczeń, napisy tekstowe wykorzystywane do przechowywania i przetwarzania tekstu oraz wartości logiczne, przyjmujące jedną z dwóch wartości: prawda lub fałsz.

Typ danych wpływa na sposób interpretacji i przetwarzania informacji przez program. Na przykład na liczbach można wykonywać operacje matematyczne, a na napisach łączenie lub modyfikację tekstu.

- **number** – liczby całkowite i zmiennoprzecinkowe
- **string** – ciągi znaków (tekst)
- **boolean** – wartości logiczne **true** / **false**
- **null** – oznaczenie braku wartości
- **undefined** – oznaczenie braku zdefiniowanej wartości
- **object** – obiekty złożone, takie jak tablice i obiekty

Rys. 1. Typy danych [źródło: <https://podstawyjs.pl/typy-danych-javascript/>]

Instrukcje warunkowe

Instrukcje warunkowe umożliwiają podejmowanie decyzji w programie na podstawie określonych warunków logicznych. Dzięki nim aplikacja może reagować w różny sposób w zależności od aktualnego stanu danych, na przykład wartości zmiennej, wyniku obliczeń czy działań użytkownika. Mechanizm ten sprawia, że program nie wykonuje zawsze tych samych instrukcji, lecz dostosowuje swoje działanie do sytuacji.

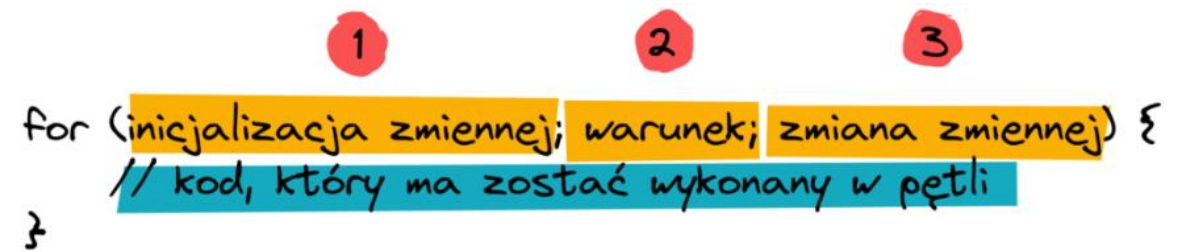
W praktyce instrukcje warunkowe pozwalają realizować różne scenariusze w zależności od wyniku operacji czy podejmowanie decyzji o dalszym przebiegu programu.

```
1  <?php
2  $liczba = 34;
3      if($liczba % 8 == 0){
4          echo 'podzielna przez 8';
5      }elseif($liczba % 4 == 0){
6          echo 'podzielna przez 4, ale nie przez 8';
7      }elseif($liczba % 2 == 0){
8          echo 'podzielna przez 2, ale nie przez 8 i nie przez 4';
9      }else{
10         echo 'ta liczba jest nieparzysta';
11     }
12  ?>
13
```

Rys. 2. Przykład instrukcji warunkowej [źródło:
<https://izabelamd.wordpress.com/programowanie/php/instrukcje-warunkowe/>]

Pętle

Pętle pozwalają na wielokrotne wykonywanie tego samego fragmentu kodu bez konieczności jego ręcznego powielania. Dzięki nim program może automatycznie powtarzać określone operacje, na przykład tyle razy, ile wynosi liczba elementów w zbiorze danych. W aplikacjach webowych pętle są często wykorzystywane do przetwarzania list danych, takich jak tablice wyników czy elementy pobrane z serwera, a także do dynamicznego generowania elementów strony. Pozwalają one tworzyć interfejsy, które dostosowują się do ilości i rodzaju danych, bez konieczności ręcznego definiowania każdego elementu



```
for (inicjalizacja zmiennej; warunek; zmiana zmiennej) {  
    // kod, który ma zostać wykonany w pętli  
}
```

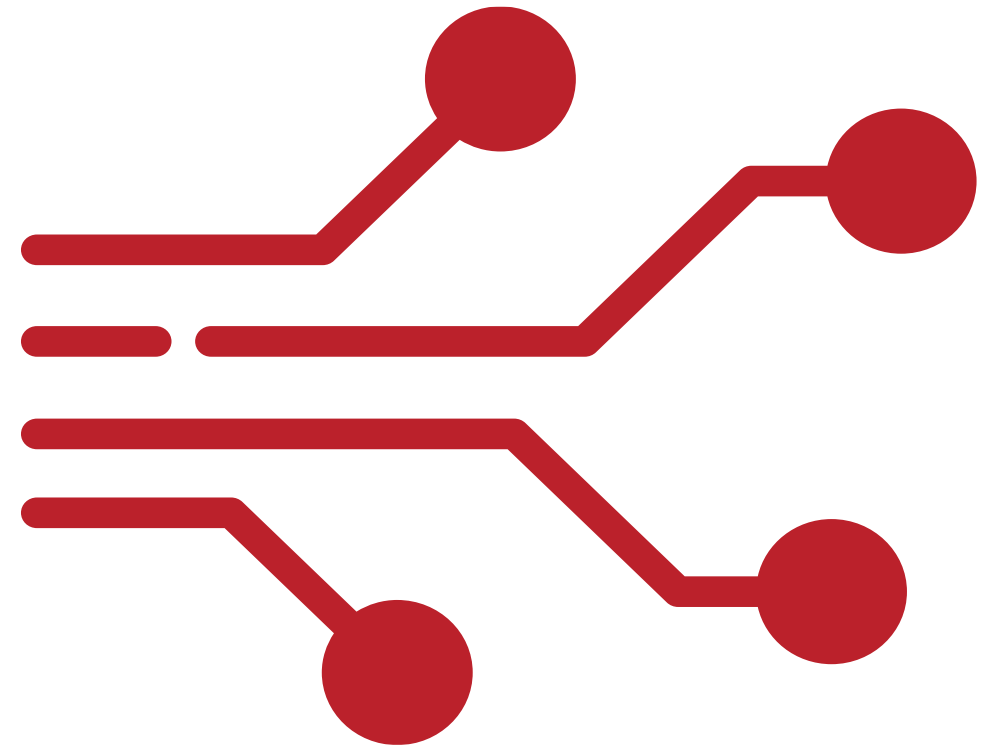
Rys. 3. Struktura pętli for [źródło: <https://podstawyjs.pl/petla-for-w-javascript/>]

Funkcje

Funkcje umożliwiają grupowanie instrukcji w logiczne bloki, które realizują konkretne zadania w programie. Zamiast wielokrotnie powtarzać ten sam kod w różnych miejscach, można go umieścić w funkcji i wywoływać w razie potrzeby. Takie podejście upraszcza strukturę programu i zmniejsza ryzyko błędów.

Stosowanie funkcji poprawia czytelność kodu, ponieważ poszczególne fragmenty programu mają jasno określoną odpowiedzialność. Umożliwia także ponowne wykorzystanie tej samej logiki w różnych częściach aplikacji, co jest szczególnie istotne w większych projektach webowych i sprzyja łatwiejszemu utrzymaniu oraz rozbudowie aplikacji.

Manipulacja strukturą dokumentu (DOM)



16

Obiektowy Model Dokumentu

Obiektowy Model Dokumentu (DOM) to sposób reprezentacji dokumentu HTML w przeglądarce w postaci struktury obiektów. Po załadowaniu strony przeglądarka analizuje kod HTML i tworzy z niego drzewo DOM, w którym każdy element, atrybut oraz fragment tekstu staje się obiektem posiadającym określone właściwości i metody. Struktura ta wiernie odzwierciedla hierarchię elementów znajdujących się w dokumencie.

Dzięki DOM JavaScript może wchodzić w bezpośrednią interakcję ze stroną internetową. Programista ma możliwość odczytywania zawartości elementów, zmieniania ich treści, stylów i atrybutów, a także dodawania lub usuwania elementów w trakcie działania strony. To właśnie Obiektowy Model Dokumentu stanowi podstawę dynamicznego działania aplikacji webowych i umożliwia reagowanie na działania użytkownika bez konieczności ponownego ładowania strony.

17

Dostęp do elementów DOM

JavaScript umożliwia dostęp do elementów DOM poprzez wyszukiwanie ich na podstawie identyfikatorów, klas, nazw znaczników lub relacji w strukturze dokumentu. Dzięki temu program może precyzyjnie wskazać konkretne elementy strony, z którymi ma pracować. Mechanizmy te pozwalają na elastyczne poruszanie się po strukturze dokumentu HTML i wybieranie zarówno pojedynczych elementów, jak i ich grup.

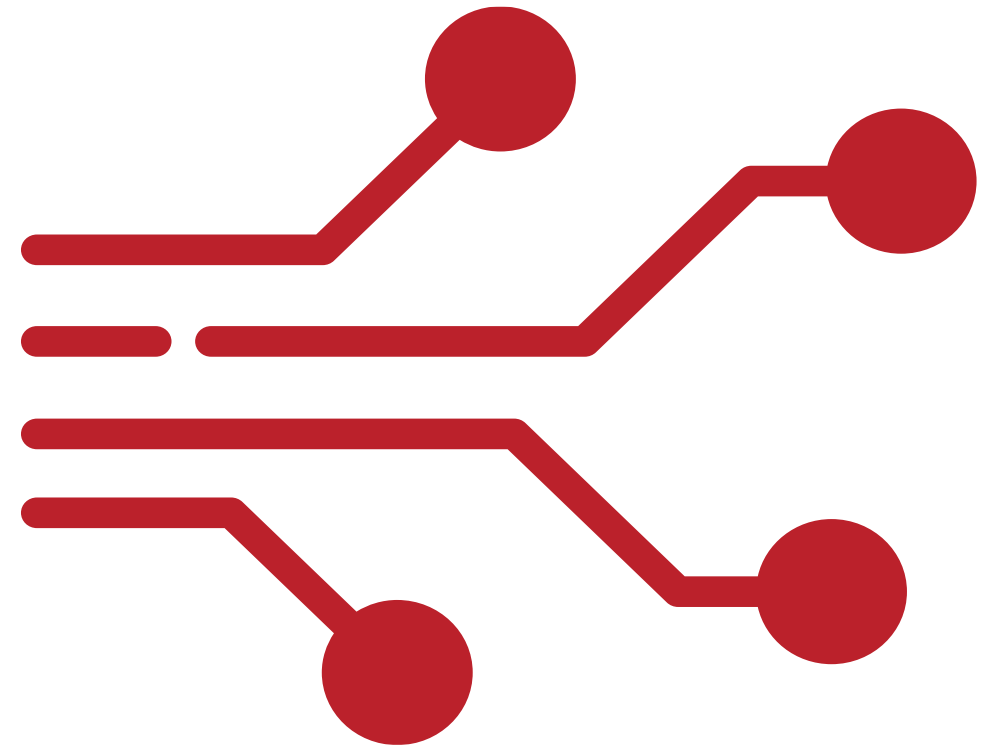
Uzyskany w ten sposób dostęp do elementów DOM umożliwia dynamiczną zmianę treści, modyfikowanie atrybutów oraz aktualizowanie stylów CSS w czasie działania strony. Dzięki temu aplikacja może reagować na działania użytkownika, aktualizować dane bez przeładowania strony oraz tworzyć interaktywne i responsywne interfejsy użytkownika.

Dynamiczna modyfikacja strony

Dynamiczna modyfikacja strony polega na zmianie jej zawartości w czasie rzeczywistym bez konieczności ponownego ładowania dokumentu. Dzięki manipulowaniu obiektami DOM JavaScript może aktualizować tekst, zmieniać atrybuty elementów, modyfikować style oraz reagować na działania użytkownika natychmiast po ich wykonaniu. Takie podejście znacząco zwiększa płynność działania aplikacji.

Możliwe jest również dodawanie nowych elementów do strony lub usuwanie już istniejących, na przykład podczas wyświetlania wyników wyszukiwania, aktualizacji listy danych czy obsługi komunikatów. Dynamiczna praca z DOM stanowi podstawę nowoczesnych interfejsów webowych i pozwala tworzyć aplikacje, które są interaktywne, responsywne i wygodne w użyciu.

Obsługa zdarzeń w aplikacjach webowych



20

Czym są zdarzenia?

Zdarzenia to sygnały wysyłane przez przeglądarkę, które informują o określonych działaniach użytkownika lub o zmianach stanu samej strony. Mogą one dotyczyć interakcji użytkownika, takich jak kliknięcia myszy, ruch kursora, wciśnięcia klawiszy czy wysłanie formularza, ale także zdarzeń systemowych, na przykład załadowania dokumentu lub zmiany rozmiaru okna przeglądarki.

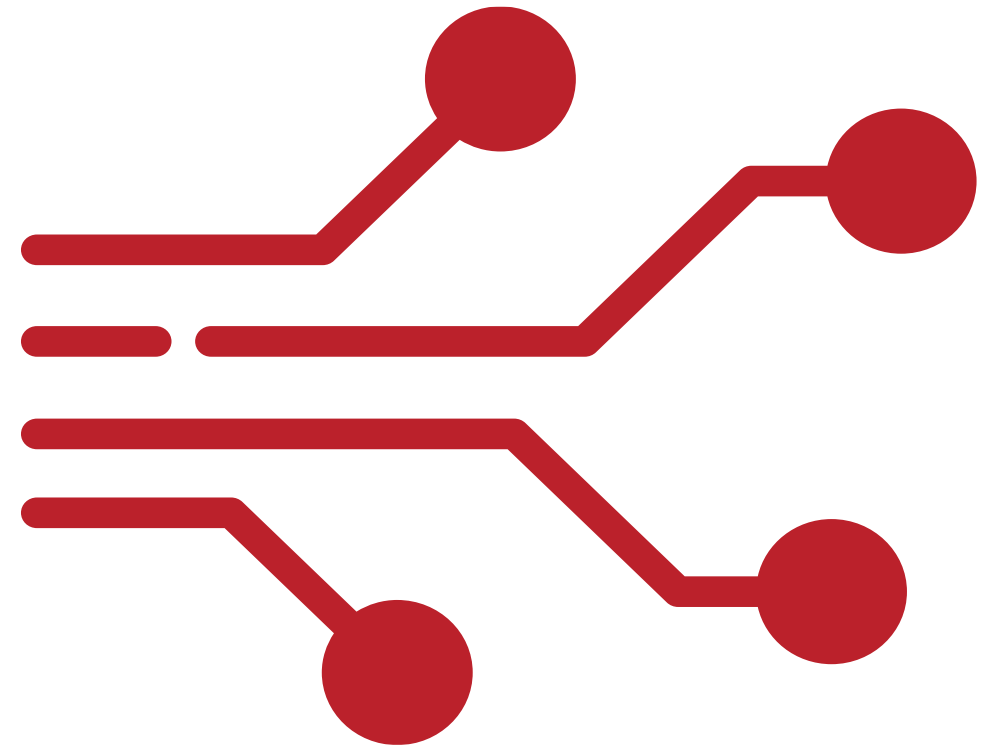
Zdarzenia stanowią podstawę interaktywności aplikacji webowych, ponieważ umożliwiają reagowanie na bieżące działania użytkownika. Dzięki nim JavaScript może wykonywać określone fragmenty kodu w odpowiedzi na konkretne sytuacje, co pozwala tworzyć dynamiczne, responsywne i przyjazne w obsłudze interfejsy użytkownika.

Reakcja na zdarzenia

JavaScript umożliwia reagowanie na zdarzenia poprzez przypisywanie do nich funkcji obsługujących, które są uruchamiane w momencie wystąpienia określonej akcji. Oznacza to, że program może „nasłuchiwać” zdarzeń generowanych przez użytkownika lub przeglądarkę i wykonywać odpowiedni kod dokładnie wtedy, gdy jest to potrzebne.

Dzięki obsłudze zdarzeń aplikacja webowa może sprawdzać poprawność danych wprowadzanych w formularzach, dynamicznie zmieniać wygląd interfejsu, wyświetlać komunikaty czy reagować na kliknięcia przycisków. Mechanizm ten jest podstawą interaktywności stron internetowych i pozwala tworzyć aplikacje, które aktywnie odpowiadają na działania użytkownika w czasie rzeczywistym.

Podsumowanie



23

Programowanie po stronie klienta umożliwia tworzenie dynamicznych i interaktywnych stron internetowych, które reagują na działania użytkownika w czasie rzeczywistym. JavaScript, jako podstawowy język skryptowy przeglądarek, pozwala na manipulowanie strukturą dokumentu poprzez DOM oraz na obsługę zdarzeń generowanych przez użytkownika i przeglądarkę. Dzięki temu możliwe jest budowanie nowoczesnych, responsywnych interfejsów, a zdobyta wiedza stanowi solidną podstawę do dalszej nauki frameworków frontendowych i zaawansowanych aplikacji webowych.

Literatura:

- [1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.
- [2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.
- [3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.
- [4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.
- [5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Dziękuję za uwagę

Wykład nr 7
Temat: Tworzenie
komponentów interfejsu
użytkownika. Dynamiczne
ładowanie treści.
Asynchroniczna wymiana
danych pomiędzy klientem a
serwerem. Animacje

dr hab. inż. Michał Wydra, prof. Uczelni

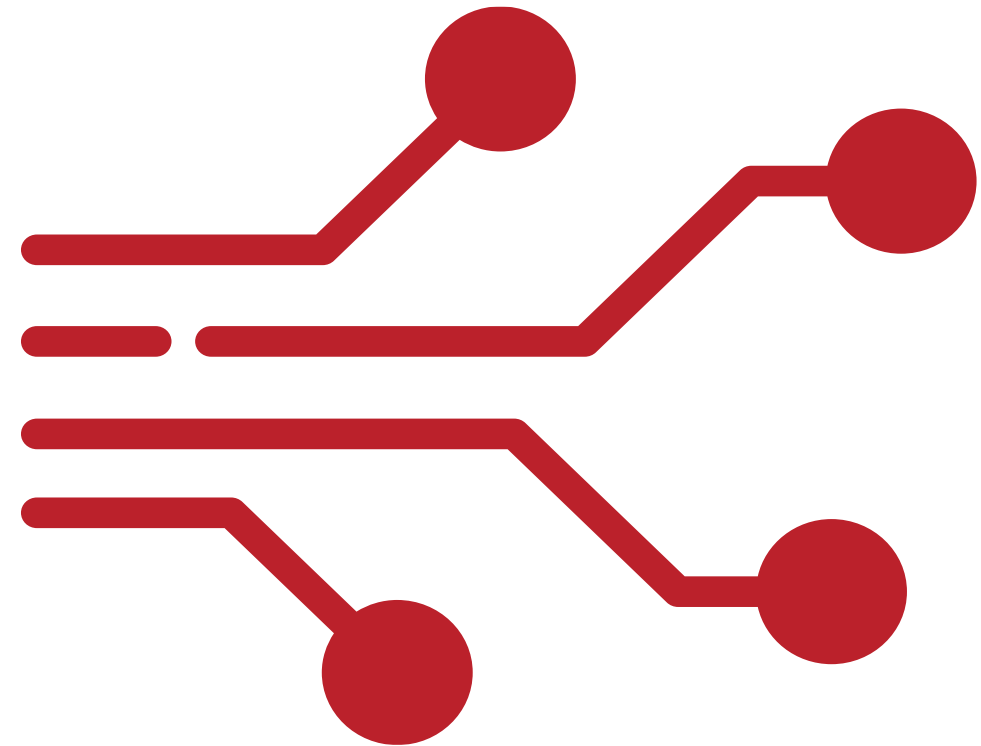
Katedra Informatyki Stosowanej

Plan prezentacji

- Komponenty interfejsu użytkownika
- Dynamiczne generowanie i aktualizacja treści
- Asynchroniczna komunikacja klient-serwer
- AJAX i Fetch API
- Animacje w aplikacjach internetowych



Komponenty interfejsu użytkownika



3

Interfejs użytkownika w aplikacjach webowych

Interfejs użytkownika to warstwa aplikacji odpowiedzialna za bezpośredni kontakt z użytkownikiem. To właśnie poprzez interfejs użytkownik wykonuje operacje, wprowadza dane oraz otrzymuje informacje zwrotne. W aplikacjach internetowych interfejs użytkownika tworzony jest z wykorzystaniem HTML, CSS oraz JavaScript i powinien być zaprojektowany w sposób czytelny, intuicyjny i spójny wizualnie.



Rys. 1. UI i UX [źródło: <https://udigroup.pl/blog/co-to-jest-interfejs-uzytkownika-user-interface/>]

Czym są komponenty UI i UX?

Komponent **interfejsu użytkownika UI** to wydzielony fragment interfejsu realizujący określoną funkcję, na przykład przycisk, formularz, lista lub tabela danych. Komponenty pozwalają porządkować strukturę interfejsu oraz ponownie wykorzystywać te same rozwiązania w różnych miejscach aplikacji, co ułatwia jej rozwój i utrzymanie.

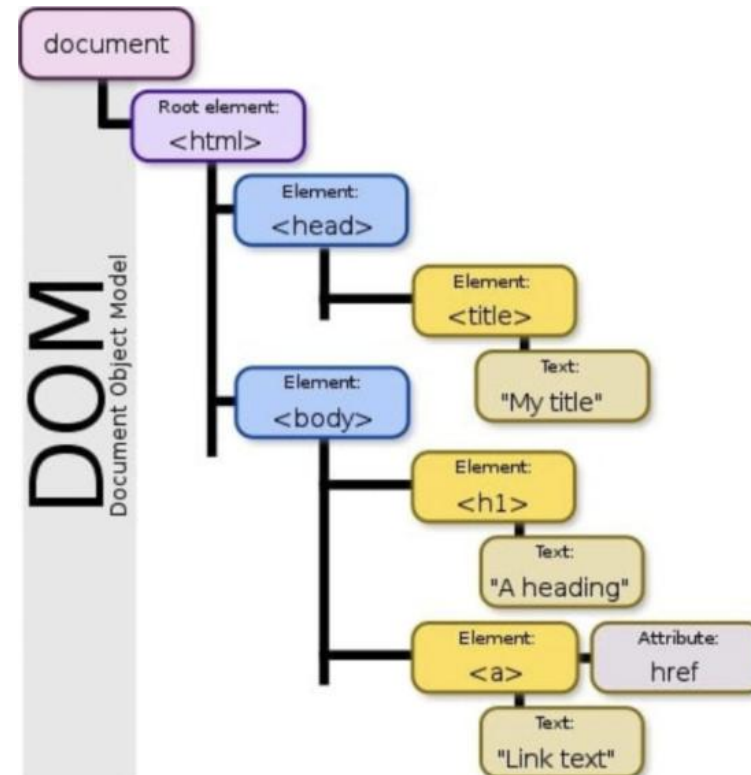
Projektowanie komponentów nie dotyczy jednak wyłącznie ich wyglądu. Równie istotne jest **UX (User Experience)**, czyli doświadczenie użytkownika podczas korzystania z aplikacji. UX koncentruje się na tym, czy interfejs jest intuicyjny, zrozumiały i wygodny w użyciu.

W praktyce oznacza to, że komponenty UI są nośnikiem UX – poprzez ich strukturę, rozmieszczenie i reakcje na działania użytkownika kształtowane jest ogólne wrażenie z korzystania z aplikacji.

5

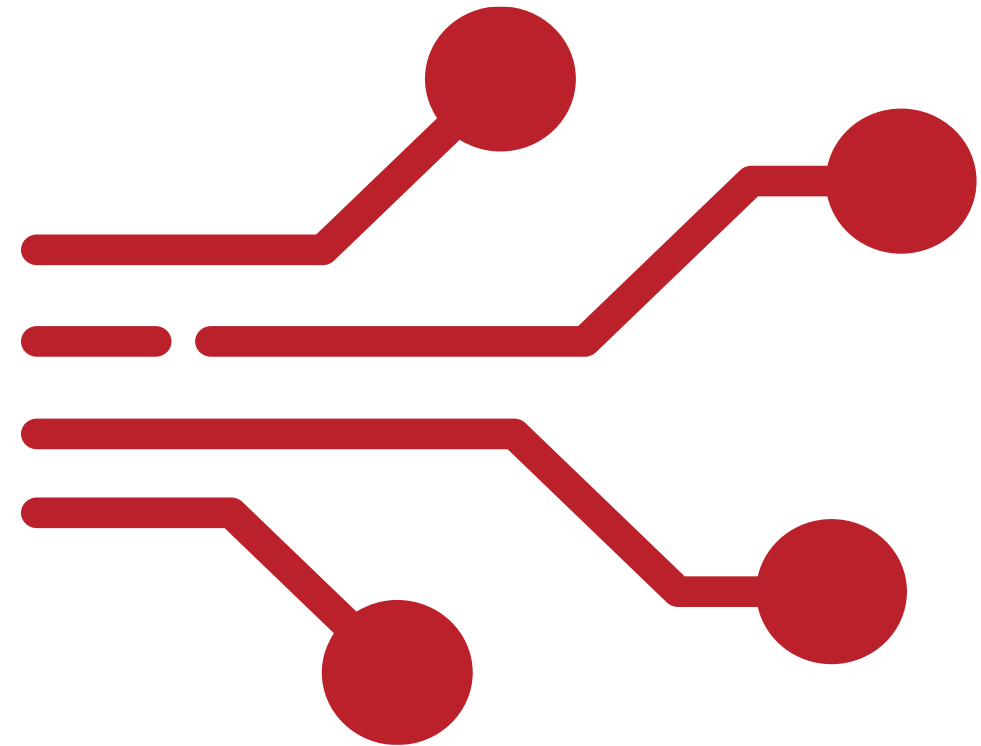
Komponenty a struktura DOM

Każdy komponent interfejsu użytkownika jest reprezentowany w przeglądarce jako zestaw elementów w strukturze DOM. JavaScript umożliwia dostęp do tych elementów, ich modyfikację oraz tworzenie nowych. Dzięki temu komponenty mogą być generowane dynamicznie w zależności od danych lub działań użytkownika, a nie tylko zapisane statycznie w kodzie HTML.



Rys. 2. Struktura DOM [źródło:
<https://digitallyher.pl/drzewo-dom-o-co-chodzi/>]

Dynamiczne generowanie i aktualizacja treści



Dynamiczna modyfikacja strony

Dynamiczna modyfikacja strony polega na zmianie jej zawartości lub wyglądu w czasie działania aplikacji, bez konieczności ponownego ładowania całego dokumentu w przeglądarce. Mechanizm ten jest jednym z fundamentów nowoczesnych aplikacji internetowych, ponieważ pozwala na natychmiastową reakcję interfejsu na działania użytkownika, takie jak kliknięcia, wpisywanie danych czy wybór opcji z formularza.

Za dynamiczną modyfikację strony odpowiada przede wszystkim JavaScript, który umożliwia dostęp do struktury dokumentu za pomocą DOM API. Programista może w ten sposób dodawać nowe elementy, usuwać istniejące, zmieniać ich treść, atrybuty lub style. Dzięki temu możliwe jest np. wyświetlanie komunikatów bez przeładowania strony, aktualizacja list danych, rozwijanie i zwijanie sekcji interfejsu czy walidacja formularzy w czasie rzeczywistym.

Dynamiczne aktualizacje mają również istotne znaczenie z punktu widzenia doświadczenia użytkownika. Ograniczenie przeładowań strony sprawia, że aplikacja działa szybciej i płynniej, a użytkownik zachowuje kontekst swojej pracy. W efekcie interfejs sprawia wrażenie bardziej responsywnego i nowoczesnego, co bezpośrednio wpływa na pozytywny odbiór aplikacji.

Komponenty a struktura DOM

DOM API udostępnia mechanizmy umożliwiające tworzenie nowych elementów HTML bezpośrednio z poziomu JavaScript. Programista może tworzyć elementy, ustawiać ich atrybuty, dodawać treść oraz wstawiać je w odpowiednie miejsce drzewa DOM. Pozwala to budować interfejs użytkownika w sposób dynamiczny, np. na podstawie danych pobranych z serwera.

```
const nameField = document.getElementById("userName");
const sendButton = document.getElementById("sendButton");

sendButton.disabled = true;
// [note: this is disabled since it causes this article to always load with
this example focused and scrolled into view]
// nameField.focus();

nameField.addEventListener("input", (event) => {
  const elem = event.target;
  const valid = elem.value.length !== 0;

  if (valid && sendButton.disabled) {
    sendButton.disabled = false;
  } else if (!valid && !sendButton.disabled) {
    sendButton.disabled = true;
  }
});
```

Rys. 3. Przykład wykorzystania DOM API [źródło:
https://developer.mozilla.org/en-US/docs/Web/API/HTML_DOM_API#examples]

9

Aktualizacja zawartości komponentów

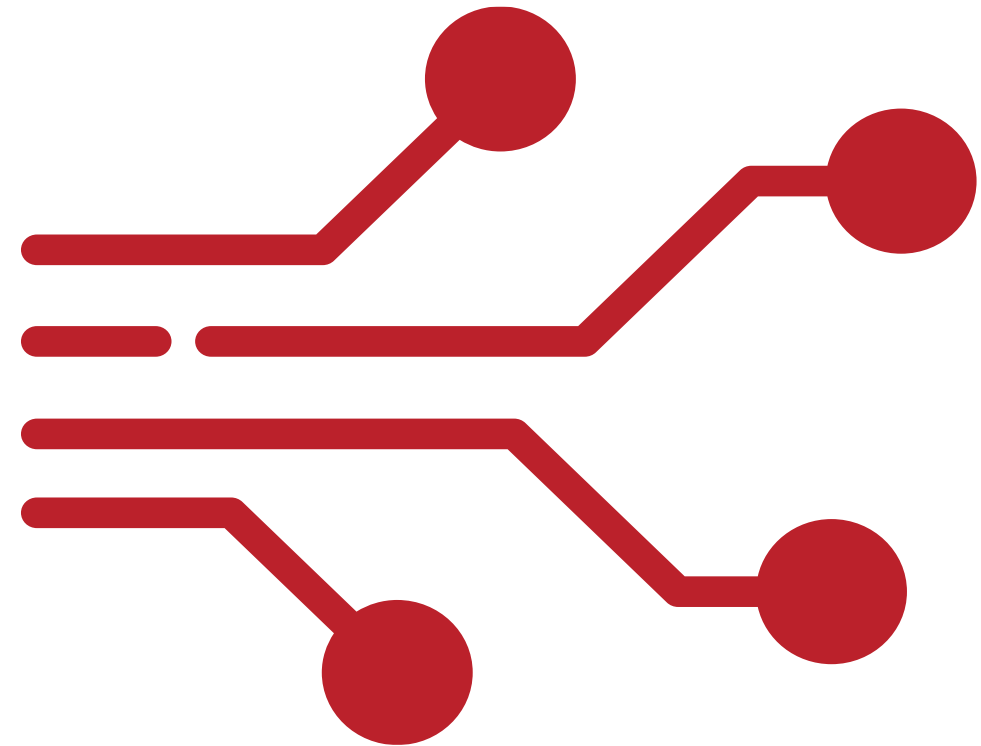
Aktualizacja zawartości komponentów interfejsu użytkownika polega na dynamicznej zmianie informacji wyświetlanych na stronie w odpowiedzi na działania użytkownika lub dane otrzymane z serwera. Zamiast generować nową stronę, aplikacja modyfikuje jedynie te elementy interfejsu, które wymagają zmiany, co znacząco poprawia płynność i komfort pracy użytkownika.

JavaScript, wykorzystując DOM API, umożliwia zmianę treści tekstowej komponentów, ich atrybutów oraz struktury wewnętrznej. Przykładem może być aktualizacja wyników wyszukiwania po wpisaniu zapytania, wyświetlanie komunikatów o błędach walidacji formularza, zmiana stanu przycisków czy odświeżanie zawartości tabel na podstawie nowych danych. Tego typu operacje wykonywane są w czasie rzeczywistym i nie wymagają przeładowania całej strony.

Dynamiczna aktualizacja zawartości komponentów ma także istotne znaczenie z punktu widzenia użyteczności i doświadczenia użytkownika. Użytkownik otrzymuje natychmiastową informację zwrotną, co zwiększa czytelność interfejsu i ogranicza liczbę błędów. Odpowiednio zaprojektowane aktualizacje treści pomagają użytkownikowi zrozumieć, co dzieje się w aplikacji, oraz ułatwiają poruszanie się pomiędzy jej funkcjami.

10

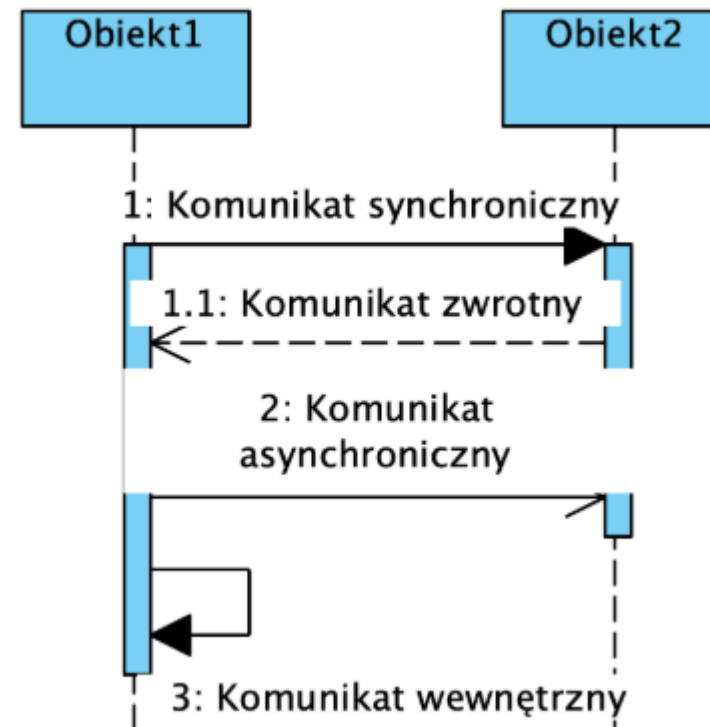
Asynchroniczna komunikacja klient-serwer



11

Komunikacja synchroniczna i asynchroniczna

W komunikacji synchronicznej przeglądarka wysyła żądanie do serwera i czeka na odpowiedź, blokując dalsze działanie aplikacji. W podejściu asynchronicznym żądania są wysyłane w tle, a aplikacja może nadal reagować na działania użytkownika. Asynchroniczność jest kluczowa dla płynnego działania nowoczesnych aplikacji webowych.



Rys. 4. Schemat komunikacji synchronicznej i asynchronicznej [źródło: <https://www.nietylkoprogramowanie.pl/komunikacja-synchroniczna-i-asynchroniczna/>]

12

Idea asynchronicznej wymiany danych

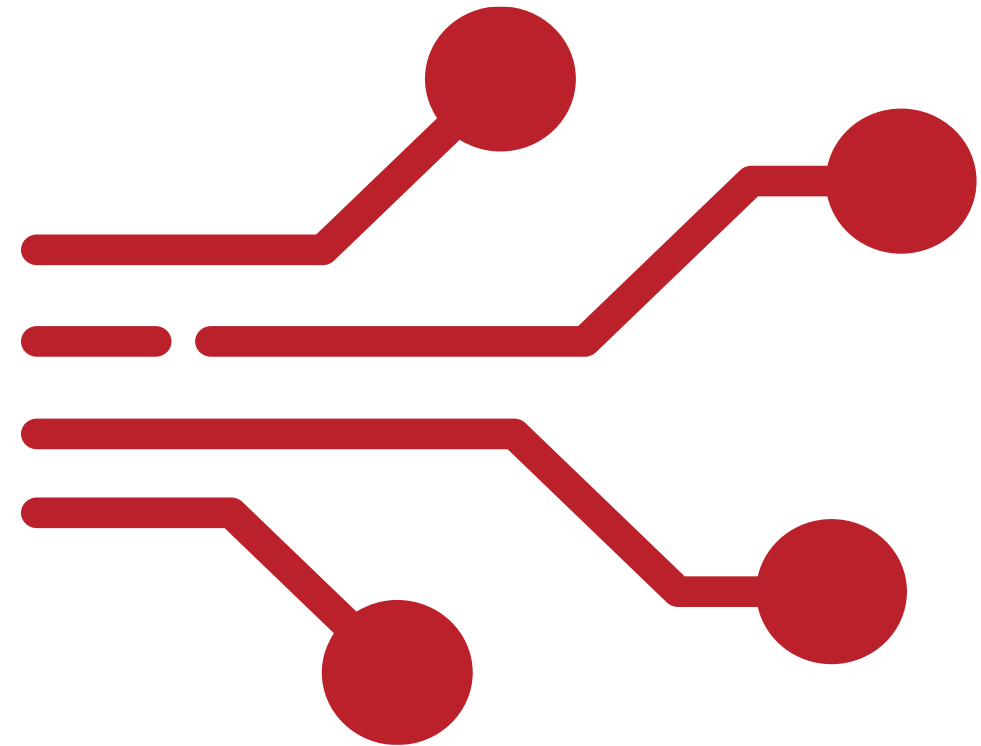
Asynchroniczna wymiana danych polega na przesyłaniu informacji pomiędzy klientem a serwerem w taki sposób, że aplikacja nie musi wstrzymywać swojego działania na czas oczekiwania na odpowiedź. W przeciwieństwie do komunikacji synchronicznej, w której użytkownik jest zmuszony czekać na zakończenie operacji, podejście asynchroniczne pozwala na dalszą interakcję z interfejsem w trakcie realizacji zapytania.

W praktyce oznacza to, że aplikacja może wysyłać żądania do serwera w tle, a po otrzymaniu odpowiedzi jedynie zaktualizować wybrane fragmenty interfejsu użytkownika. Dzięki temu możliwe jest np. ładowanie dodatkowych danych podczas przewijania strony, pobieranie wyników wyszukiwania bez przeładowania widoku czy wysyłanie danych formularza bez utraty aktualnego stanu aplikacji.

Asynchroniczna wymiana danych ma kluczowe znaczenie dla wydajności i użyteczności nowoczesnych aplikacji internetowych. Pozwala tworzyć interfejsy, które są bardziej responsywne i sprawiają wrażenie „ciągłości działania”, co znacząco poprawia doświadczenie użytkownika. Jednocześnie wymaga od programisty odpowiedniego zarządzania stanem aplikacji oraz obsługi sytuacji, w których odpowiedź z serwera pojawia się z opóźnieniem lub nie pojawia się wcale.

13

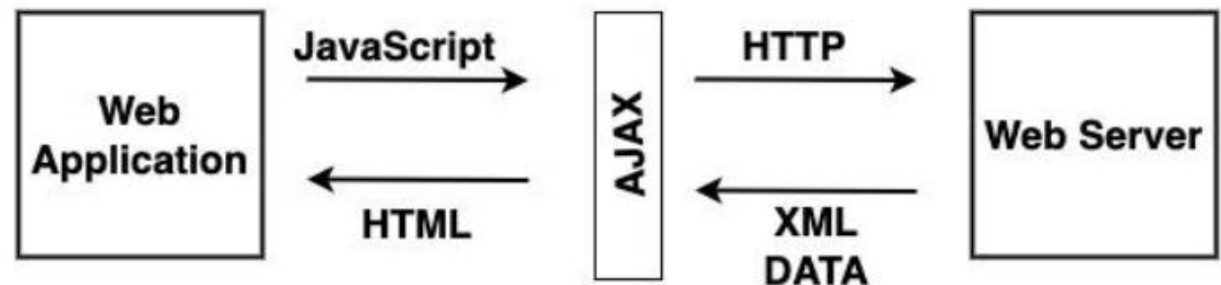
AJAX i Fetch API



14

Czym jest AJAX?

AJAX to technika polegająca na wykonywaniu asynchronicznych zapytań HTTP z poziomu JavaScript. Umożliwia ona pobieranie danych z serwera i aktualizowanie wybranych fragmentów strony bez jej przeładowania. AJAX był jednym z kluczowych kroków w rozwoju nowoczesnych aplikacji internetowych



Rys. 5. Model AJAX [źródło:
https://www.tutorialspoint.com/ajax/what_is_ajax.htm]

Dane przesyłane między klientem a serwerem

W aplikacjach internetowych dane są nieustannie wymieniane pomiędzy przeglądarką użytkownika a serwerem. Dane te mogą dotyczyć treści wyświetlanych w interfejsie, informacji wprowadzanych przez użytkownika, wyników obliczeń czy komunikatów systemowych. Sposób przesyłania i reprezentacji danych ma bezpośredni wpływ na wydajność, czytelność oraz bezpieczeństwo aplikacji.

Współczesne aplikacje webowe najczęściej wykorzystują format JSON do przesyłania danych pomiędzy klientem a serwerem. Jest to format tekstowy, który w czytelny sposób reprezentuje struktury danych, takie jak obiekty czy tablice. JSON jest lekki, łatwy do parsowania w JavaScript i niezależny od konkretnej technologii backendowej, co czyni go standardem w komunikacji webowej.

Po stronie klienta dane otrzymane z serwera są przetwarzane przez JavaScript i wykorzystywane do aktualizacji interfejsu użytkownika lub logiki aplikacji. Mogą one posłużyć do wyświetlania list, tabel, komunikatów czy stanów komponentów. Z kolei dane wysyłane z przeglądarki do serwera, np. z formularzy, są podstawą dalszego przetwarzania po stronie backendu.

16

Fetch API

Fetch API to nowoczesny interfejs JavaScript służący do komunikacji z serwerem. Zapewnia prostą i czytelną składnię oraz natywną obsługę asynchroniczności. Fetch API jest obecnie preferowanym sposobem wykonywania zapytań HTTP w aplikacjach webowych.

```
fetch('https://reqres.in/api/users', {  
  method: 'POST',  
  headers: {  
    'Content-type': 'application/json'  
  },  
  body: JSON.stringify({  
    name: 'Sankalan'  
  })  
})  
  
.then(res => res.json())  
.then(data => console.log(data));
```

Rys. 6. Modyfikacja danych z wykorzystaniem Fetch() [źródło: <https://sankalan.hashnode.dev/using-fetch-method-javascript>]

17

Dynamiczne ładowanie treści

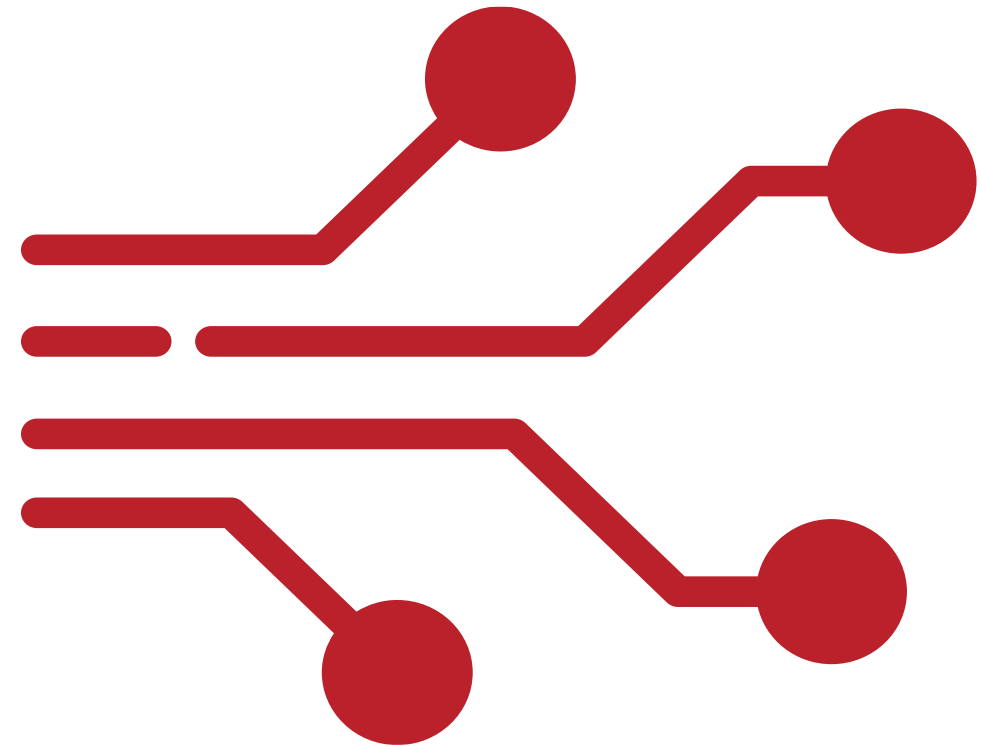
Dynamiczne ładowanie treści polega na pobieraniu danych z serwera tylko w momencie, gdy są one faktycznie potrzebne, bez konieczności przeładowania całej strony internetowej. Mechanizm ten pozwala aplikacji reagować na działania użytkownika w sposób płynny i naturalny, bez przerywania pracy i utraty aktualnego kontekstu. Dzięki temu interfejs sprawia wrażenie szybkiego i nowoczesnego.

W praktyce dynamiczne ładowanie treści jest realizowane za pomocą asynchronicznych zapytań HTTP wykonywanych z poziomu JavaScript, najczęściej z wykorzystaniem Fetch API. Po otrzymaniu odpowiedzi z serwera aplikacja przetwarza dane, a następnie aktualizuje odpowiednie komponenty interfejsu użytkownika, takie jak listy, tabele czy panele informacyjne. Cały proces odbywa się w tle i jest niemal niezauważalny dla użytkownika.

Dynamiczne ładowanie treści stanowi podstawę działania aplikacji typu Single Page Application, w których cała aplikacja działa w obrębie jednego dokumentu HTML. Takie podejście pozwala ograniczyć liczbę zapytań o pełne strony, zmniejszyć obciążenie serwera oraz poprawić ogólną wydajność aplikacji.

18

Animacje w aplikacjach internetowych



19

Rola animacji w UI

Animacje w interfejsach użytkownika pełnią znacznie większą rolę niż wyłącznie funkcję estetyczną. Ich głównym zadaniem jest wspieranie komunikacji pomiędzy aplikacją a użytkownikiem poprzez wizualne przedstawienie zmian zachodzących w systemie. Animacje pomagają użytkownikowi zrozumieć, co właśnie się wydarzyło, jaki jest efekt jego działania oraz jaki jest aktualny stan aplikacji.

Dobrze zaprojektowane animacje mogą wskazywać przejścia pomiędzy różnymi widokami, podkreślać aktywne elementy interfejsu lub sygnalizować rozpoczęcie i zakończenie określonego procesu, na przykład ładowania danych. Dzięki temu użytkownik nie ma wrażenia „nagłych” zmian na stronie, a interfejs wydaje się bardziej spójny i przewidywalny. Animacje pełnią więc funkcję informacyjną i porządkującą, a nie tylko dekoracyjną.

Animacje z wykorzystaniem CSS

CSS umożliwia tworzenie animacji i przejść pomiędzy różnymi stanami elementów. Animacje CSS są wydajne i często stosowane do prostych efektów wizualnych, takich jak zmiana koloru, położenia lub przezroczystości elementów. Są one łatwe w implementacji i nie wymagają dodatkowego kodu JavaScript.

```
div {  
  animation-name: example;  
  animation-duration: 5s;  
  animation-timing-function: linear;  
  animation-delay: 2s;  
  animation-iteration-count: infinite;  
  animation-direction: alternate;  
}
```

Rys. 7. Właściwość skrócona animacji [źródło: <https://rrogacz.pl/css-animacje>]

Animacje sterowane JavaScript

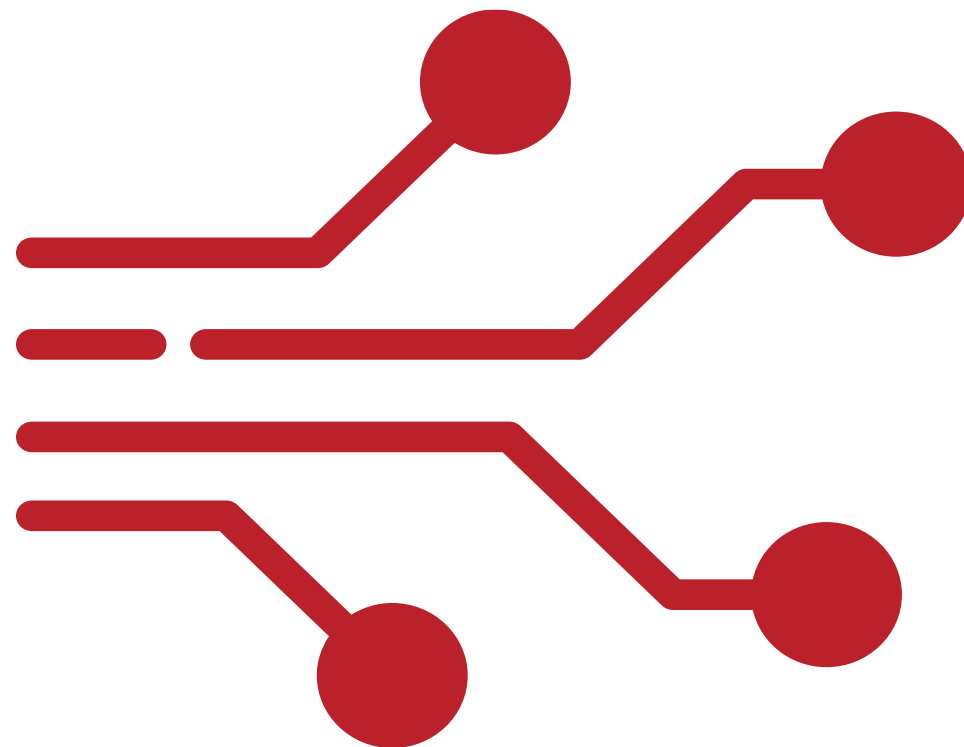
Animacje sterowane za pomocą JavaScript umożliwiają znacznie większą kontrolę nad zachowaniem elementów interfejsu niż animacje definiowane wyłącznie w CSS. JavaScript pozwala uruchamiać, zatrzymywać i modyfikować animacje w odpowiedzi na konkretne zdarzenia, takie jak kliknięcia użytkownika, ruch myszy, przewijanie strony czy zakończenie operacji asynchronicznej, na przykład pobierania danych z serwera.

Dzięki wykorzystaniu JavaScript animacje mogą być ściśle powiązane z logiką aplikacji oraz jej aktualnym stanem. Przykładowo, animacja może sygnalizować rozpoczęcie procesu ładowania danych, stopniowo ujawniać nowe elementy interfejsu po otrzymaniu odpowiedzi z serwera lub wizualnie informować o błędach i komunikatach systemowych.

Należy jednak pamiętać, że animacje sterowane JavaScript wymagają świadomego projektowania pod kątem wydajności. Nadmierna liczba animacji lub nieoptymalne operacje na DOM mogą prowadzić do spadku płynności interfejsu. Dlatego animacje powinny być stosowane tam, gdzie realnie wspierają użyteczność i zrozumienie działania aplikacji, a nie wyłącznie w celach wizualnych.

22

Podsumowanie



23

Tworzenie nowoczesnych aplikacji internetowych opiera się na dynamicznych komponentach interfejsu użytkownika, które są generowane i aktualizowane w czasie rzeczywistym. Dzięki asynchronicznej komunikacji z serwerem możliwe jest pobieranie i wysyłanie danych bez przerywania pracy użytkownika, co znacząco poprawia płynność i responsywność aplikacji. Mechanizmy takie jak DOM API, AJAX oraz Fetch API stanowią fundament dynamicznego ładowania treści i interakcji z backendem.

Uzupełnieniem tych technik są animacje, które wspierają komunikację wizualną pomiędzy aplikacją a użytkownikiem. Odpowiednio zaprojektowane animacje pomagają zrozumieć zmiany zachodzące w interfejsie, poprawiają doświadczenie użytkownika oraz zwiększają czytelność aplikacji. Połączenie komponentowego podejścia do UI, asynchronicznej wymiany danych oraz animacji stanowi podstawę tworzenia współczesnych, interaktywnych i użytecznych aplikacji webowych.

Literatura:

- [1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.
- [2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.
- [3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.
- [4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.
- [5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Dziękuję za uwagę

Wykład nr 8
Temat: Projektowanie interfejsów
użytkownika. Dostosowanie stron
do różnych urządzeń (RWD), ich
elastyczność i dostępność.
Udoskonalanie witryn pod kątem
wyszukiwarek oraz zasady SEO.
Wykorzystanie narzędzi
programistycznych dostępnych w
przeglądarkach

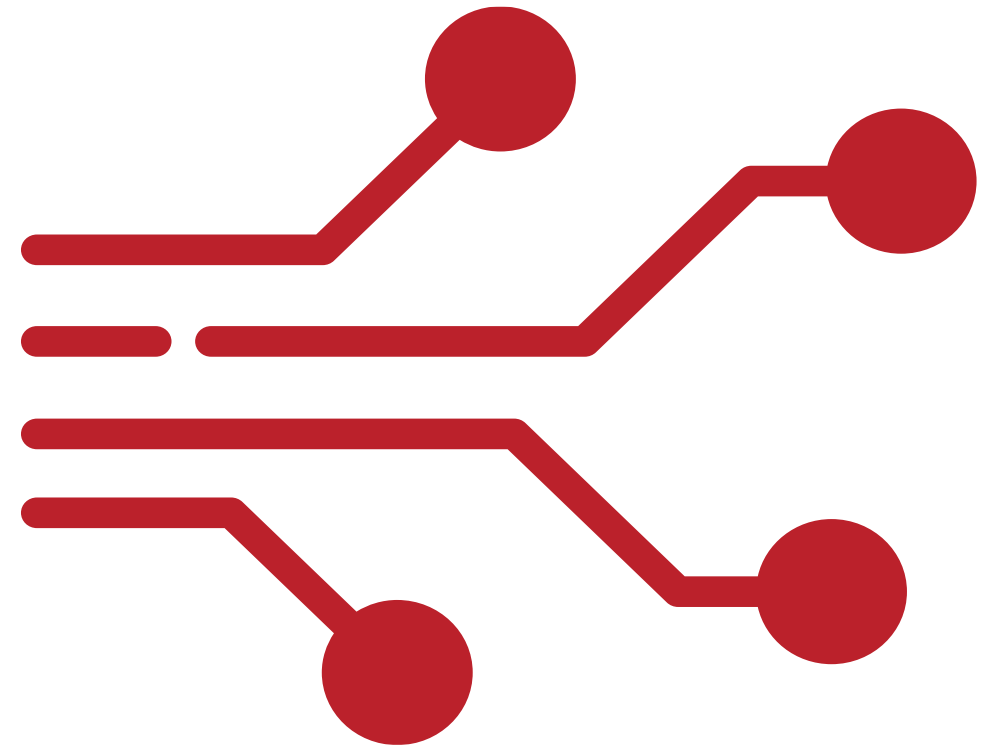
dr hab. inż. Michał Wydra, prof. Uczelni
Katedra Informatyki Stosowanej

Plan prezentacji

- Projektowanie UI i UX
- Responsywność i elastyczność stron (RWD)
- Dostępność cyfrowa i jej znaczenie
- Optymalizacja stron pod kątem SEO
- Narzędzia programistyczne w przeglądarkach



Projektowanie UI i UX

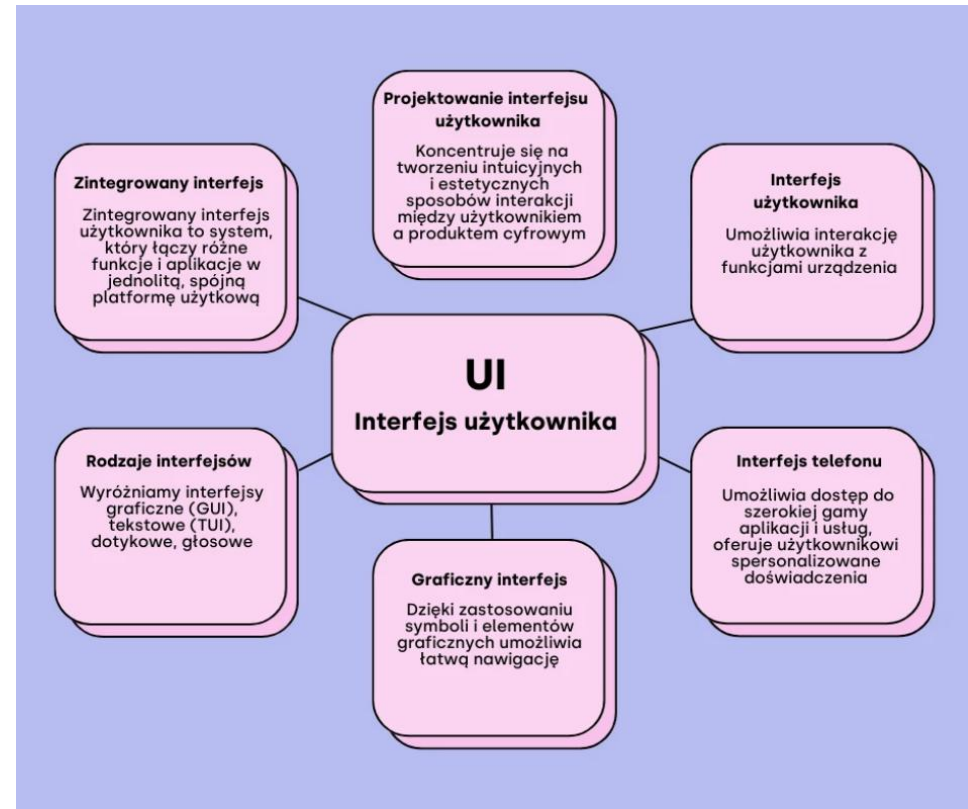


3

Projektowanie interfejsu użytkownika UI

Interfejs użytkownika (UI) obejmuje wszystkie elementy wizualne strony internetowej, z którymi użytkownik wchodzi w bezpośrednią interakcję. Są to m.in. przyciski, formularze, menu, ikony, kolory oraz układ treści. Dobrze zaprojektowany interfejs powinien być spójny, czytelny i przewidywalny, tak aby użytkownik intuicyjnie rozumiał sposób działania strony bez potrzeby dodatkowych instrukcji.

Projektowanie UI wymaga nie tylko znajomości technologii webowych, ale także zasad estetyki, hierarchii wizualnej i ergonomii. Celem jest stworzenie interfejsu, który wspiera użytkownika w realizacji jego celu, a nie utrudnia korzystanie z aplikacji.

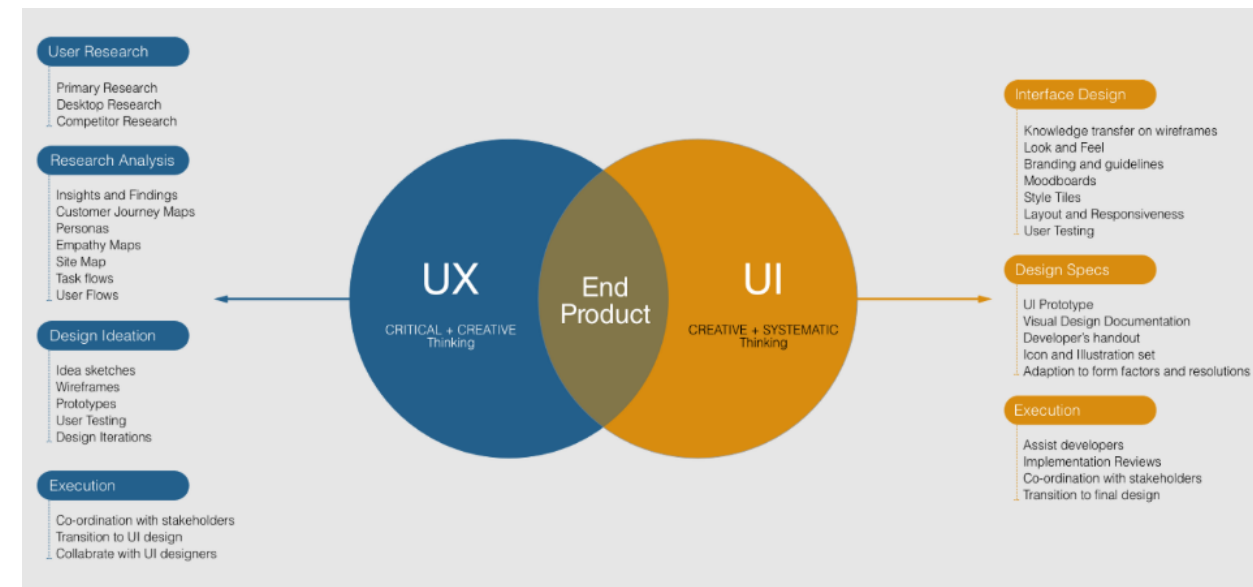


Rys. 1. UI [źródło: <https://1stplace.pl/blog/interfejs-uzytkownika-ux/>]

UX jako uzupełnienie UI

UX (User Experience) odnosi się do całokształtu doświadczeń użytkownika podczas korzystania ze strony internetowej. Obejmuje ono łatwość nawigacji, szybkość działania, zrozumiałość komunikatów oraz ogólny komfort użytkowania. Nawet estetyczny interfejs UI może być źle oceniany, jeśli korzystanie z niego jest nieintuicyjne lub frustrujące.

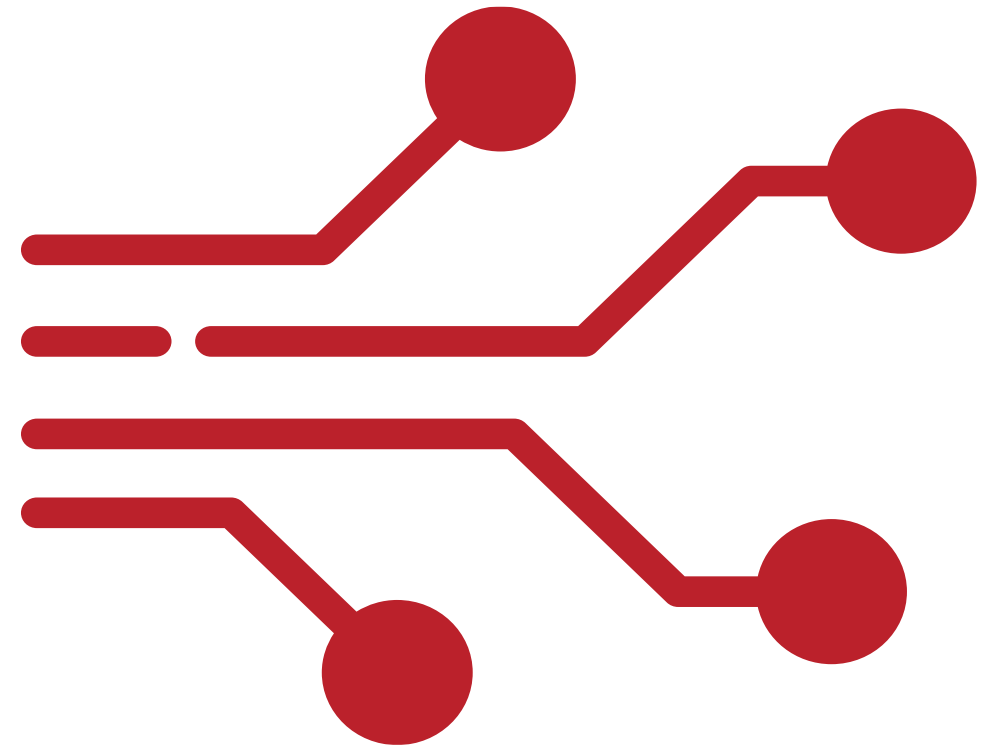
Projektowanie UX polega na przewidywaniu zachowań użytkownika i dostosowywaniu interfejsu do jego potrzeb. W praktyce oznacza to m.in. logiczną strukturę strony, jasne komunikaty, odpowiednie rozmieszczenie elementów oraz konsekwentne zachowanie interfejsu w całej witrynie.



Rys. 2. Różnice między UX a UI [źródło: <https://www.learndesign.pl/blog/czy-ux-i-ui-to-to-samo-i-dlaczego-nie>]

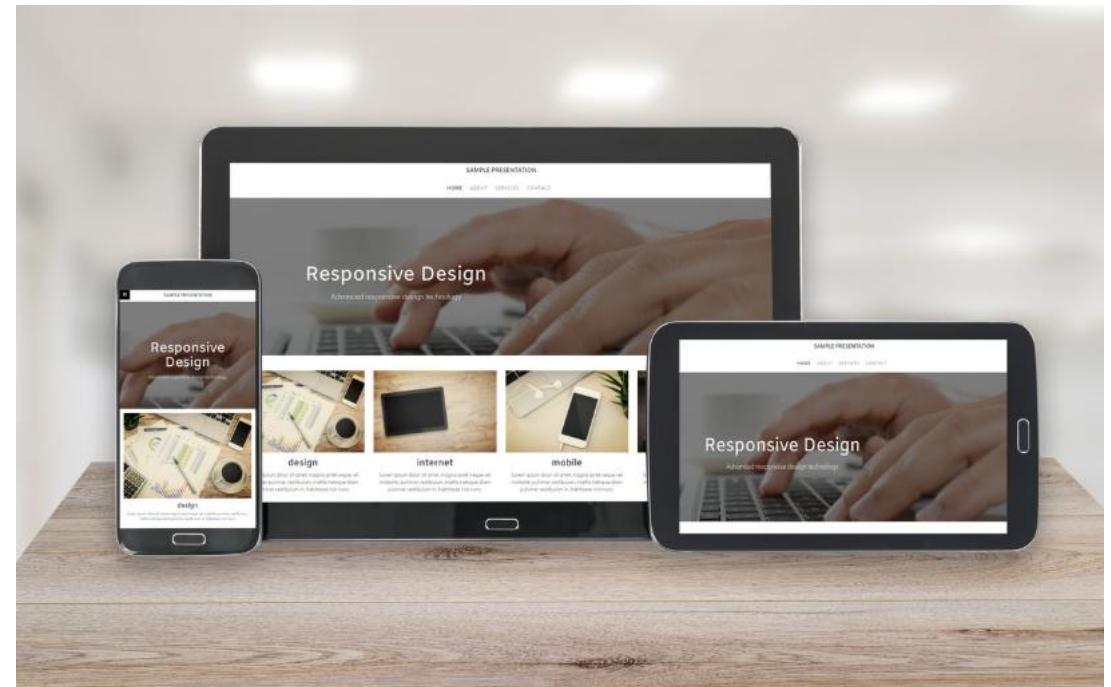
5

Responsywność i elastyczność stron (RWD)



Różnorodność urządzeń i wyzwania projektowe

Współczesne strony internetowe są wyświetlane na bardzo różnych urządzeniach: komputerach stacjonarnych, laptopach, tabletach, smartfonach, a nawet telewizorach czy zegarkach. Różnice w rozdzielczości ekranu, proporcjach i sposobie interakcji powodują, że projektowanie jednej „sztywnej” wersji strony przestaje być wystarczające. Projektant musi brać pod uwagę zarówno wielkość ekranu, jak i sposób obsługi – np. dotyk zamiast myszy. To właśnie ta różnorodność doprowadziła do rozwoju koncepcji responsywnego projektowania stron internetowych.

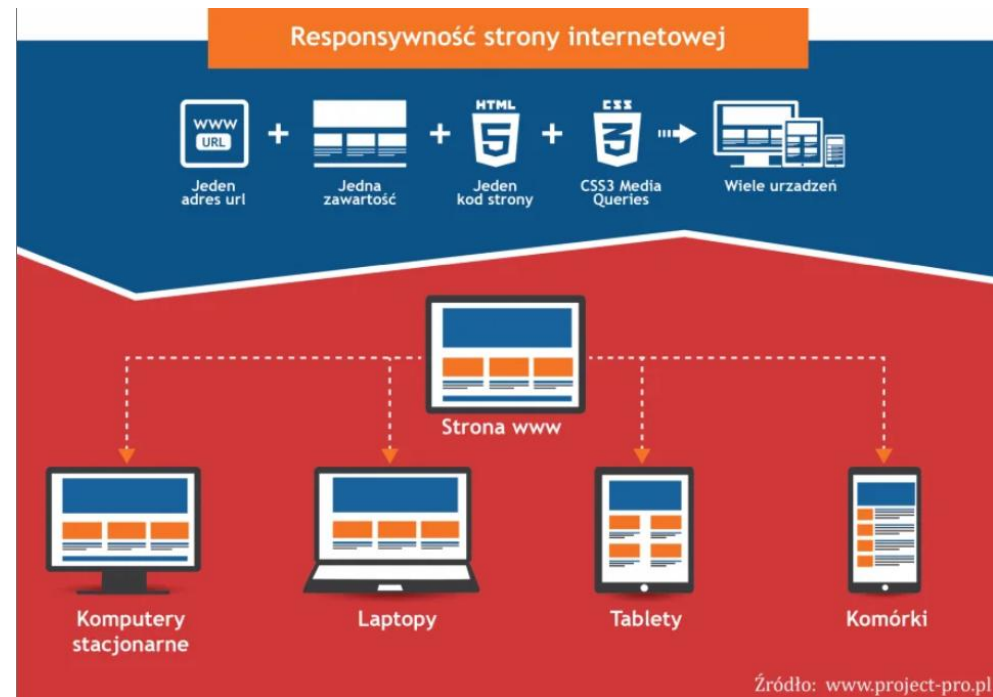


Rys. 3. Wyświetlanie strony na różnych urządzeniach [źródło: <https://famatech.pl/blog/projektowanie-stron-www-co-musisz-o-tym-wiedziec>]

7

Responsywne projektowanie stron (RWD)

Responsywne projektowanie stron internetowych (Responsive Web Design) polega na tworzeniu witryn, które automatycznie dostosowują swój wygląd i układ do rozmiaru ekranu użytkownika. Zamiast tworzyć osobne wersje strony dla różnych urządzeń, stosuje się jeden elastyczny projekt, który reaguje na warunki wyświetlania. RWD opiera się na takich technikach jak elastyczne siatki, media queries oraz responsywne obrazy. Dzięki temu strona pozostaje czytelna i funkcjonalna niezależnie od tego, czy jest oglądana na dużym monitorze czy na ekranie smartfona.



Rys. 4. Responsywność strony [źródło: <https://www.project-pro.pl/responsywnosc-strony-internetowej-czym-jest-i-dlaczego-jest-tak-wazna-przy-prowadzeniu-biznesu>]

8

Elastyczność układu strony

Elastyczność strony oznacza zdolność jej elementów do płynnego dopasowywania się do dostępnej przestrzeni. Zamiast stosować sztywne wartości w pikselach, wykorzystuje się jednostki względne, takie jak procenty, viewport units czy em/rem. Pozwala to zachować proporcje elementów i poprawną czytelność treści.

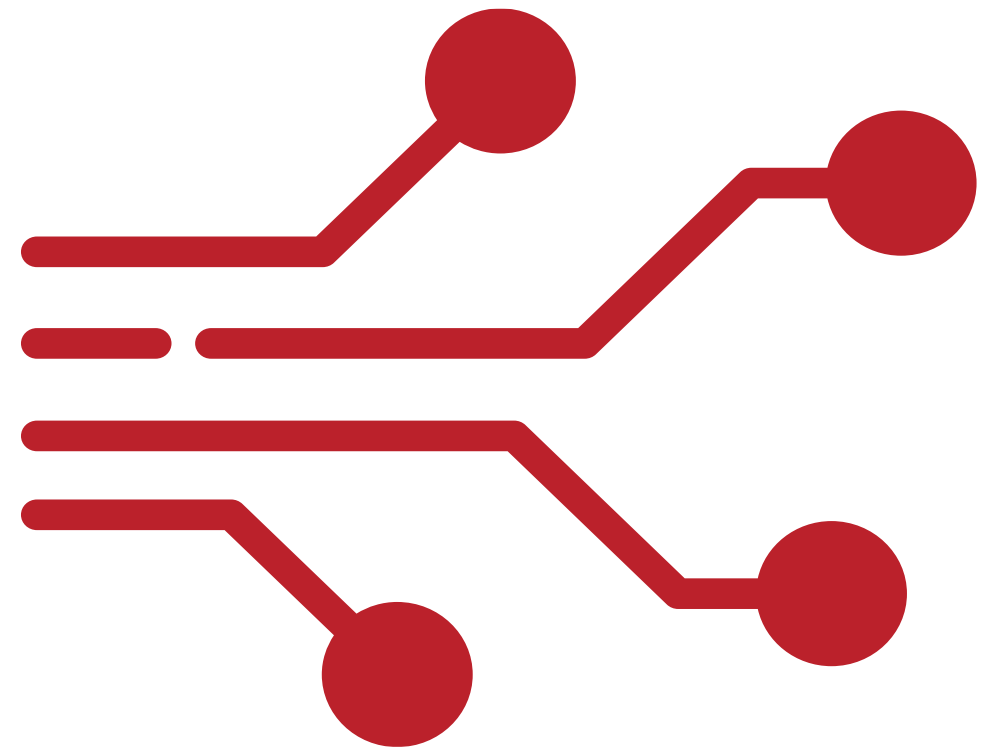
Elastyczny układ zwiększa trwałość projektu – strona lepiej znosi zmiany rozdzielczości, skalowanie tekstu oraz przyszłe urządzenia o niestandardowych parametrach.



Rys. 5. Elastyczność strony [źródło:
<https://designsolutions.pl/2012/08/16/10-cech-dobrej-firmowej-strony-internetowej/>]

9

Dostępność cyfrowa i jej znaczenie



10

Dostępność stron internetowych

Dostępność cyfrowa oznacza projektowanie stron internetowych w taki sposób, aby mogły z nich korzystać również osoby z różnymi niepełnosprawnościami. Dotyczy to m.in. użytkowników niewidomych, słabowidzących, z ograniczeniami ruchowymi czy problemami poznawczymi.

Tworzenie dostępnych stron nie jest wyłącznie kwestią etyki, ale coraz częściej także wymogiem prawnym. Co istotne, rozwiązania poprawiające dostępność zazwyczaj zwiększają także ogólną użyteczność strony dla wszystkich użytkowników.

Zasady dostępności (WCAG)

Elastyczność strony oznacza zdolność jej elementów do płynnego dopasowywania się do dostępnej przestrzeni. Zamiast stosować sztywne wartości w pikselach, wykorzystuje się jednostki względne, takie jak procenty, viewport units czy em/rem. Pozwala to zachować proporcje elementów i poprawną czytelność treści.

Elastyczny układ zwiększa trwałość projektu – strona lepiej znosi zmiany rozdzielczości, skalowanie tekstu oraz przyszłe urządzenia o niestandardowych parametrach.

Zasady dostępności standardu WCAG 2.1



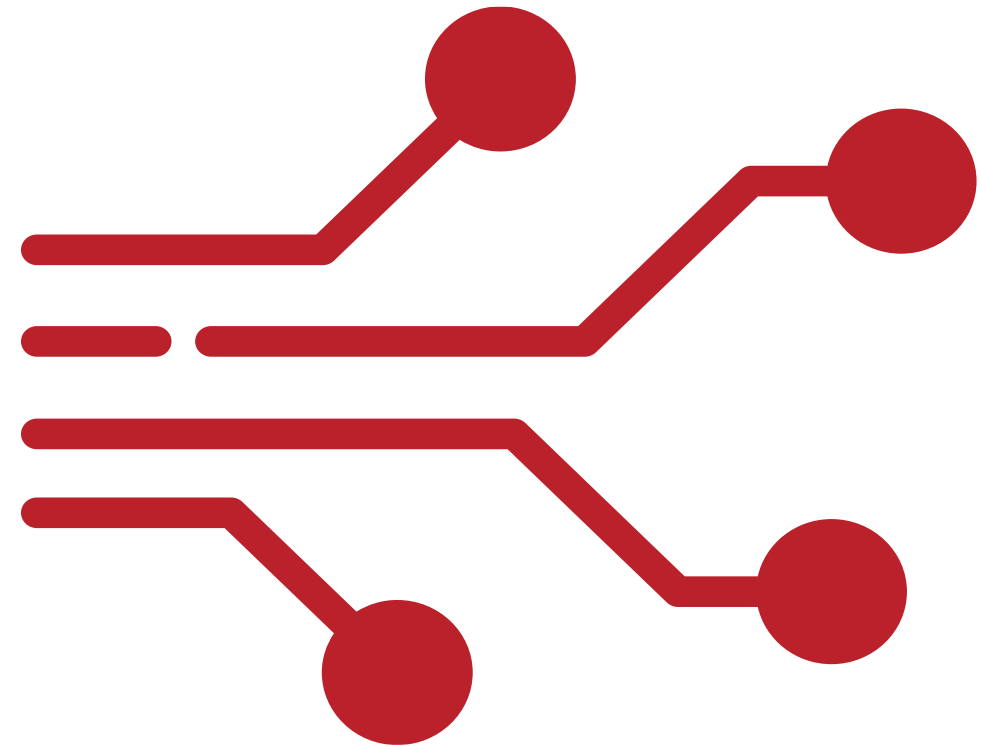
Rys. 6. Zasady dostępności WCAG 2.1 [źródło: <https://ks.pl/blog/wcag-21-wytyczne-dostepnosci-cyfrowej-stron-internetowych>]

Dostępność a SEO

Dostępność stron internetowych jest ściśle powiązana z optymalizacją pod kątem wyszukiwarek internetowych. Wyszukiwarki, podobnie jak technologie asystujące, analizują strukturę dokumentu, nagłówki, opisy alternatywne obrazów oraz logiczny układ treści.

Strona dobrze zaprojektowana pod względem dostępności jest zazwyczaj lepiej indeksowana przez wyszukiwarki, co przekłada się na wyższą pozycję w wynikach wyszukiwania. Dzięki temu dostępność staje się również elementem strategii SEO.

Optymalizacja stron pod kątem SEO

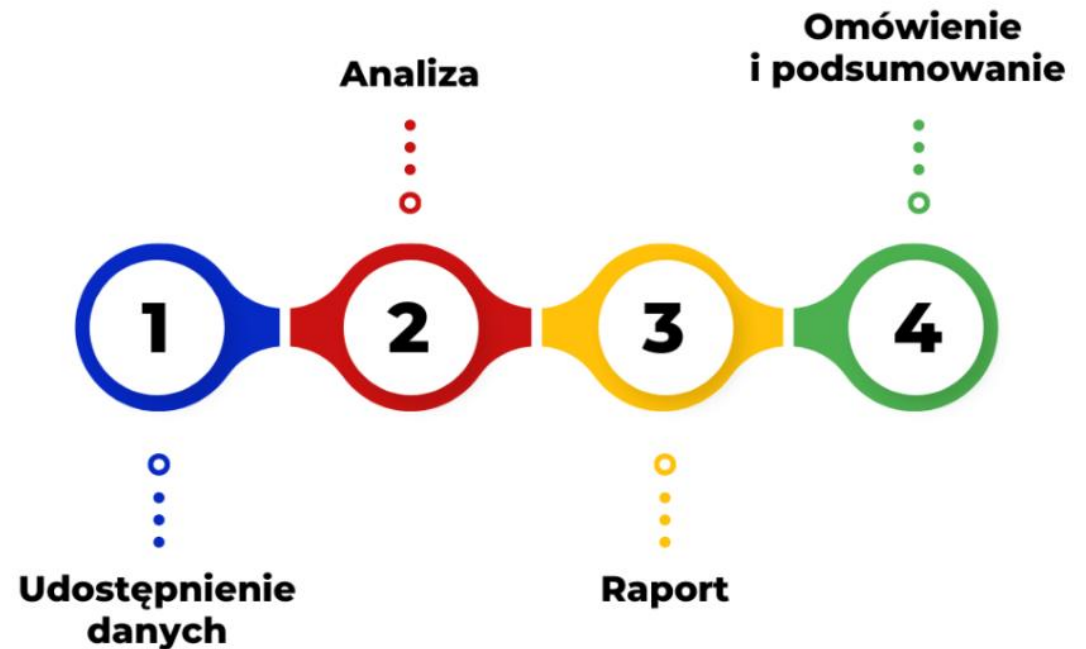


14

Podstawy SEO

SEO (Search Engine Optimization) to zbiór działań mających na celu poprawę widoczności strony internetowej w wynikach wyszukiwania. Obejmuje ono zarówno aspekty techniczne, jak i treściowe. Kluczowe znaczenie ma poprawna struktura HTML, logiczne nagłówki oraz unikalne i wartościowe treści.

Dla programisty istotne jest, aby strona była czytelna nie tylko dla użytkownika, ale także dla algorytmów wyszukiwarek, które analizują jej zawartość i strukturę.



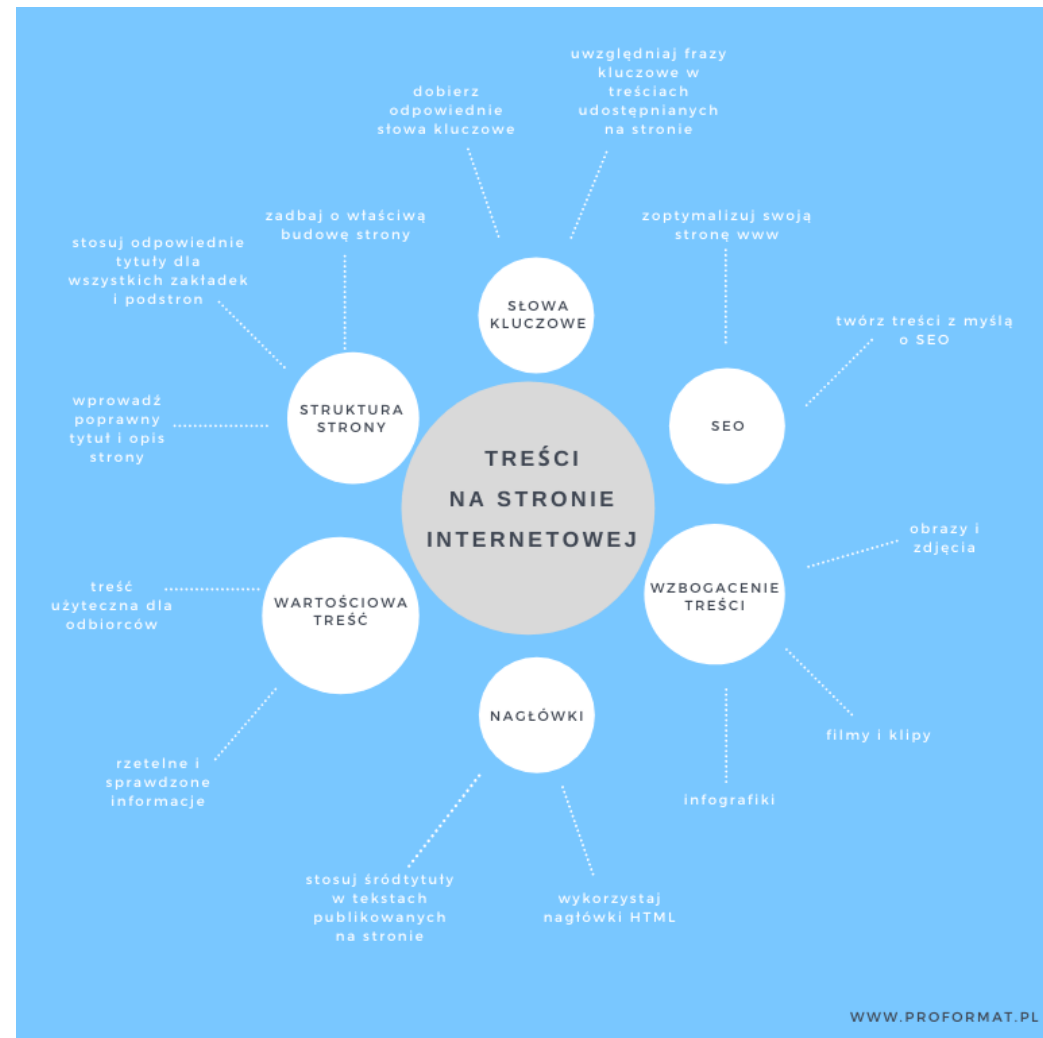
Rys. 7. Schemat audytu SEO [źródło: <https://monikakolodziejczyk.pl/audyt-marketingu/audyt-seo/>]

15

Znaczenie treści i struktury strony

Treść strony internetowej powinna być jasno sformułowana, logicznie uporządkowana i zgodna z oczekiwaniami użytkownika. Odpowiednie użycie nagłówków, akapitów i list ułatwia zarówno czytanie treści, jak i jej analizę przez wyszukiwarki.

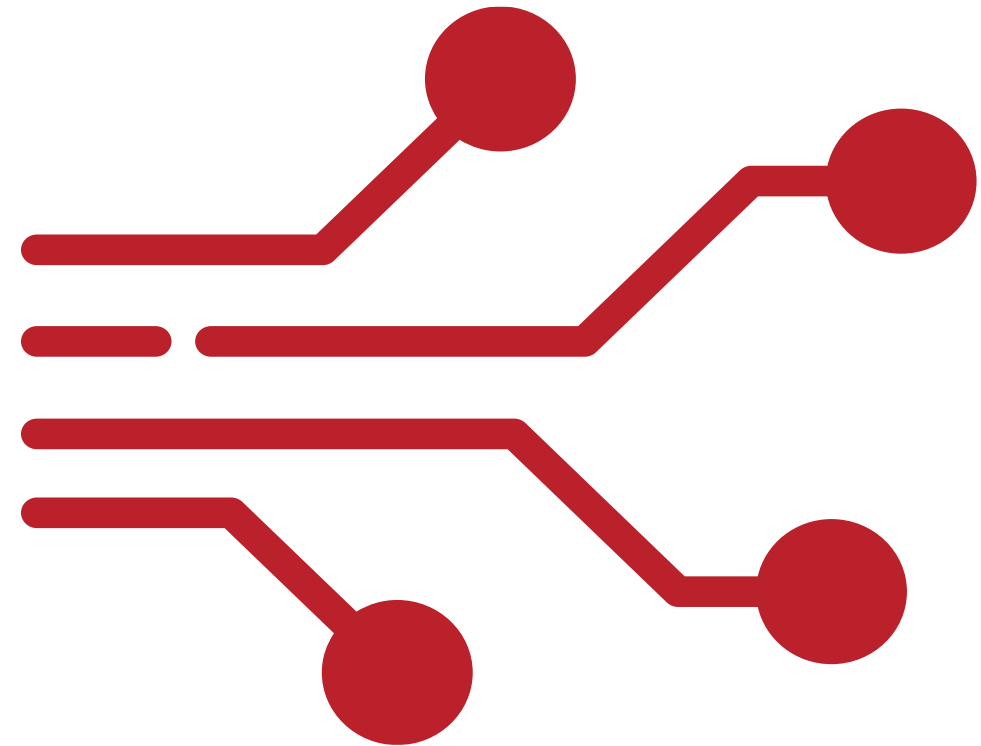
Struktura strony wpływa również na UX – użytkownik szybciej odnajduje potrzebne informacje, co zwiększa jego satysfakcję i czas spędzony na stronie.



Rys. 8. Treści na stronie internetowej
[źródło:
<https://www.proformat.pl/blog/tworzenie-tresci-na-strone-www/>]

17

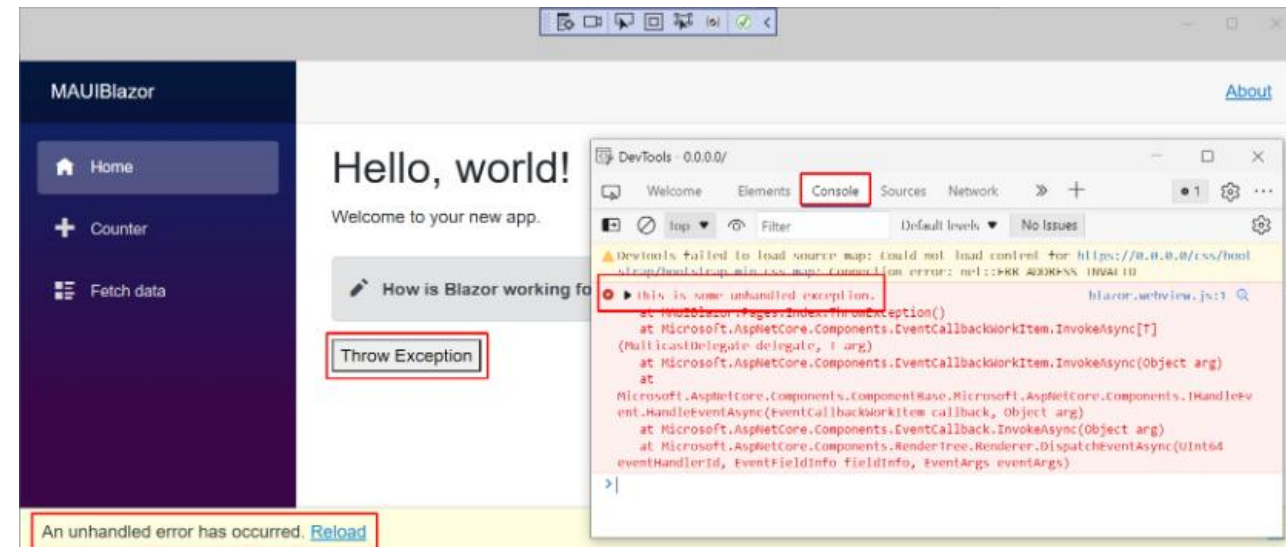
Narzędzia programistyczne w przeglądarkach



Narzędzia programistyczne przeglądarek

Nowoczesne przeglądarki internetowe oferują rozbudowane narzędzia programistyczne, które wspierają proces projektowania i debugowania stron. Pozwalają one analizować strukturę DOM, style CSS, działanie JavaScriptu oraz komunikację z serwerem.

Dzięki tym narzędziom programista może szybko identyfikować błędy, testować różne rozwiązania i optymalizować działanie aplikacji.



Rys. 9. Narzędzia deweloperskie MAUIBlazor
[źródło: <https://learn.microsoft.com/pl-pl/aspnet/core/blazor/hybrid/developer-tools?view=aspnetcore-10.0&pivots=windows>]

19

Testowanie responsywności i dostępności

Narzędzia deweloperskie umożliwiają symulację różnych rozmiarów ekranów oraz testowanie zachowania strony na urządzeniach mobilnych. Pozwalają także sprawdzać kontrast kolorów, strukturę nagłówków oraz poprawność semantyki HTML.

Regularne testowanie strony pod kątem responsywności i dostępności pozwala wykryć problemy na wczesnym etapie i znacząco poprawia jakość końcowego produktu.

7 PODSTAWOWYCH TESTÓW DOSTĘPNOŚCI



Czy wiesz, że 1,5 miliarda ludzi na świecie żyje z niepełnosprawnościami? Dostępność stron www to nie tylko wymóg prawny, ale przede wszystkim szacunek i troska o użytkowników.

1

LINK "PRZESKOCZ DO TREŚCI"

Czy istnieje nawigacja, która umożliwia korzystającym z klawiatury?

Czy można pominąć menu i przejść od razu do sedna strony?



2

TEST KŁAWIATURY

Czy strona działa bez myszy?

Czy da się dotrzeć do wszystkich elementów strony za pomocą klawiszy Tab, Enter, spacji itp.?



3

TEKST ALTERNATYWNY

Czy obrazek ma opis?

Czy osoba niewidoma, korzystająca z czytnika ekranu, zrozumie, co przedstawia zdjęcie lub grafikę?



KONTRAST

4

Czy tekst jest czytelny?

Czy kontrast między tekstem a tłem jest wystarczający dla osób słabowidzących lub z daltonizmem? (użyj narzędzi do testowania kontrastu)



5

NAGŁÓWKI (H1-H6)

Czy struktura strony jest logiczna?

Czy nagłówki tworzą czytelną hierarchię treści, ułatwiając nawigację?



6

CHARAKTERYSTYKA SENSORYCZNA

Czy informacje nie otrzymujemy tylko na jednym zmysle?

Czy istotne informacje nie są przekazywane wyłącznie za pomocą koloru, kształtu, dźwięku lub położenia elementów?

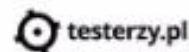


7

POWIEKSZENIE 200%

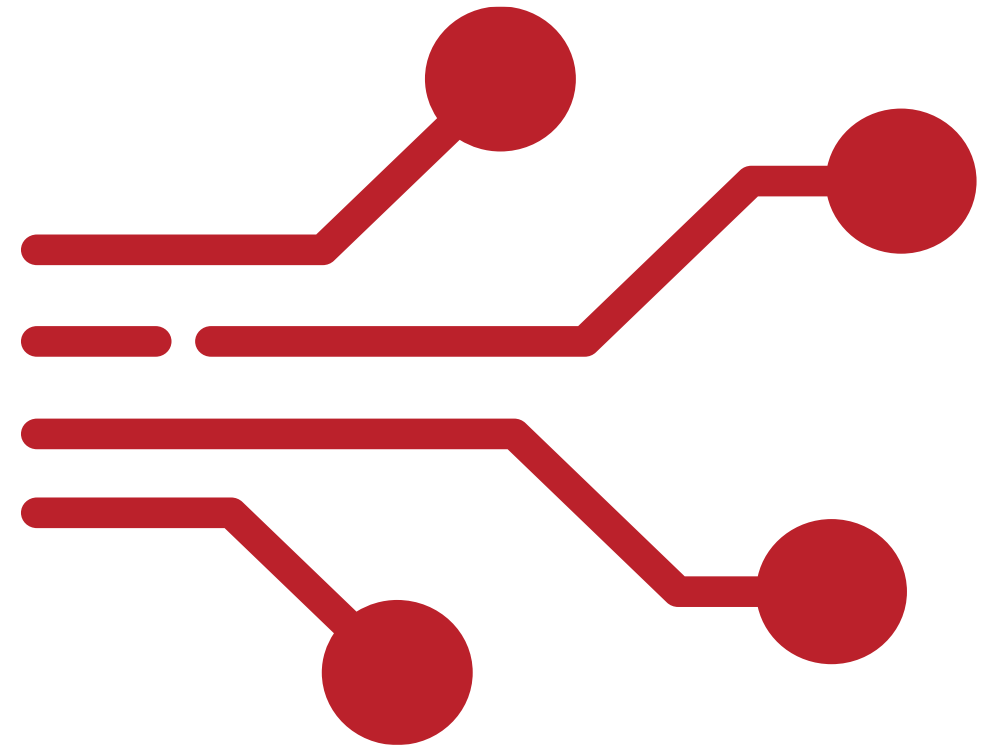
Czy strona jest czytelna przy dużym powiększeniu?

Czy układ strony nie rozjeżdża się, a tekst i elementy nie są ściśnięte?



Rys. 9. 7 podstawowych testów dostępności [źródło: <https://testerzy.pl/baza-wiedzy/artykuly/7-podstawowych-testow-dostepnosci>]

Podsumowanie



22

Projektowanie nowoczesnych interfejsów użytkownika wymaga dziś znacznie więcej niż estetycznego wyglądu strony. Kluczowe znaczenie ma połączenie zasad UI i UX, które pozwalają tworzyć interfejsy czytelne, intuicyjne i dostosowane do realnych potrzeb użytkowników. Równie istotna jest responsywność i elastyczność stron internetowych, dzięki którym treści i funkcjonalności mogą być poprawnie wyświetlane na różnych urządzeniach, niezależnie od rozmiaru ekranu czy sposobu interakcji.

Istotnym elementem współczesnego projektowania jest także dostępność cyfrowa, która umożliwia korzystanie z witryn osobom o różnych ograniczeniach i jednocześnie podnosi ogólną jakość oraz użyteczność strony. Zasady dostępności są ściśle powiązane z optymalizacją pod kątem wyszukiwarek internetowych, ponieważ dobrze zaprojektowana struktura dokumentu, semantyczny HTML i czytelna treść sprzyjają zarówno użytkownikom, jak i algorytmom wyszukiwarek. Dzięki temu dostępność i SEO wzajemnie się uzupełniają, wspierając widoczność i skuteczność witryn w sieci.

Literatura:

- [1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.
- [2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.
- [3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.
- [4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.
- [5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Dziękuję za uwagę

Wykład nr 9 Temat: Podstawy programowania po stronie serwera. Przetwarzanie żądań HTTP. Obsługa formularzy i walidacja danych

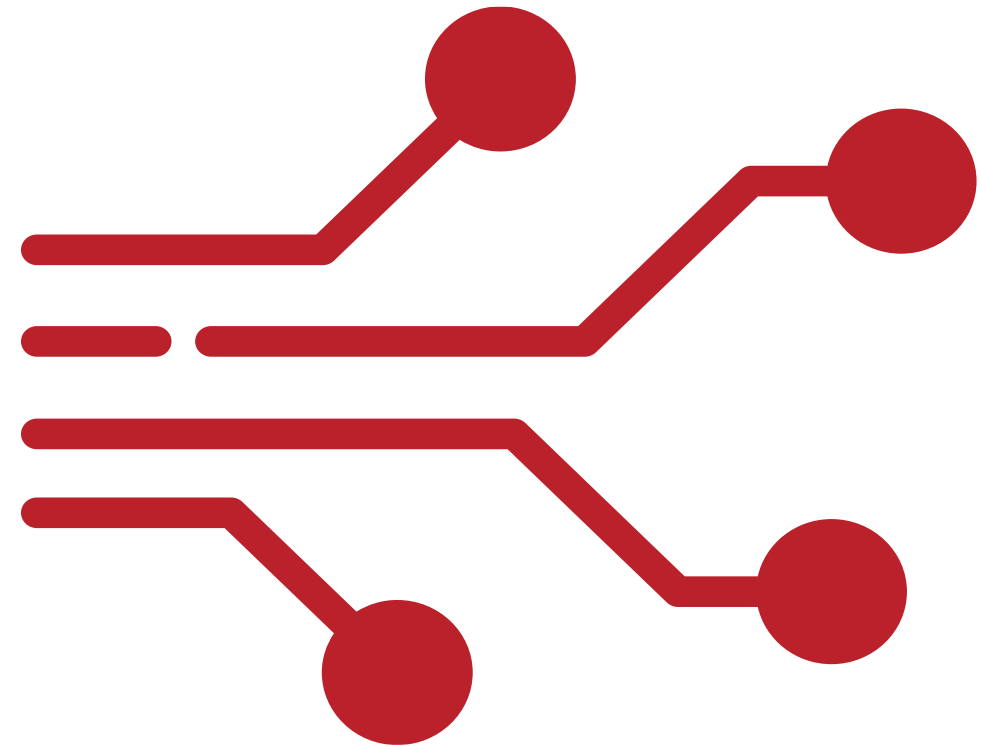
dr hab. inż. Michał Wydra, prof. Uczelni
Katedra Informatyki Stosowanej

Plan prezentacji

- Programowanie po stronie serwera
- Protokół HTTP – podstawy i właściwości
- Struktura komunikacji HTTP
- Metody HTTP, kody statusu i bezpieczeństwo
- Obsługa formularzy i walidacja danych



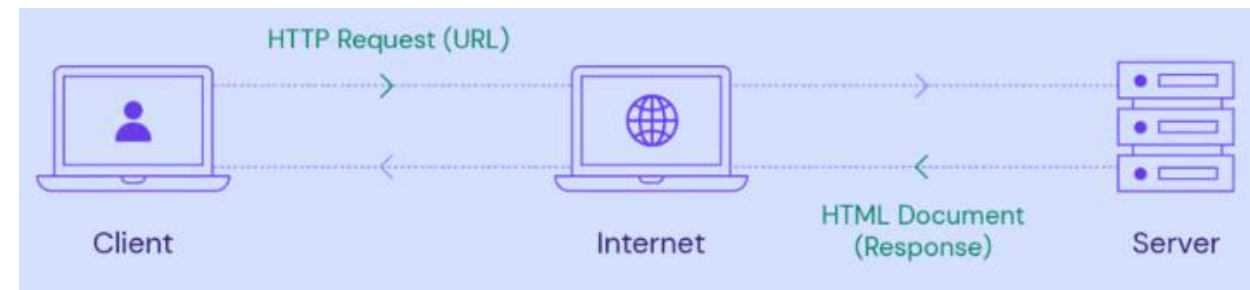
Programowanie po stronie serwera



3

Rola serwera w aplikacjach internetowych

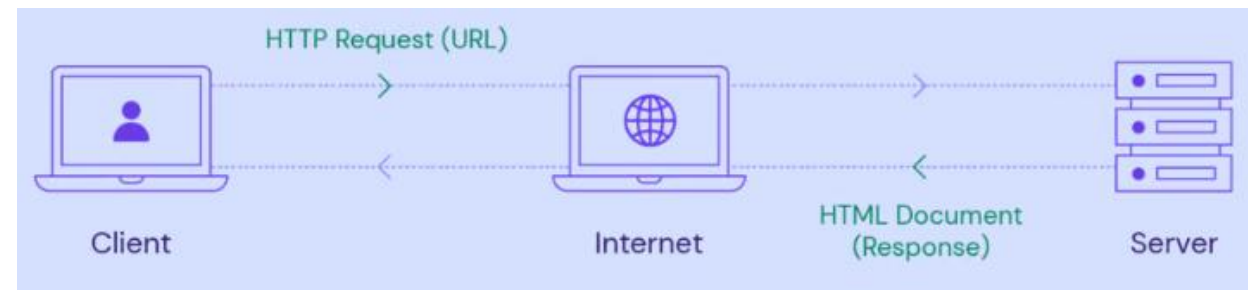
Serwer pełni kluczową rolę w aplikacjach internetowych, ponieważ odpowiada za realizację logiki biznesowej oraz przetwarzanie danych. To na serwerze wykonywane są operacje takie jak zapisywanie informacji w bazie danych, uwierzytelnianie użytkowników czy generowanie odpowiedzi na żądania klienta. Przeglądarka użytkownika pełni rolę klienta, który inicjuje komunikację.



Rys. 1. Schemat działania serwera WWW [źródło: <https://www.hostinger.com/pl/tutoriale/co-to-jest-serwer-www>]

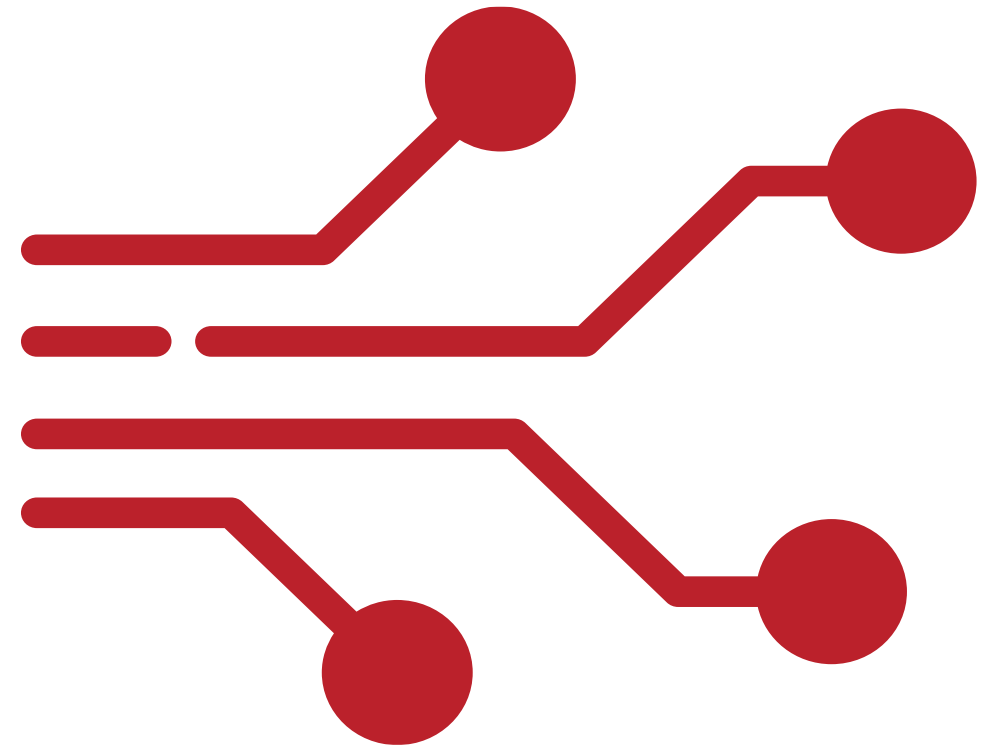
Odpowiedzialności serwera w aplikacji webowej

Serwer odpowiada za wiele kluczowych zadań, takich jak przetwarzanie danych wejściowych, komunikacja z bazą danych, uwierzytelnianie i autoryzacja użytkowników oraz generowanie odpowiedzi dla klienta. To na serwerze zapada decyzja, czy dane żądanie jest poprawne, czy użytkownik ma odpowiednie uprawnienia oraz jakie dane mogą zostać zwrócone. Dzięki temu możliwe jest centralne zarządzanie logiką aplikacji i jej bezpieczeństwem.



Rys. 1. Schemat działania serwera WWW [źródło: <https://www.hostinger.com/pl/tutoriale/co-to-jest-serwer-www>]

Protokół HTTP – podstawy i właściwości



6

HTTP jako fundament komunikacji w sieci Web

HTTP jest podstawowym protokołem wykorzystywanym do komunikacji pomiędzy klientem a serwerem w Internecie. Określa on sposób formułowania żądań oraz odpowiedzi, a także zasady ich interpretacji. Protokół HTTP został zaprojektowany jako prosty i uniwersalny mechanizm wymiany danych, co przyczyniło się do jego ogromnej popularności i powszechnego zastosowania.



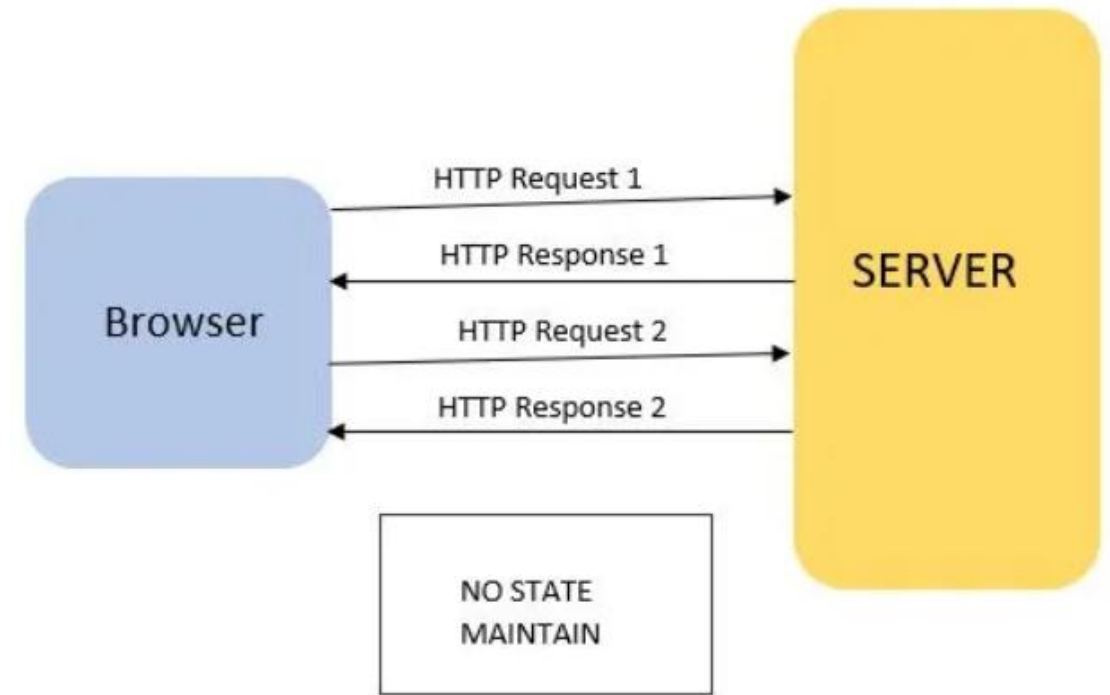
Rys. 2. HTTP

[źródło: <https://creassolutions.com/baza-wiedzy/http>]

7

Bezstanowość protokołu HTTP

Jedną z najważniejszych cech HTTP jest jego bezstanowość. Oznacza to, że każde żądanie jest traktowane niezależnie, a serwer nie pamięta wcześniejszych interakcji z klientem. Takie podejście upraszcza architekturę protokołu, ale jednocześnie wymaga stosowania dodatkowych mechanizmów, takich jak sesje czy ciasteczka, aby możliwe było utrzymanie stanu użytkownika w aplikacji.



Rys. 3. Bezstanowość protokołu HTTP

[źródło: <https://medium.com/%40aminsohailmanna324/cookies-and-session-70aa2646134d>]

8

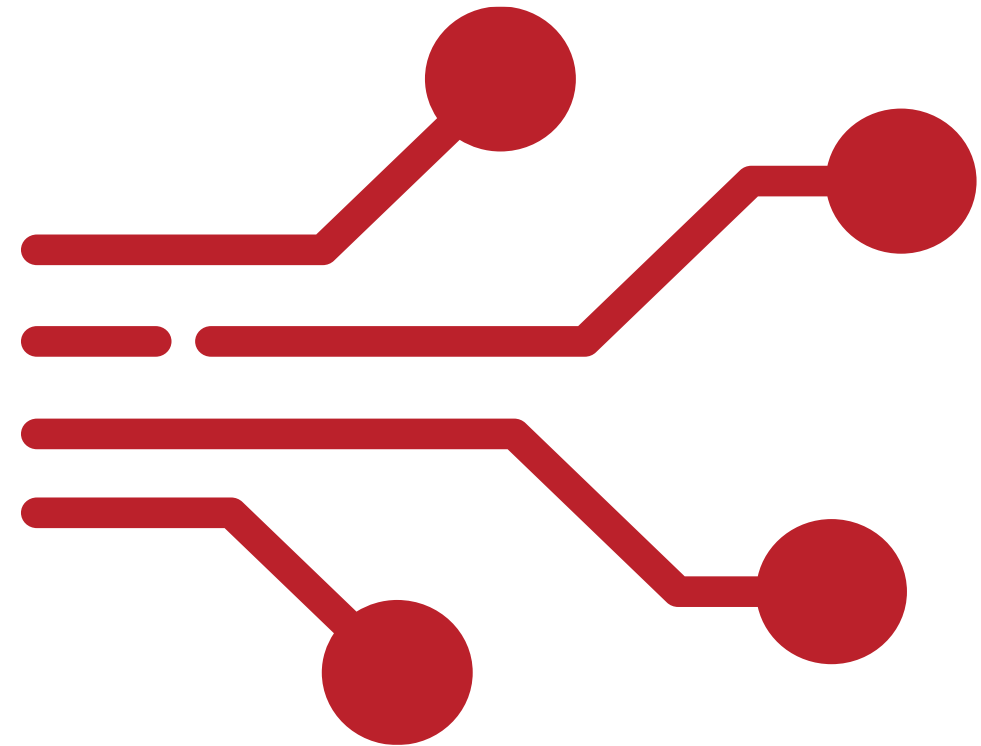
Przesyłanie danych w HTTP

W protokole HTTP dane mogą być przesyłane na kilka różnych sposobów, w zależności od rodzaju operacji i charakteru informacji. Najprostszą formą są parametry umieszczane w adresie URL, które są czytelne i łatwe do wykorzystania, lecz widoczne dla użytkownika i ograniczone pod względem bezpieczeństwa. Innym sposobem jest przekazywanie danych w nagłówkach HTTP, które służą głównie do przesyłania informacji technicznych, takich jak dane autoryzacyjne czy typ przesyłanej treści.

Najczęściej stosowaną metodą przesyłania większych lub wrażliwych danych jest umieszczanie ich w ciele żądania HTTP, na przykład podczas wysyłania formularzy lub danych w formacie JSON. Wybór sposobu przesyłania danych wpływa na bezpieczeństwo, czytelność komunikacji oraz przeznaczenie danego rozwiązania. Dlatego programista backendu musi świadomie decydować, w jaki sposób dane są odbierane, walidowane i interpretowane przez serwer, aby zapewnić poprawne i bezpieczne działanie aplikacji.

9

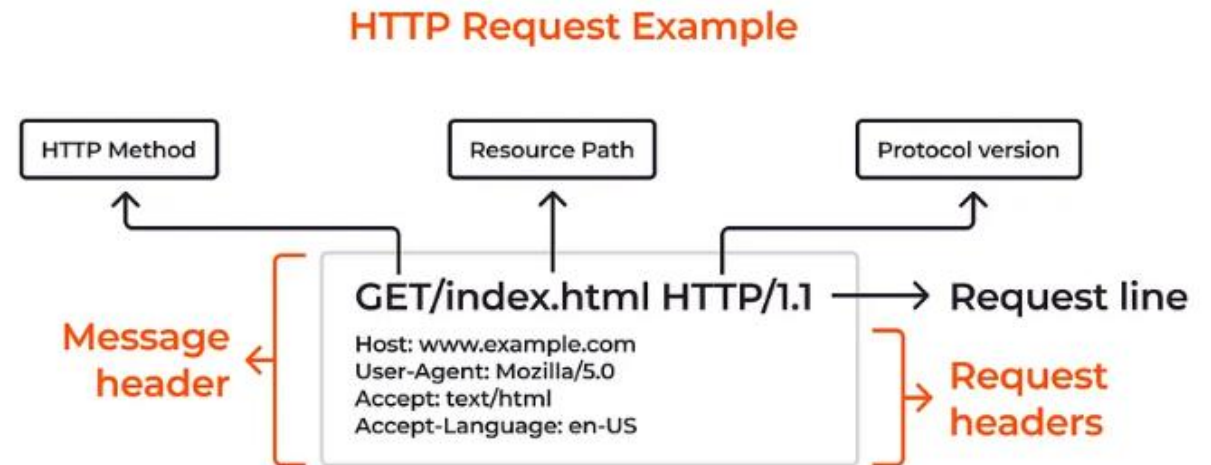
Struktura komunikacji HTTP



10

Budowa żądania HTTP

Żądanie HTTP składa się z kilku części, które razem precyzyjnie opisują intencję klienta. Linia początkowa zawiera metodę HTTP oraz adres zasobu, nagłówki przekazują dodatkowe informacje techniczne, a ciało żądania może zawierać dane użytkownika. Zrozumienie tej struktury jest kluczowe dla poprawnego przetwarzania żądań po stronie serwera.



Rys. 4. Przykładowy schemat żądania HTTP
[źródło: <https://medium.com/@cirilptomass/understanding-http-requests-responses-status-codes-and-methods-176bf7e0628a>]

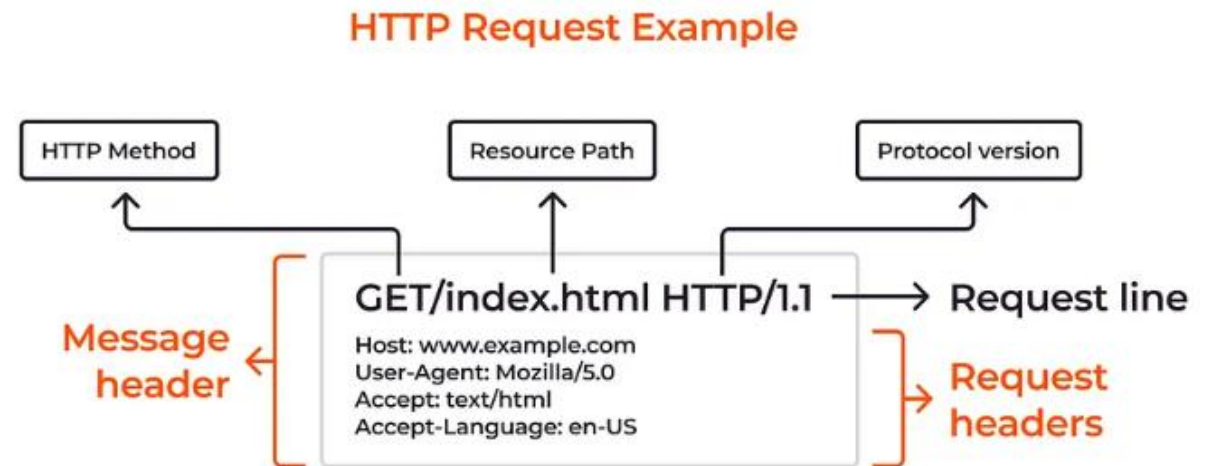
Metody zapytań HTTP

Metoda	Ciało (body) w żądaniu	Ciało (body) w odpowiedzi	Bezpieczna	Idempotentna	Wspiera cache
GET	Opcjonalnie	Tak	Tak	Tak	Tak
HEAD	Opcjonalnie	Nie	Tak	Tak	Tak
POST	Tak	Tak	Nie	Nie	Tak
PUT	Tak	Tak	Nie	Tak	Nie
DELETE	Opcjonalnie	Tak	Nie	Tak	Nie
CONNECT	Opcjonalnie	Tak	Nie	Nie	Nie
OPTIONS	Opcjonalnie	Tak	Tak	Tak	Nie
TRACE	Nie	Tak	Tak	Tak	Nie
PATCH	Tak	Tak	Nie	Nie	Nie

Rys. 5. Metody zapytań HTTP
[źródło: <https://lukasz-zielinski.pl/blog/posts/2021-08-07-rest-api/>]

Budowa odpowiedzi HTTP

Odpowiedź HTTP zwracana przez serwer zawiera informację o wyniku przetwarzania żądania. Kluczowym elementem jest kod statusu, który informuje klienta o powodzeniu lub błędzie. Odpowiedź może zawierać dane w różnych formatach, takich jak HTML, JSON czy pliki binarne, w zależności od charakteru aplikacji.



Rys. 6. Przykładowy schemat żądania HTTP
[źródło: <https://medium.com/@cirilptomass/understanding-http-requests-responses-status-codes-and-methods-176bf7e0628a>]

Kody odpowiedzi HTTP

1XX

Informacja
np. 100 Continue

2XX

Sukces
np. 200 OK, 201 Created,
202 Accepted

3XX

Przekierowanie
np. 301 Moved Permanently

4XX

Błąd klienta
np. 400 Bad Request, 404 Not Found,
418 I'm a teapot

5XX

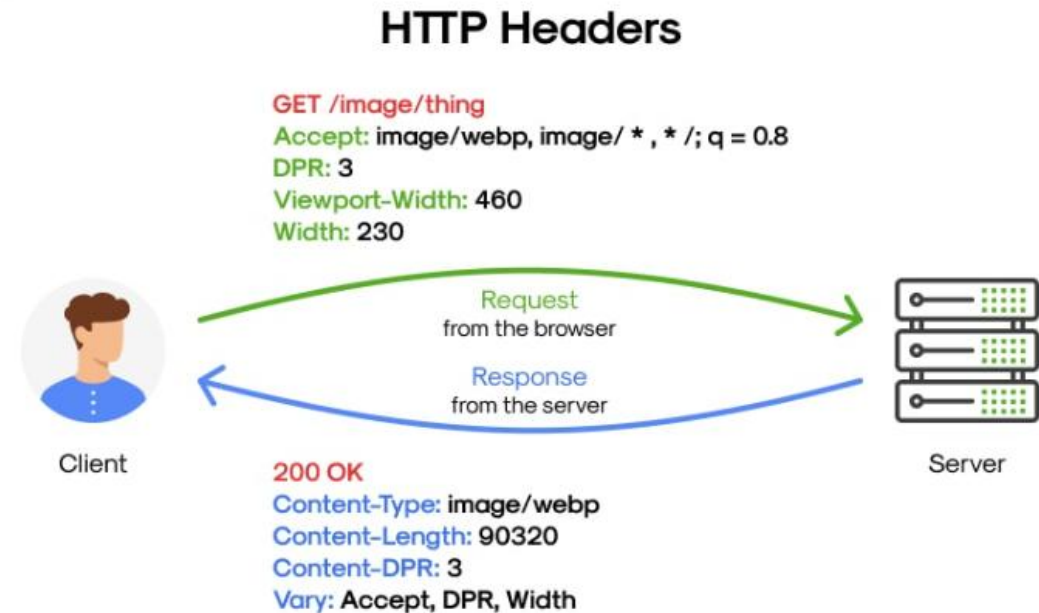
Błąd serwera
np. 500 Internal Server Error,
508 Loop Detected

Rys. 7. Kody odpowiedzi HTTP
[źródło: <https://lukasz-zielinski.pl/blog/posts/2021-08-07-rest-api/>]

14

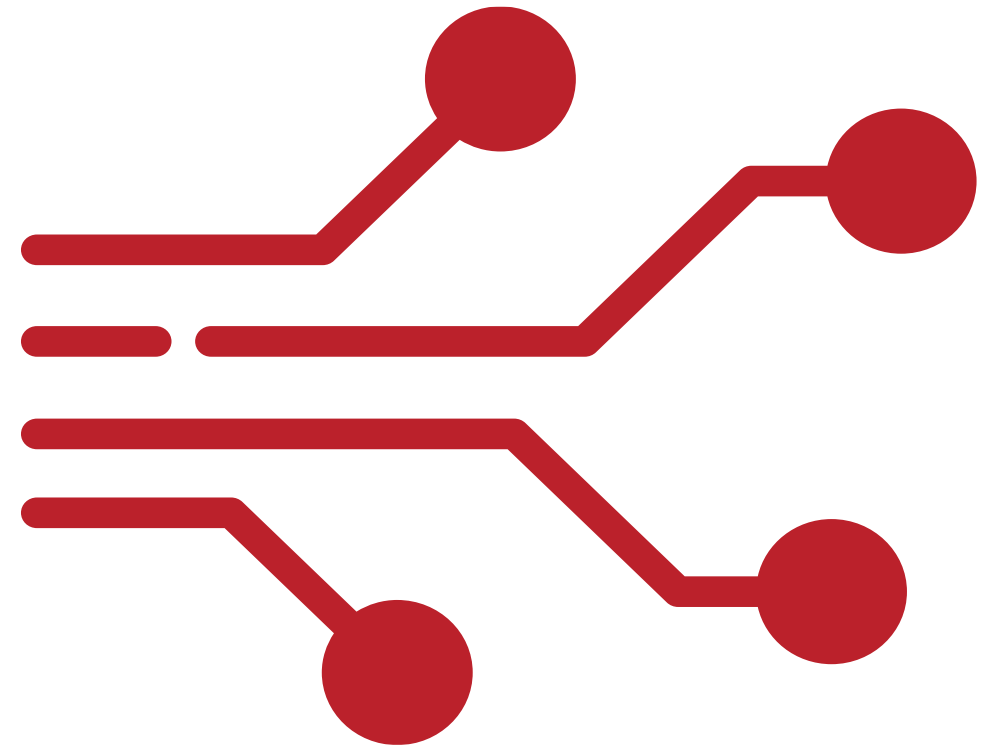
Nagłówki HTTP i ich znaczenie

Nagłówki HTTP pełnią bardzo ważną rolę w komunikacji klient-serwer. Pozwalają przekazywać informacje o typie danych, kodowaniu, autoryzacji czy mechanizmach cache'owania. Poprawne wykorzystanie nagłówków wpływa na wydajność aplikacji, bezpieczeństwo oraz poprawną interpretację danych przez klienta.



Rys. 8. Działanie nagłówków HTTP
[źródło: <https://www.wallarm.com/what/http-headers>]

Metody HTTP, kody statusu i bezpieczeństwo



16

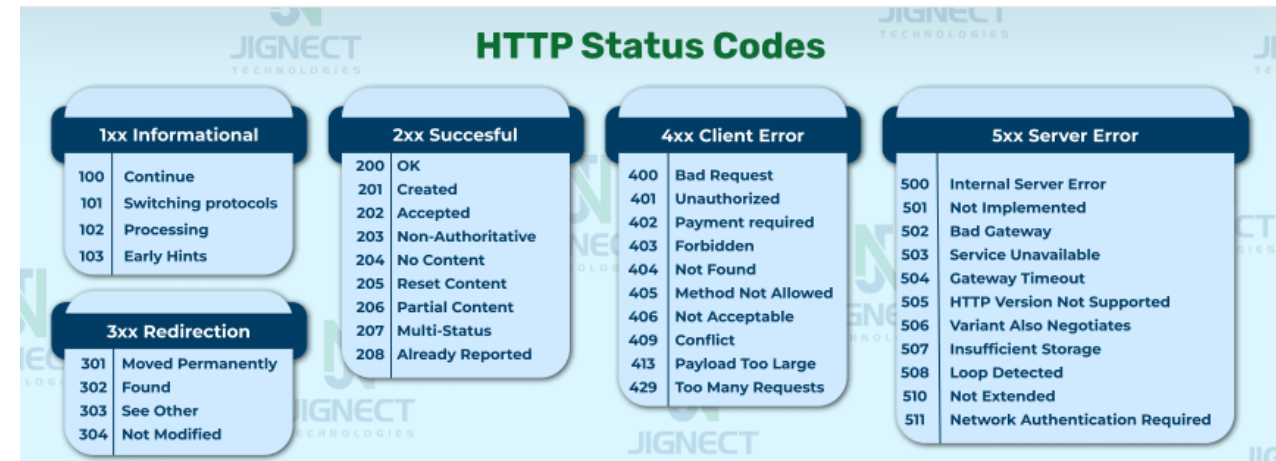
Metody HTTP i ich semantyka

Metody HTTP określają intencję żądania wysłanego przez klienta do serwera, czyli informują, jakiego rodzaju operacja ma zostać wykonana na danym zasobie. Dzięki nim serwer wie, czy klient chce jedynie pobrać dane, zapisać nowe informacje, zmodyfikować istniejące zasoby czy je usunąć. Mechanizm ten stanowi podstawę czytelnej i uporządkowanej komunikacji w aplikacjach webowych.

Oprócz najczęściej stosowanych metod, takich jak GET do pobierania danych oraz POST do ich przesyłania, istnieją także inne metody o jasno określonej semantyce. Metoda PUT służy do aktualizacji lub zastąpienia zasobu, DELETE do jego usunięcia, natomiast PATCH do częściowej modyfikacji danych. Poprawne i konsekwentne stosowanie metod HTTP ma duże znaczenie dla czytelności API, zgodności ze standardami sieci Web oraz łatwości integracji systemów.

Kody statusu HTTP jako mechanizm informacji

Kody statusu HTTP informują klienta o wyniku przetwarzania żądania. Dzięki nim aplikacja frontendowa może odpowiednio zareagować na sukces, błąd użytkownika lub problem po stronie serwera. Świadome używanie kodów statusu zwiększa przejrzystość komunikacji oraz ułatwia debugowanie aplikacji.



1xx Informational		2xx Successful		4xx Client Error		5xx Server Error	
100	Continue	200	OK	400	Bad Request	500	Internal Server Error
101	Switching protocols	201	Created	401	Unauthorized	501	Not Implemented
102	Processing	202	Accepted	402	Payment required	502	Bad Gateway
103	Early Hints	203	Non-Authoritative	403	Forbidden	503	Service Unavailable
		204	No Content	404	Not Found	504	Gateway Timeout
		205	Reset Content	405	Method Not Allowed	505	HTTP Version Not Supported
		206	Partial Content	406	Not Acceptable	506	Variant Also Negotiates
		207	Multi-Status	409	Conflict	507	Insufficient Storage
		208	Already Reported	413	Payload Too Large	508	Loop Detected
				429	Too Many Requests	510	Not Extended
						511	Network Authentication Required

Rys. 9. Kody statusu HTTP

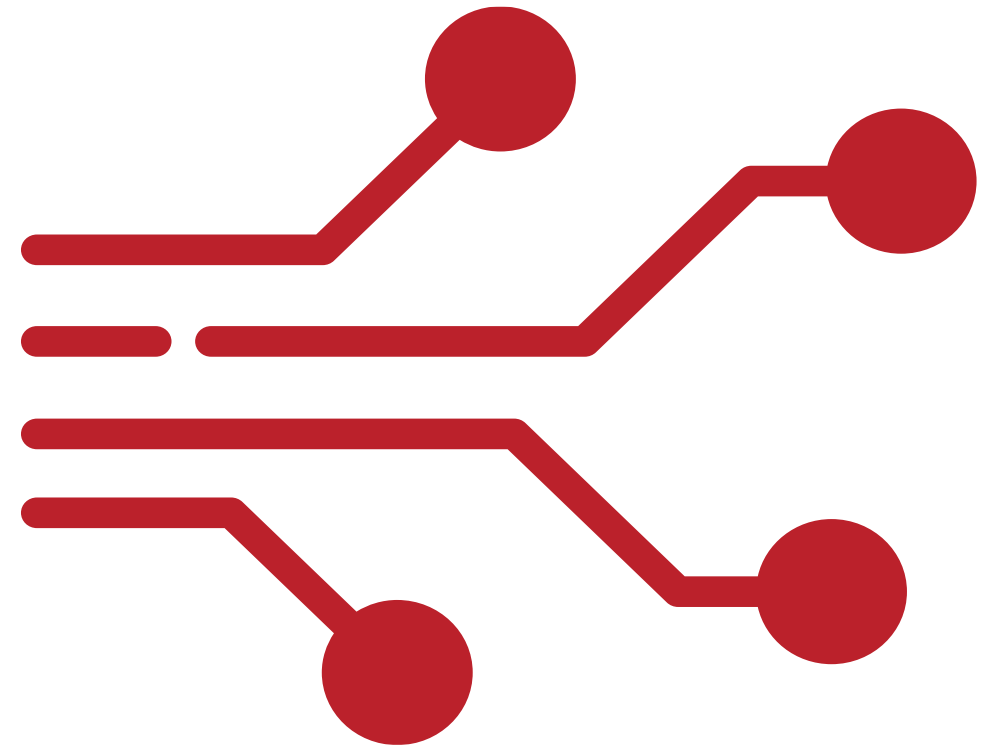
[źródło: <https://jignect.tech/http-status-codes-understanding-their-significance-and-importance/>]

HTTP a bezpieczeństwo aplikacji

Protokół HTTP sam w sobie nie zapewnia poufności ani integralności przesyłanych danych, co oznacza, że informacje mogą zostać przechwycone lub zmodyfikowane w trakcie transmisji. Z tego powodu we współczesnych aplikacjach standardem jest stosowanie HTTPS, czyli HTTP z szyfrowaniem TLS, które chroni dane przesyłane pomiędzy przeglądarką a serwerem przed podsłuchem i manipulacją.

Oprócz stosowania HTTPS istotną rolę w zapewnieniu bezpieczeństwa odgrywa backend aplikacji. To on odpowiada za weryfikację poprawności żądań, walidację danych wejściowych oraz ochronę przed atakami, takimi jak wstrzykiwanie danych czy próby nieautoryzowanego dostępu. Bezpieczna obsługa komunikacji HTTP jest więc jednym z kluczowych zadań programisty backendu i ma bezpośredni wpływ na stabilność oraz zaufanie do aplikacji webowej.

Obsługa formularzy i walidacja danych



Formularze jako źródło danych użytkownika

Formularze stanowią podstawowy sposób przekazywania danych od użytkownika do aplikacji serwerowej. Umożliwiają one wprowadzanie informacji takich jak dane logowania, dane kontaktowe czy treści przesyłane w ramach formularzy rejestracyjnych i zamówień. Po wysłaniu formularza dane trafiają do backendu, gdzie są dalej przetwarzane, zapisywane w bazie danych lub wykorzystywane do realizacji określonych operacji.

Każdy formularz jest jednak potencjalnym źródłem błędów lub zagrożeń bezpieczeństwa, ponieważ dane pochodzą bezpośrednio od użytkownika i nie można zakładać ich poprawności. Z tego powodu backend musi dokładnie weryfikować i walidować otrzymane informacje, sprawdzając ich format, zakres oraz zgodność z oczekiwaniami aplikacji. Staranna obsługa formularzy ma kluczowe znaczenie dla bezpieczeństwa, stabilności i poprawnego działania systemu webowego.

Przetwarzanie danych formularza na serwerze

Po otrzymaniu danych przesłanych z formularza serwer rozpoczyna proces ich przetwarzania. Na tym etapie analizowana jest struktura danych oraz ich znaczenie w kontekście działania aplikacji. W zależności od funkcjonalności systemu dane mogą zostać zapisane w bazie danych, wykorzystane do wykonania obliczeń lub przekazane do innych usług i systemów zewnętrznych.

Szczególnie istotnym elementem tego procesu jest weryfikacja poprawności otrzymanych informacji. Backend musi sprawdzić, czy wszystkie wymagane pola zostały wypełnione oraz czy dane są zgodne z oczekiwanym formatem i zakresem. Takie podejście pozwala zapobiegać błędom, zwiększa bezpieczeństwo aplikacji i zapewnia poprawne działanie dalszych etapów przetwarzania.

Walidacja danych – zasady i znaczenie

Walidacja danych polega na sprawdzaniu poprawności informacji przesyłanych przez użytkownika przed ich dalszym przetwarzaniem przez aplikację. Obejmuje ona weryfikację formatu danych, na przykład poprawności adresu e-mail, sprawdzanie zakresu wartości liczbowych oraz kontrolę obecności wymaganych pól. Celem walidacji jest upewnienie się, że otrzymane dane są zgodne z założeniami logiki aplikacji.

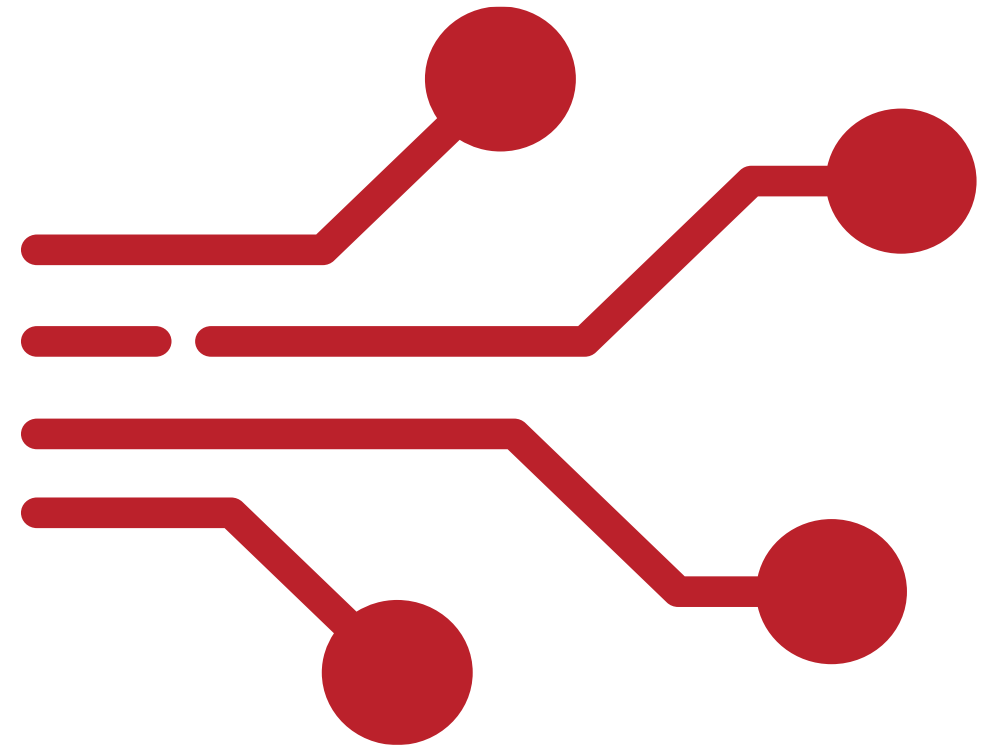
Szczególne znaczenie ma walidacja po stronie serwera, która jest absolutnie konieczna niezależnie od ewentualnych zabezpieczeń po stronie klienta. Dane pochodzące od użytkownika nie mogą być uznane za wiarygodne, ponieważ mogą zostać zmodyfikowane lub wysłane z pominięciem interfejsu aplikacji. Poprawnie zaimplementowana walidacja zwiększa bezpieczeństwo systemu, zapobiega błędom oraz chroni aplikację przed nieprawidłowym lub złośliwym wykorzystaniem.

Obsługa błędów i komunikacja z użytkownikiem

Obsługa błędów i właściwa komunikacja z użytkownikiem są istotnym elementem działania aplikacji serwerowej. W sytuacji, gdy serwer wykryje błąd w przesłanych danych lub nie może poprawnie zrealizować żądania, powinien zwrócić jasną i jednoznaczną informację o problemie. Czytelne komunikaty pomagają użytkownikowi zrozumieć, co poszło nie tak, oraz wskazują, jakie działania należy podjąć, aby poprawić dane lub ponowić operację.

Ważnym elementem tej komunikacji jest stosowanie odpowiednich kodów statusu HTTP, które informują o rodzaju problemu, na przykład błędzie po stronie klienta lub serwera. Dzięki temu zarówno użytkownik, jak i twórcy aplikacji mogą szybciej zdiagnozować przyczynę niepowodzenia. Jednocześnie serwer nie powinien ujawniać szczegółowych informacji technicznych, takich jak struktura bazy danych czy szczegóły konfiguracji systemu, ponieważ mogłoby to stanowić zagrożenie dla bezpieczeństwa aplikacji.

Podsumowanie



25

Programowanie po stronie serwera obejmuje szeroki zakres zadań związanych z obsługą żądań HTTP, przetwarzaniem danych oraz realizacją logiki biznesowej aplikacji. To właśnie backend odpowiada za podejmowanie decyzji, wykonywanie operacji na danych, komunikację z bazami danych oraz integrację z innymi systemami i usługami zewnętrznymi. Jego działanie jest niewidoczne dla użytkownika, jednak ma kluczowe znaczenie dla poprawnego funkcjonowania całej aplikacji. Podstawą tworzenia nowoczesnych backendów jest dobre zrozumienie protokołu HTTP, w tym jego bezstanowego charakteru, dostępnych metod, nagłówków oraz kodów statusu. Elementy te determinują sposób komunikacji pomiędzy klientem a serwerem i pozwalają na budowanie czytelnych, przewidywalnych oraz zgodnych ze standardami interfejsów API. Umiejętne wykorzystanie tych mechanizmów ułatwia rozwój aplikacji oraz jej integrację z innymi systemami. Szczególnie istotną rolę w programowaniu po stronie serwera odgrywa obsługa formularzy i danych pochodzących od użytkowników. Poprawna walidacja danych, kontrola ich formatu oraz zakresu, a także bezpieczne przetwarzanie informacji są niezbędne do zapewnienia stabilności, bezpieczeństwa i niezawodności aplikacji internetowych. Dzięki temu backend może skutecznie chronić system przed błędami i zagrożeniami, jednocześnie dostarczając użytkownikom spójne i bezpieczne działanie aplikacji.

26

Literatura:

- [1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.
- [2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.
- [3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.
- [4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.
- [5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Dziękuję za uwagę



Wykład nr 10

Temat: Tworzenie API dla aplikacji internetowych

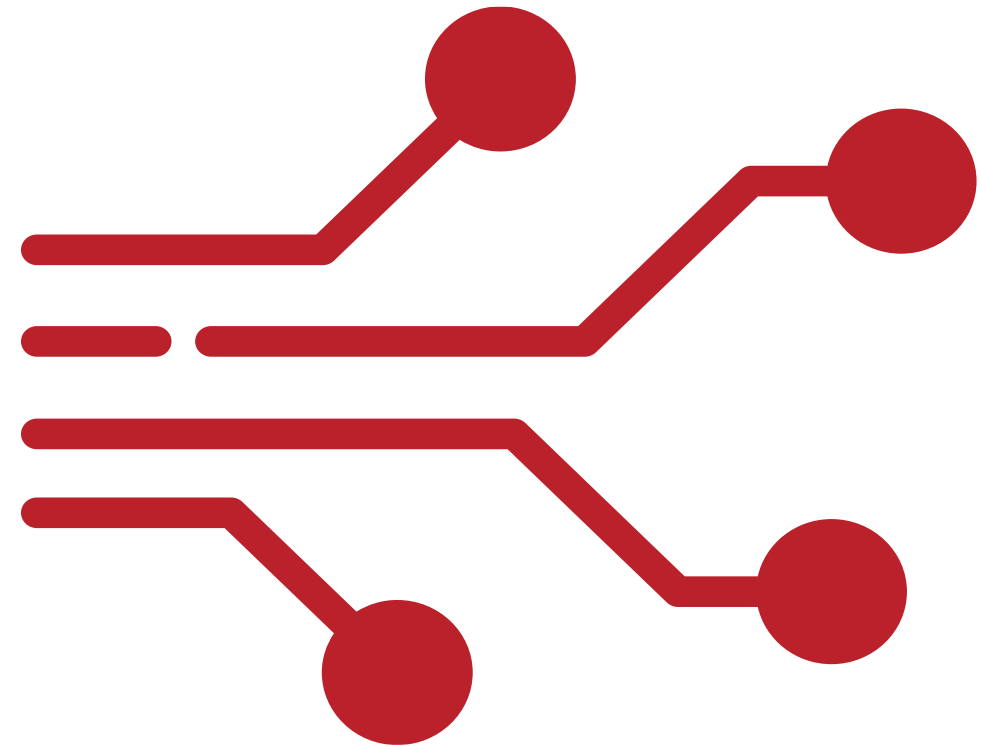
dr hab. inż. Michał Wydra, prof. Uczelni
Katedra Informatyki Stosowanej

Plan prezentacji

- Czym jest API i dlaczego jest potrzebne?
- API w architekturze aplikacji internetowych
- Projektowanie i komunikacja z API
- Bezpieczeństwo i niezawodność API
- Testowanie i rozwój API



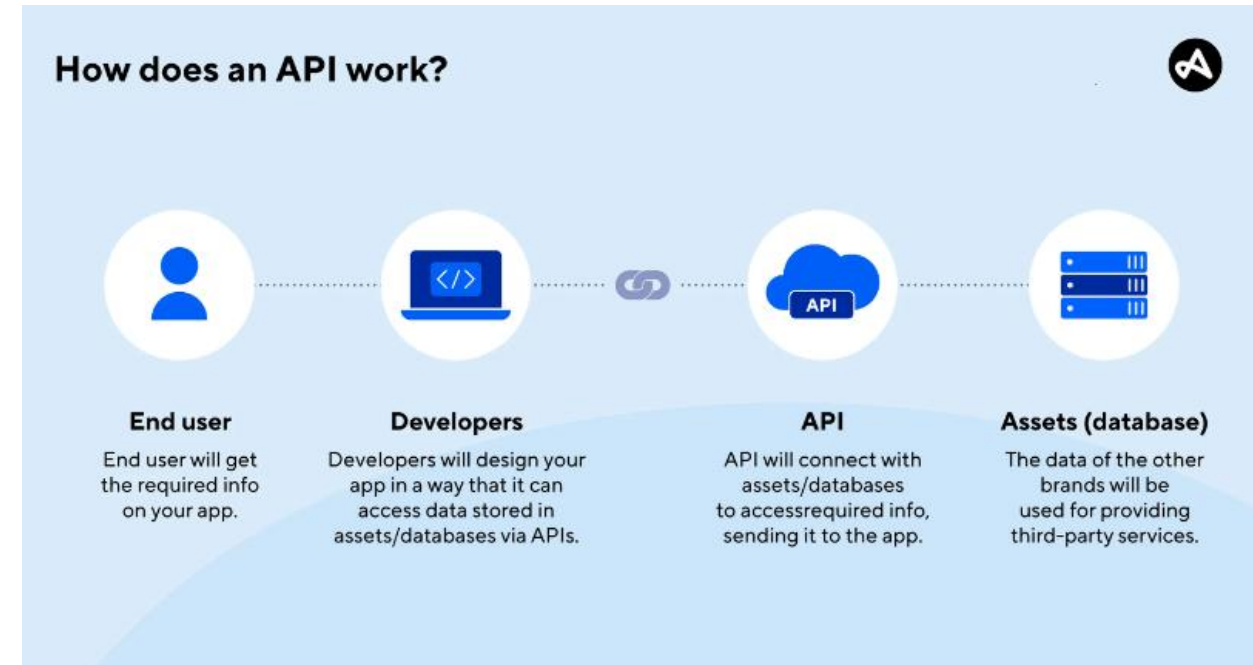
Czym jest API i dlaczego jest potrzebne?



3

Pojęcie API

API, czyli Application Programming Interface, to zestaw reguł i mechanizmów umożliwiających komunikację pomiędzy różnymi komponentami oprogramowania. W kontekście aplikacji internetowych API najczęściej przyjmuje postać interfejsu sieciowego, który umożliwia klientom wysyłanie żądań do serwera i odbieranie odpowiedzi w ustandaryzowany sposób. API definiuje, jakie operacje są dostępne oraz w jaki sposób można z nich korzystać.



Rys. 1. Schemat działania API

[źródło: <https://www.adjust.com/glossary/api/>]

API jako warstwa pośrednia

API pełni rolę warstwy pośredniej pomiędzy klientem a logiką aplikacji działającą na serwerze. Klient nie ma bezpośredniego dostępu do bazy danych ani wewnętrznych mechanizmów aplikacji – komunikuje się wyłącznie poprzez API. Takie podejście zwiększa bezpieczeństwo, ułatwia kontrolę dostępu do danych oraz pozwala na rozwój aplikacji bez konieczności modyfikowania interfejsu użytkownika.



Rys. 2. Komunikacja między aplikacjami przez API
[źródło: <https://devstockacademy.pl/blog/narzedzia-i-automatyzacja/co-to-jest-api-i-jak-dziala-kompletny-przewodnik-dla-programistow/?srsltid=AfmBOopMTmknzKJNqvnT6FHsjtyhnbdkhKPsiiFVcY0BmQwPki242Uht>]

Rodzaje API

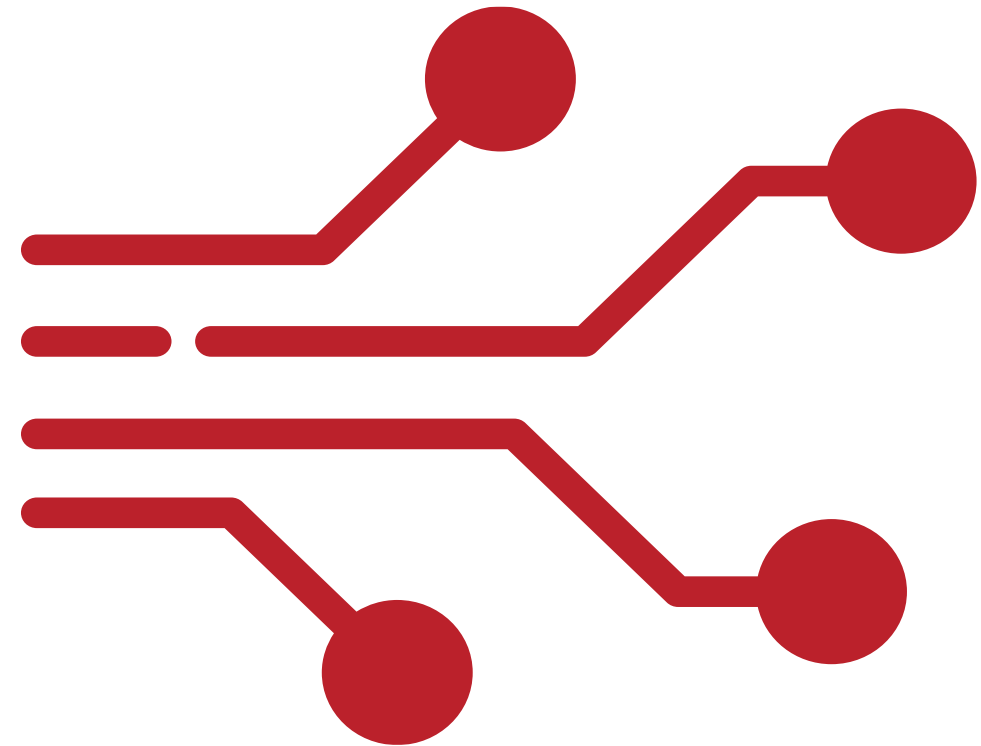
W praktyce tworzenia aplikacji internetowych spotyka się różne podejścia do projektowania i udostępniania API. Różnią się one sposobem komunikacji, formatem danych oraz stopniem formalizacji. Do najczęściej omawianych należą REST API, SOAP API oraz RPC API.

REST API (Representational State Transfer) to obecnie najpopularniejszy styl projektowania API w aplikacjach webowych. Opiera się on na wykorzystaniu protokołu HTTP oraz jego metod, takich jak GET, POST, PUT czy DELETE, do wykonywania operacji na zasobach. REST API jest lekkie, czytelne i elastyczne, a dane są najczęściej przesyłane w formacie JSON. Główną cechą REST jest bezstanowość – każde żądanie zawiera komplet informacji potrzebnych do jego obsługi, co ułatwia skalowanie aplikacji i integrację z różnymi klientami.

SOAP API (Simple Object Access Protocol) to bardziej sformalizowane podejście do komunikacji pomiędzy systemami. SOAP opiera się na protokole XML i wykorzystuje ściśle zdefiniowane struktury komunikatów oraz kontrakty opisujące dostępne operacje. Dzięki temu SOAP API jest często wykorzystywane w dużych systemach korporacyjnych, gdzie ważna jest jednoznaczność, bezpieczeństwo oraz zgodność ze standardami. Z drugiej strony, jego złożoność i większy narzut komunikacyjny sprawiają, że jest rzadziej stosowane w nowoczesnych aplikacjach webowych.

RPC API (Remote Procedure Call) to podejście, w którym API udostępnia zestaw procedur lub funkcji, które klient może wywoływać zdalnie, podobnie jak lokalne metody w programie. W tym modelu klient nie operuje bezpośrednio na zasobach, lecz wywołuje konkretne operacje. RPC API może wykorzystywać różne protokoły i formaty danych, a jego zaletą jest prostota oraz bezpośrednie odwzorowanie logiki aplikacji. Wadą jest mniejsza elastyczność oraz słabsze dopasowanie do architektury sieci Web w porównaniu do REST.

API w architekturze aplikacji internetowych



API w nowoczesnych aplikacjach webowych

W nowoczesnych aplikacjach webowych bardzo często stosuje się architekturę opartą na wyraźnym rozdzieleniu frontendu i backendu. Backend odpowiada za logikę biznesową, przetwarzanie danych oraz komunikację z bazami danych, natomiast swoje funkcjonalności udostępnia w postaci API. Frontend nie komunikuje się bezpośrednio z bazą danych, lecz korzysta z API, wysyłając zapytania HTTP i odbierając odpowiedzi w ustalonym formacie, najczęściej JSON.

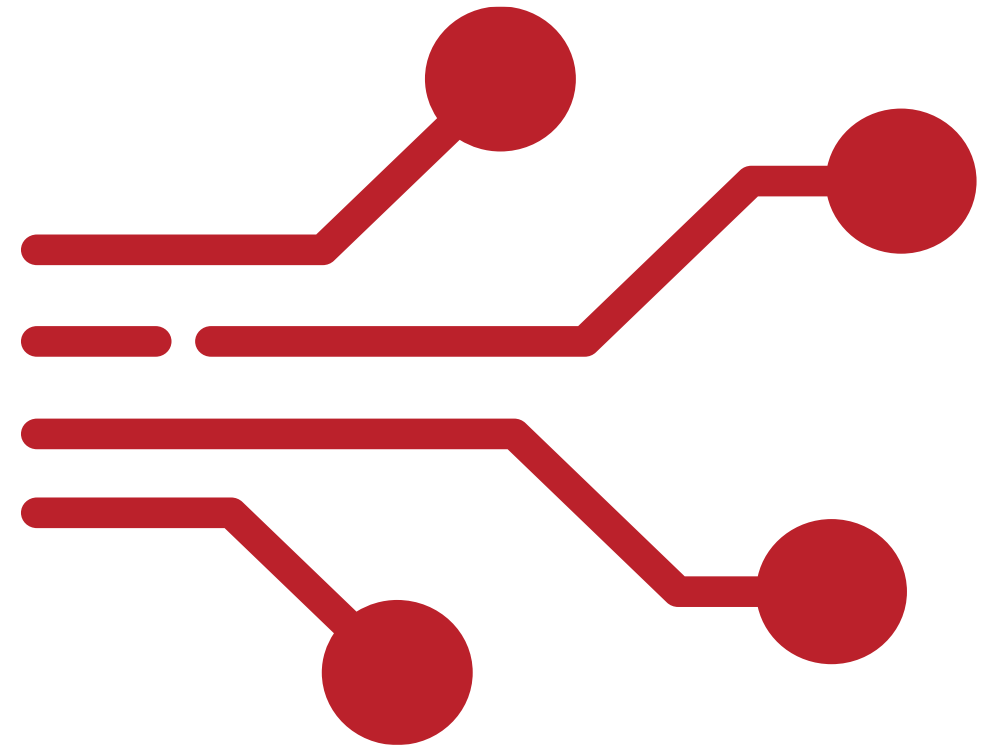
Takie podejście zapewnia dużą elastyczność systemu. Jedna warstwa API może obsługiwać różne typy klientów, takie jak aplikacje webowe, mobilne, desktopowe czy systemy zewnętrzne, bez konieczności powielania logiki po stronie serwera. Oddzielenie frontendu od backendu ułatwia także rozwój, testowanie i skalowanie aplikacji, a zespoły mogą pracować niezależnie nad poszczególnymi częściami systemu.

API a skalowalność systemu

Zastosowanie API w architekturze aplikacji webowych znacząco sprzyja ich skalowalności. Dzięki wyraźnemu oddzieleniu interfejsu API od warstwy prezentacji poszczególne komponenty systemu mogą być rozwijane, testowane i wdrażane niezależnie od siebie. Pozwala to na elastyczne reagowanie na rosnące wymagania oraz stopniowe rozbudowywanie funkcjonalności systemu.

Backend udostępniający API może być optymalizowany pod kątem wydajności, rozbudowywany o nowe funkcje lub skalowany poziomo bez konieczności wprowadzania zmian w interfejsie użytkownika. Takie podejście jest szczególnie istotne w dużych systemach oraz aplikacjach obsługujących rosnącą liczbę użytkowników, gdzie stabilność, dostępność i możliwość dalszego rozwoju mają kluczowe znaczenie.

Projektowanie i komunikacja z API



11

Zasady projektowania API

Projektowanie API polega na świadomym definiowaniu punktów końcowych, metod komunikacji oraz struktury danych. Dobrze zaprojektowane API powinno być czytelne, spójne i przewidywalne.

Jednym z najczęściej stosowanych podejść do projektowania API jest REST API (Representational State Transfer). REST opiera się na zasobach identyfikowanych za pomocą adresów URL oraz standardowych metodach HTTP, takich jak GET, POST, PUT czy DELETE, które jednoznacznie określają intencję operacji.



12

Rys. 3. Zasady stylu REST API

[źródło: <https://lukasz-zielinski.pl/blog/posts/2021-08-07-rest-api/>]



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Endpointy API

Endpointy API to konkretne adresy URL, pod którymi dostępne są określone funkcjonalności serwera. Każdy endpoint odpowiada za realizację konkretnego zadania, np. pobranie listy danych, dodanie nowego zasobu czy aktualizację istniejących informacji. Jasne i logiczne nazewnictwo endpointów znacząco wpływa na czytelność i użyteczność API.

ADRES API:

`http://www.super-kotki.pl/api`

ENDPOINTY:

`http://www.super-kotki.pl/api/lista`

`http://www.super-kotki.pl/api/dodaj`

`http://www.super-kotki.pl/api/kotek/{id}`

Rys. 4. Różnica między adresem API a adresem endpointu
[źródło: <http://www.infected.pl/programowanie/kursy-online-pozostale/kurs-testowanie-api/czym-sa-endpointy-w-api.html>]

13

Metody HTTP w API

API internetowe wykorzystują metody HTTP do określenia rodzaju operacji wykonywanej na zasobach. Metoda GET służy do pobierania danych, POST do ich tworzenia, PUT i PATCH do modyfikacji, a DELETE do usuwania. Poprawne stosowanie metod HTTP zwiększa zgodność API ze standardami sieci Web i ułatwia jego wykorzystanie.

METODA HTTP	AKCJA API	Działanie
GET	READ	Odczytanie danych z zasobu
POST	CREATE	Utworzenie wskazanego zasobu
PUT/PATCH	UPDATE	Aktualizacja istniejącego zasobu
DELETE	DELETE	Usunięcie zasobu

Rys. 5. Metody HTTP vs akcje API
[źródło: <https://www.nastykusieci.pl/podstawy-api/>]

14

Format danych w API

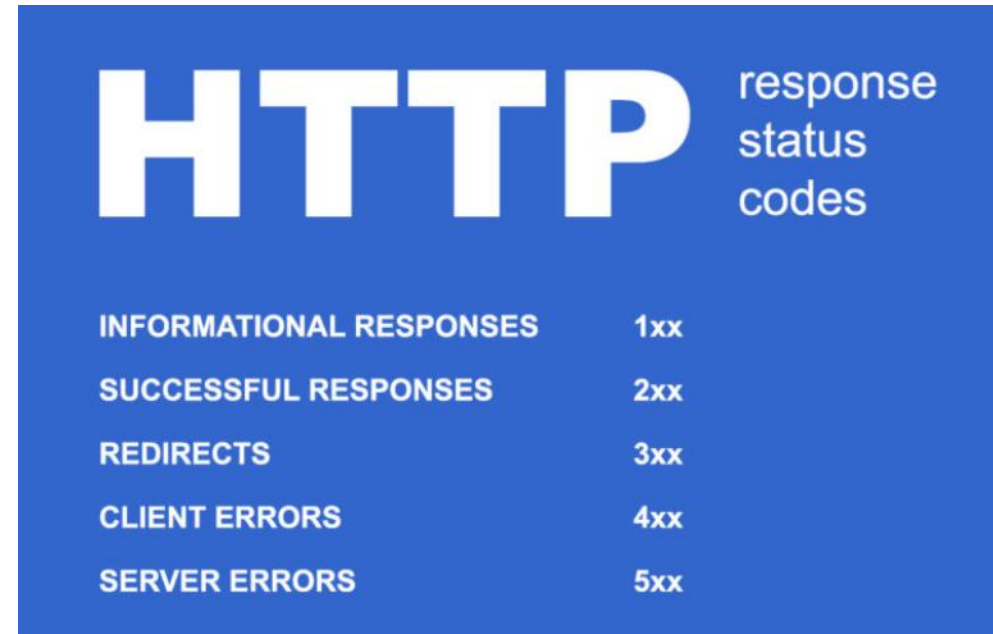
Dane przesyłane pomiędzy klientem a API muszą być zapisane w jasno określonym i uzgodnionym formacie, aby obie strony mogły je poprawnie interpretować. Najczęściej wykorzystywanym formatem jest JSON (JavaScript Object Notation), który charakteryzuje się niewielkim rozmiarem, czytelną strukturą oraz łatwością przetwarzania w wielu językach programowania. Dzięki temu jest on naturalnym wyborem dla nowoczesnych aplikacji webowych.

Jednolita i spójna struktura danych w API ułatwia implementację po stronie klienta oraz zmniejsza ryzyko błędów komunikacji. Dodatkowo sprzyja dalszemu rozwojowi API, ponieważ jasno zdefiniowane formaty odpowiedzi i żądań pozwalają na wprowadzanie zmian w sposób kontrolowany i przewidywalny, bez negatywnego wpływu na istniejących użytkowników interfejsu.

15

Kody statusu HTTP w API

Kody statusu HTTP pełnią ważną rolę w komunikacji API z klientem. Informują one o wyniku przetwarzania żądania, np. o sukcesie, błędzie użytkownika lub problemie po stronie serwera. Świadome używanie kodów statusu pozwala klientowi API odpowiednio reagować na różne sytuacje i poprawia jakość komunikacji.



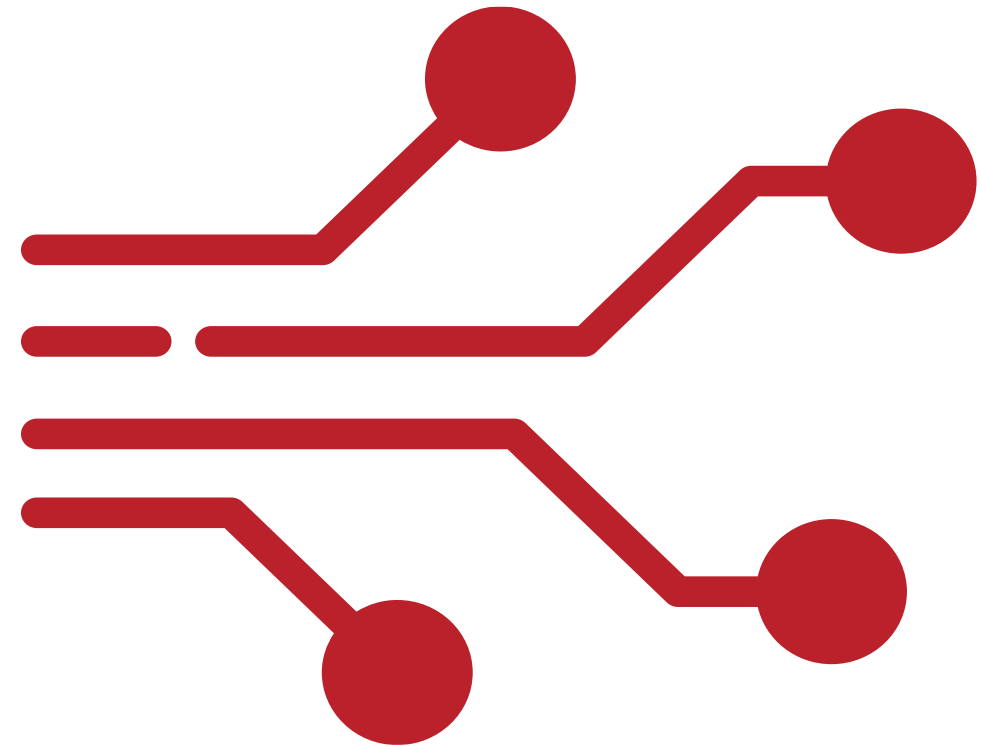
HTTP response status codes	
INFORMATIONAL RESPONSES	1xx
SUCCESSFUL RESPONSES	2xx
REDIRECTS	3xx
CLIENT ERRORS	4xx
SERVER ERRORS	5xx

Rys. 6. Kody odpowiedzi HTTP

[źródło: <https://dreamfocus.pl/kody-odpowiedzi-http/>]

16

Bezpieczeństwo i niezawodność API

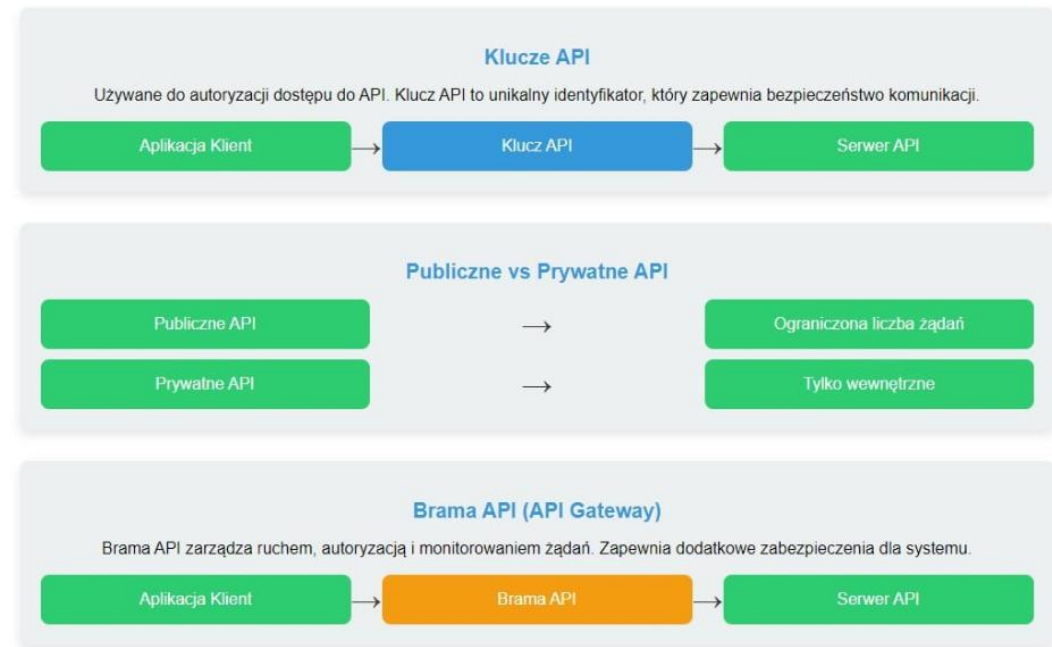


17

Bezpieczeństwo API

API jest często bezpośrednio wystawione na dostęp zewnętrzny, dlatego kwestie bezpieczeństwa mają kluczowe znaczenie. Serwer powinien kontrolować dostęp do zasobów, weryfikować dane wejściowe oraz chronić się przed nieautoryzowanymi żadaniami. Bezpieczne API stanowi fundament stabilnej i zaufanej aplikacji internetowej.

Bezpieczeństwo i dostęp do API



Rys. 7. Bezpieczeństwo i dostęp do API

[źródło: <https://devstockacademy.pl/blog/narzedzia-i-automatyzacja/co-to-jest-api-i-jak-dziala-kompletny-przewodnik-dla-programistow/?srsId=AfmBOopMTmknzKJNqvnT6FHsjtyhnbdkPsiiFVcYOBmQwPki242UHt>]

Walidacja danych w API

Walidacja danych w API jest jednym z kluczowych elementów bezpiecznego i stabilnego działania aplikacji. Każde dane przesyłane przez klienta muszą zostać sprawdzone po stronie serwera pod kątem poprawności formatu, zakresu wartości oraz kompletności. API nie może zakładać, że otrzymane informacje są prawidłowe, nawet jeśli wcześniej były weryfikowane po stronie klienta.

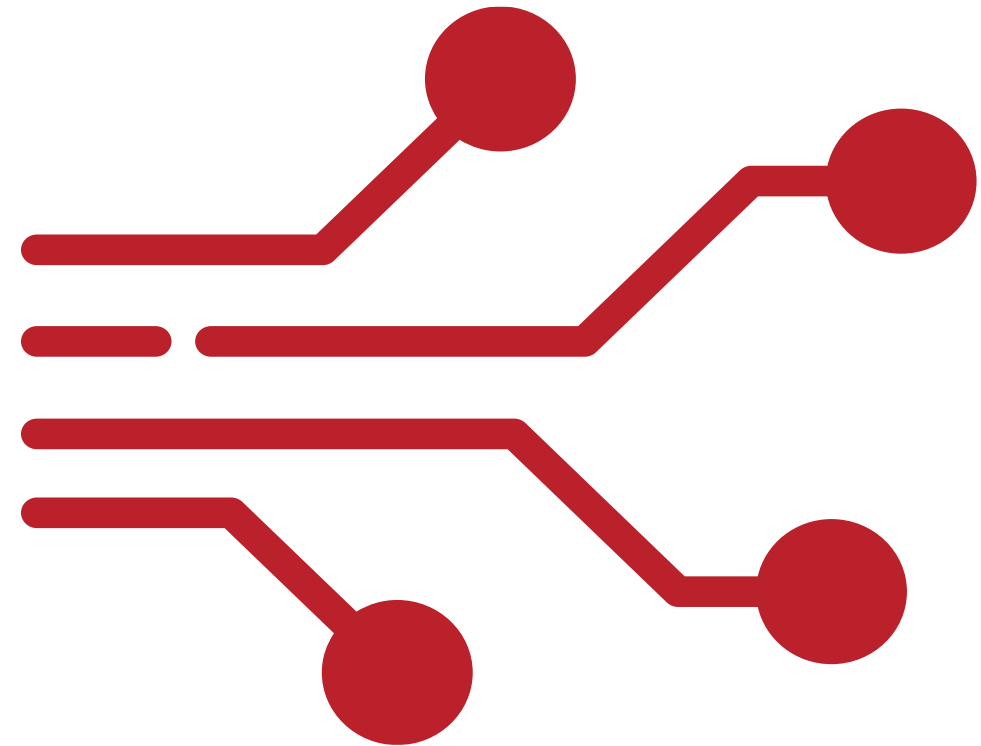
Poprawnie zaimplementowana walidacja chroni aplikację przed błędami logicznymi, zapisem nieprawidłowych danych oraz potencjalnymi atakami, takimi jak wstrzykiwanie danych czy próby obejścia zabezpieczeń. Dzięki temu API staje się bardziej odporne, przewidywalne i bezpieczne, co ma kluczowe znaczenie w systemach obsługujących wielu użytkowników i zewnętrznych klientów.

Obsługa błędów w API

Obsługa błędów w API powinna być zaprojektowana w sposób spójny, jednoznaczny i przewidywalny, tak aby klient dokładnie wiedział, co poszło nie tak. W przypadku wystąpienia błędu API powinno zwrócić odpowiedni kod statusu HTTP, który jasno określa rodzaj problemu, na przykład błąd po stronie klienta lub serwera. Ułatwia to poprawną obsługę błędów po stronie aplikacji korzystającej z API.

Odpowiedzi błędów powinny zawierać czytelne komunikaty, które pomagają programiście zrozumieć przyczynę problemu i poprawnie zareagować, na przykład poprzez korektę danych lub ponowienie żądania. Jednocześnie API nie powinno ujawniać szczegółów technicznych, takich jak struktura systemu czy treść wyjątków serwerowych, ponieważ mogłyby one zostać wykorzystane do ataków. Takie podejście zwiększa bezpieczeństwo, czytelność i profesjonalizm interfejsu API.

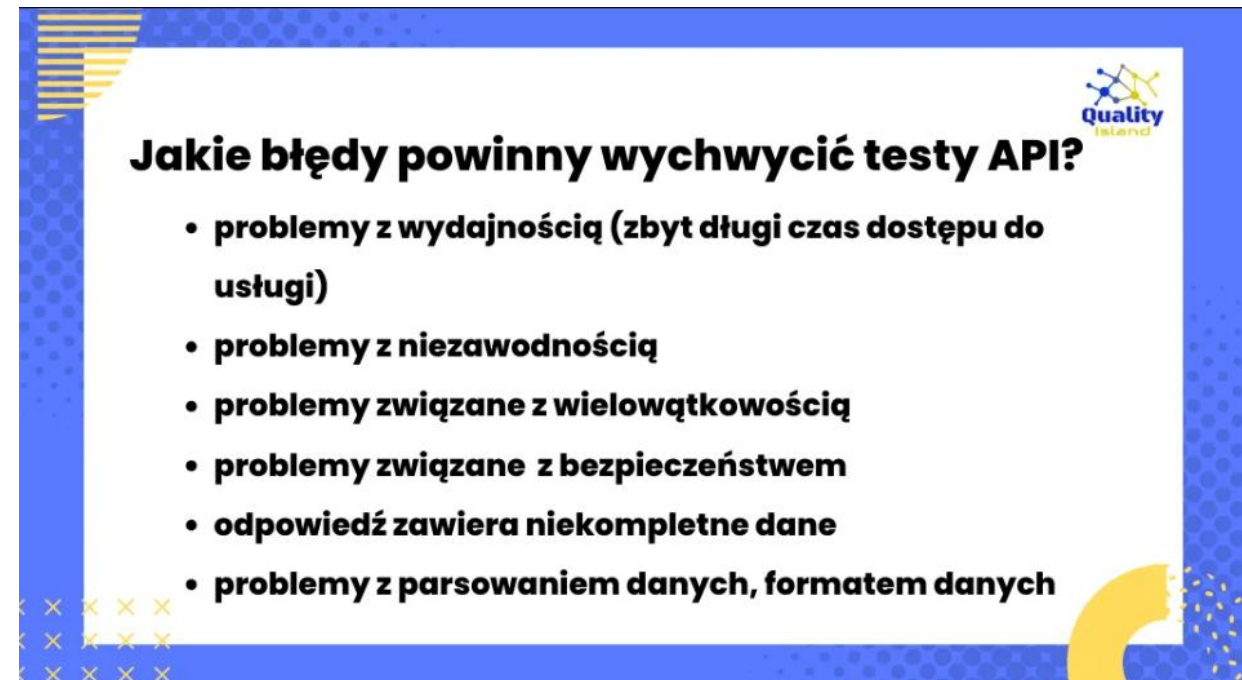
Testowanie i rozwój API



21

Testowanie API

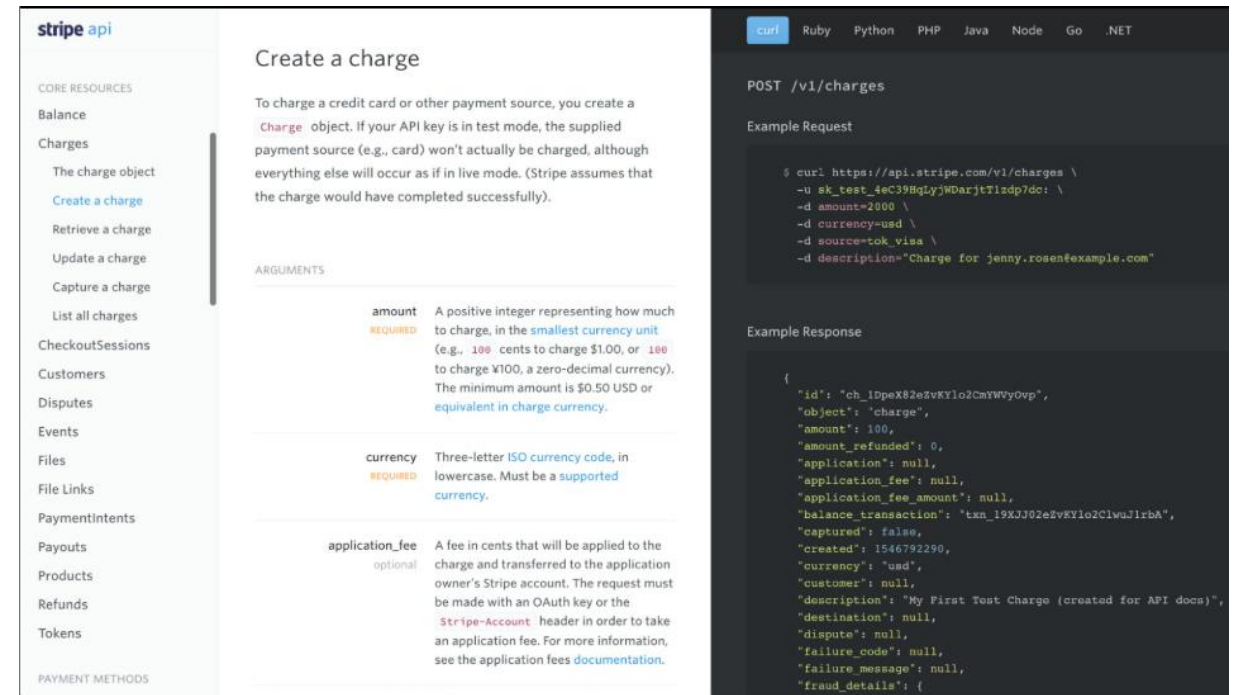
Testowanie API pozwala sprawdzić poprawność działania endpointów oraz zgodność odpowiedzi z założeniami projektowymi. Regularne testy umożliwiają szybkie wykrywanie błędów oraz zapewniają stabilność API w trakcie jego rozwoju. Jest to szczególnie istotne w przypadku API wykorzystywanego przez wiele różnych klientów.



Rys. 8. Najczęstsze błędy wychwytywane przez testy API
[źródło: <https://qualityisland.pl/testowanie-api-wprowadzenie/>]

Dokumentacja API

Dokumentacja API jest niezbędna, aby umożliwić innym programistom korzystanie z udostępnionych funkcji. Powinna ona opisywać dostępne endpointy, metody HTTP, strukturę danych oraz możliwe odpowiedzi. Dobra dokumentacja znacząco ułatwia integrację API z innymi systemami.



The screenshot shows the Stripe API documentation for the 'Create a charge' endpoint. On the left is a sidebar with a list of API resources. The main content area includes a description of the endpoint, a table of arguments (amount, currency, application_fee), and an 'Example Request' section showing a curl command. On the right, there is an 'Example Response' section showing a JSON object.

stripe api

Create a charge

To charge a credit card or other payment source, you create a `Charge` object. If your API key is in test mode, the supplied payment source (e.g., card) won't actually be charged, although everything else will occur as if in live mode. (Stripe assumes that the charge would have completed successfully).

ARGUMENTS

amount REQUIRED	A positive integer representing how much to charge, in the smallest currency unit (e.g., 100 cents to charge \$1.00, or 100 to charge ¥100, a zero-decimal currency). The minimum amount is \$0.50 USD or equivalent in charge currency.
currency REQUIRED	Three-letter ISO currency code, in lowercase. Must be a supported currency .
application_fee optional	A fee in cents that will be applied to the charge and transferred to the application owner's Stripe account. The request must be made with an OAuth key or the <code>Stripe-Account</code> header in order to take an application fee. For more information, see the application fees documentation .

Example Request

```
curl https://api.stripe.com/v1/charges \
-u sk_test_4eC398qLyjWDarjtT1zdp7dc: \
-d amount=2000 \
-d currency=usd \
-d source=tok_visa \
-d description="Charge for jenny.rosen@example.com"
```

Example Response

```
{
  "id": "ch_1DpeX82e2vKylo2CmYVYvYv",
  "object": "charge",
  "amount": 100,
  "amount_refunded": 0,
  "application": null,
  "application_fee": null,
  "application_fee_amount": null,
  "balance_transaction": "txn_19XJ302e2vKylo2CmYVYvYv",
  "captured": false,
  "created": 1546792290,
  "currency": "usd",
  "customer": null,
  "description": "My First Test Charge (created for API docs)",
  "dispute": null,
  "failure_code": null,
  "failure_message": null,
  "fraud_details": {
```

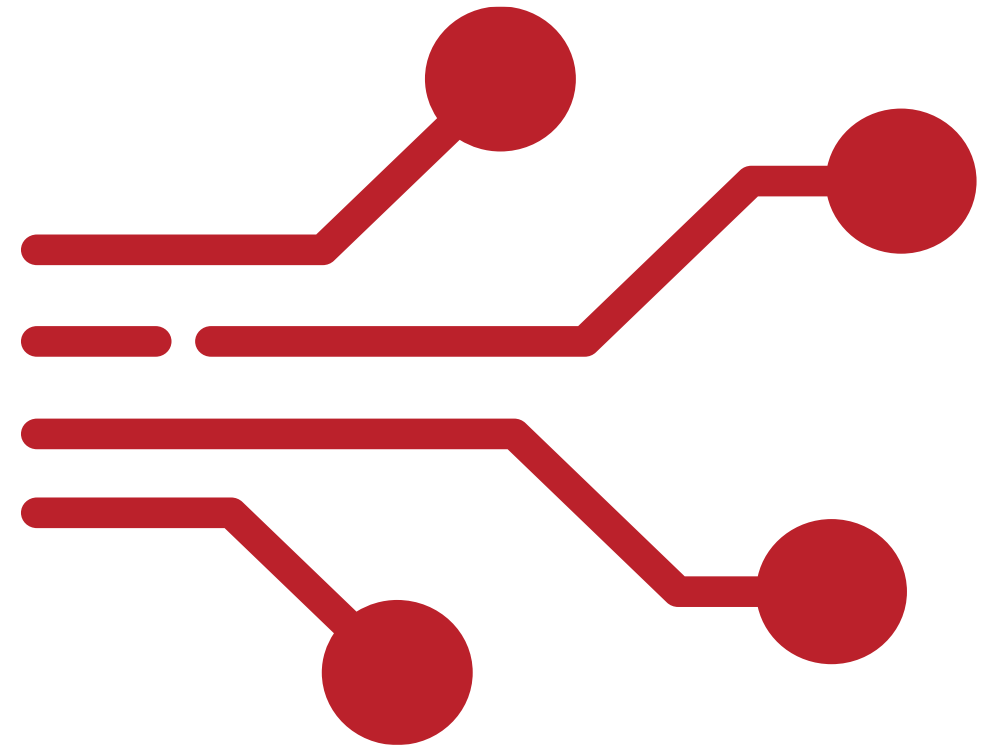
Rys. 9. Przykładowy fragment dokumentacji API
[źródło: <https://stoplight.io/api-documentation-guide>]

API jako element ekosystemu aplikacji

API bardzo często pełni rolę centralnego elementu większego ekosystemu aplikacji i usług. Stanowi ono wspólny punkt dostępu do danych i funkcjonalności backendu, z którego mogą korzystać różne aplikacje klienckie, takie jak aplikacje webowe, mobilne, desktopowe, a także systemy zewnętrzne należące do innych organizacji lub partnerów biznesowych. Dzięki temu jedna, spójna warstwa API może obsługiwać wiele różnych przypadków użycia.

Takie podejście sprzyja integracji systemów oraz dalszemu rozwojowi infrastruktury informatycznej. API umożliwia łączenie niezależnych usług, automatyzację procesów biznesowych oraz tworzenie nowych funkcjonalności bez konieczności ingerowania w istniejące aplikacje. W efekcie API staje się kluczowym narzędziem budowania elastycznych, skalowalnych i nowoczesnych systemów informatycznych, które mogą ewoluować wraz z potrzebami organizacji.

Podsumowanie



25

Tworzenie API dla aplikacji internetowych polega na zaprojektowaniu spójnego, czytelnego i bezpiecznego interfejsu komunikacyjnego pomiędzy klientem a serwerem. API umożliwia wyraźne oddzielenie logiki aplikacji od warstwy interfejsu użytkownika, co sprzyja elastyczności systemu oraz jego łatwiejszemu rozwojowi i utrzymaniu. Dzięki temu te same funkcjonalności backendu mogą być wykorzystywane przez różne typy klientów i integrowane z innymi systemami.

Duże znaczenie ma poprawne stosowanie metod HTTP, jednolitych formatów danych, rzetelnej walidacji oraz spójnej obsługi błędów. Elementy te wpływają bezpośrednio na bezpieczeństwo, stabilność i przewidywalność działania API. Dobrze zaprojektowane API stanowi fundament nowoczesnych, skalowalnych aplikacji webowych i jest jednym z najważniejszych komponentów współczesnych systemów informatycznych.

Literatura:

- [1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.
- [2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.
- [3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.
- [4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.
- [5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Dziękuję za uwagę

Wykład nr 11

Temat: Integracja z bazami danych i zarządzanie danymi. Bazy danych relacyjne i nierelacyjne. Tworzenie i optymalizacja zapytań. Tworzenie aplikacji typu CRUD

dr hab. inż. Michał Wydra, prof. Uczelni

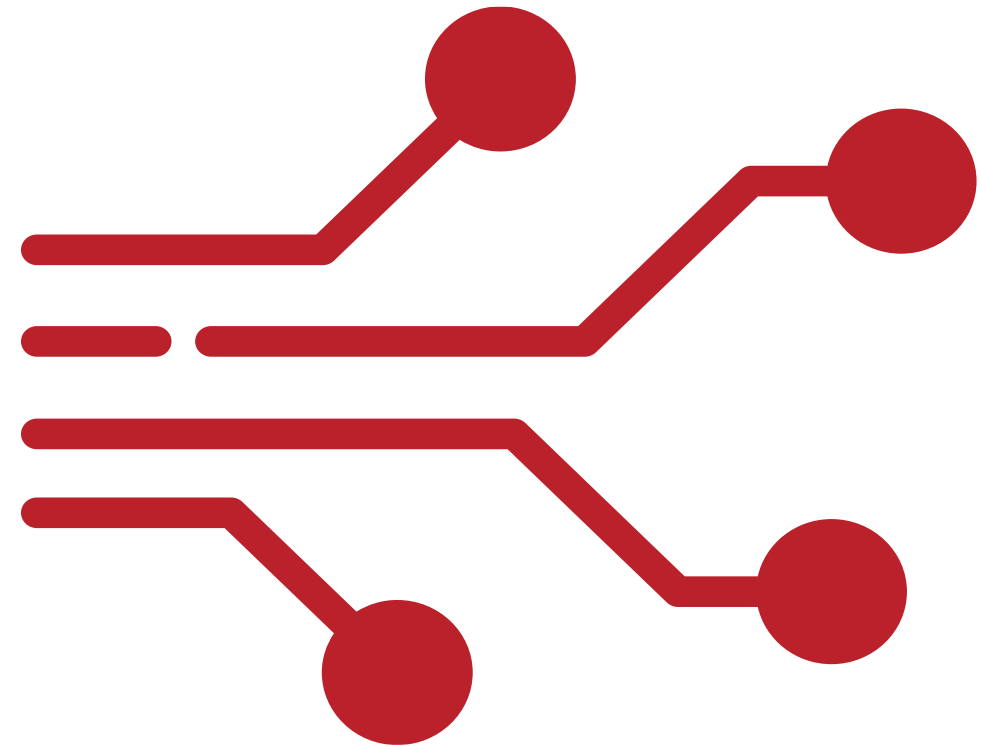
Katedra Informatyki Stosowanej

Plan prezentacji

- Rola baz danych w aplikacjach internetowych
- Relacyjne bazy danych
- Nierelacyjne bazy danych
- Zapytania do bazy danych i ich optymalizacja
- Aplikacje typu CRUD



Rola baz danych w aplikacjach internetowych



3

Dlaczego aplikacje webowe potrzebują baz danych?

Aplikacje webowe potrzebują baz danych, ponieważ umożliwiają one trwałe i uporządkowane przechowywanie informacji, które muszą być dostępne niezależnie od ponownego uruchomienia aplikacji lub serwera. Dane zapisane w bazie mogą być wielokrotnie odczytywane, modyfikowane i wykorzystywane przez różnych użytkowników w dowolnym momencie. Przykładami takich informacji są konta użytkowników, zamówienia, treści publikowane przez użytkowników, historię działań czy konfiguracje systemowe.

Bez bazy danych aplikacja byłaby ograniczona wyłącznie do danych tymczasowych przechowywanych w pamięci operacyjnej, które znikają po zakończeniu działania programu. Uniemożliwiłoby to logowanie użytkowników, zapisywanie postępów czy analizę danych w dłuższym czasie. Bazy danych stanowią więc fundament funkcjonalnych aplikacji webowych, zapewniając spójność, trwałość i bezpieczeństwo informacji oraz umożliwiając rozwój systemu wraz ze wzrostem liczby użytkowników i danych.

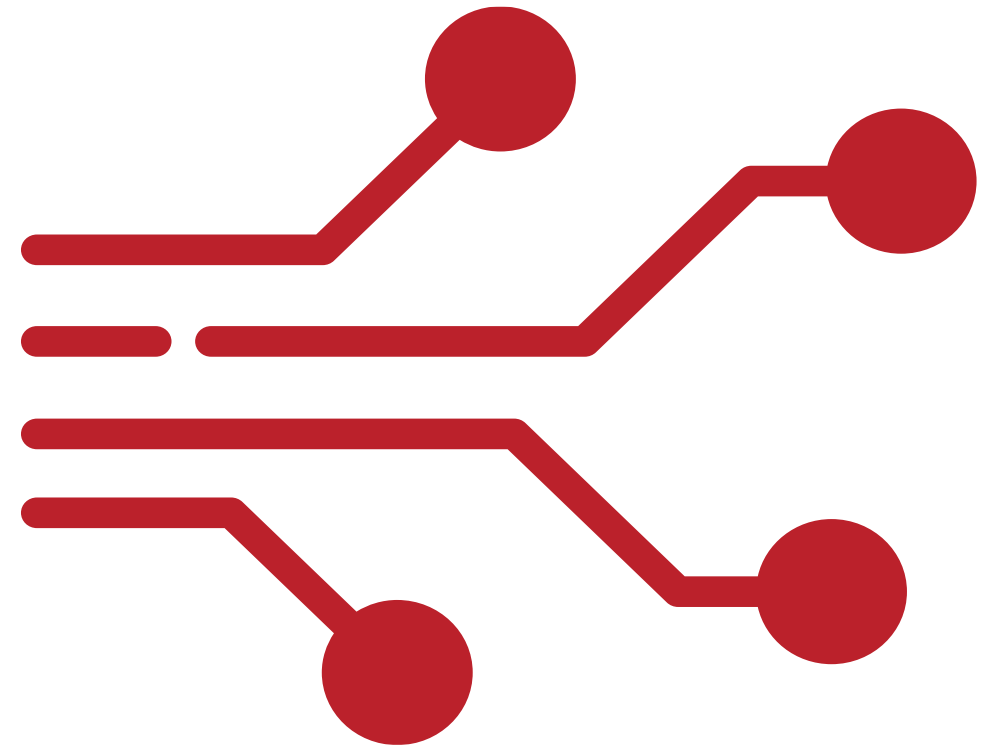
4

Warstwa dostępu do danych w backendzie

W architekturze aplikacji internetowej baza danych stanowi odrębną warstwę systemu, która nie jest dostępna bezpośrednio dla użytkownika ani dla frontendu. Komunikacja z bazą danych odbywa się wyłącznie za pośrednictwem backendu, który pełni rolę kontrolowanego pośrednika pomiędzy klientem a przechowywanymi danymi. Takie rozwiązanie zapobiega nieautoryzowanemu dostępowi i ogranicza ryzyko błędów oraz nadużyć.

Backend odpowiada za sposób odczytu, zapisu i modyfikacji danych, a także za egzekwowanie zasad logiki biznesowej, takich jak uprawnienia użytkowników czy spójność informacji. Centralne zarządzanie dostępem do danych zwiększa bezpieczeństwo aplikacji, ułatwia jej rozwój oraz pozwala na wprowadzanie zmian w strukturze bazy danych bez wpływu na działanie interfejsu użytkownika.

Relacyjne bazy danych

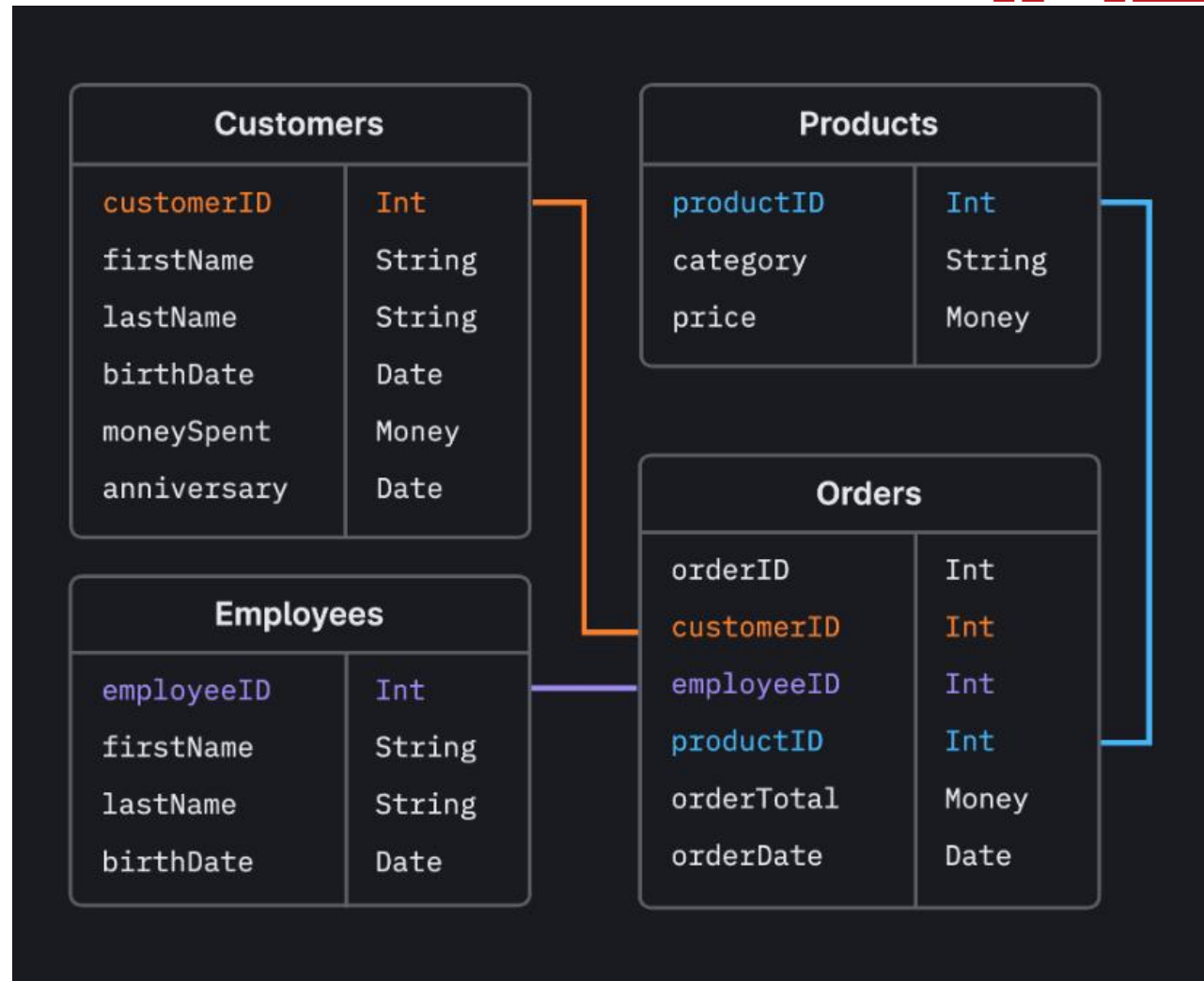


6

Czym są relacyjne bazy danych?

Relacyjne bazy danych to systemy przechowywania informacji oparte na modelu relacyjnym, w którym dane organizowane są w postaci tabel. Każda tabela składa się z wierszy i kolumn, gdzie wiersz reprezentuje pojedynczy rekord, na przykład użytkownika lub zamówienie, a kolumny opisują jego cechy, takie jak identyfikator, nazwa czy data utworzenia. Taki sposób zapisu danych jest czytelny i łatwy do zrozumienia.

Struktura relacyjnych baz danych jest ściśle zdefiniowana, co oznacza, że każda tabela ma określony schemat i typy danych. Sprzyja to zachowaniu spójności i integralności informacji, a także umożliwia definiowanie relacji pomiędzy tabelami, na przykład poprzez klucze główne i obce. Dzięki temu relacyjne bazy danych doskonale sprawdzają się w aplikacjach wymagających uporządkowanych danych, niezawodności oraz precyzyjnych reguł biznesowych.



Rys. 1. Przykład relacyjnej bazy danych [źródło: <https://planetscale.com/blog/schema-design-101-relational-databases>]

Tabele, klucze i relacje

W relacyjnych bazach danych główną rolę odgrywają klucze główne oraz klucze obce, które pozwalają porządkować dane i tworzyć pomiędzy nimi logiczne powiązania. Klucz główny to unikalny identyfikator rekordu w tabeli – dzięki niemu każdy wiersz może być jednoznacznie rozpoznany, niezależnie od pozostałych danych. Zapewnia to spójność oraz umożliwia bezpieczne odwoływanie się do konkretnych rekordów.

Klucz obcy służy natomiast do tworzenia relacji pomiędzy tabelami, wskazując na klucz główny w innej tabeli. Dzięki temu możliwe jest modelowanie rzeczywistych zależności pomiędzy danymi, takich jak relacja jeden-do-wielu (np. użytkownik i jego zamówienia) czy wiele-do-wielu (np. użytkownicy i role). Mechanizm kluczy i relacji pozwala zachować integralność danych oraz tworzyć złożone, ale logicznie spójne struktury informacyjne.

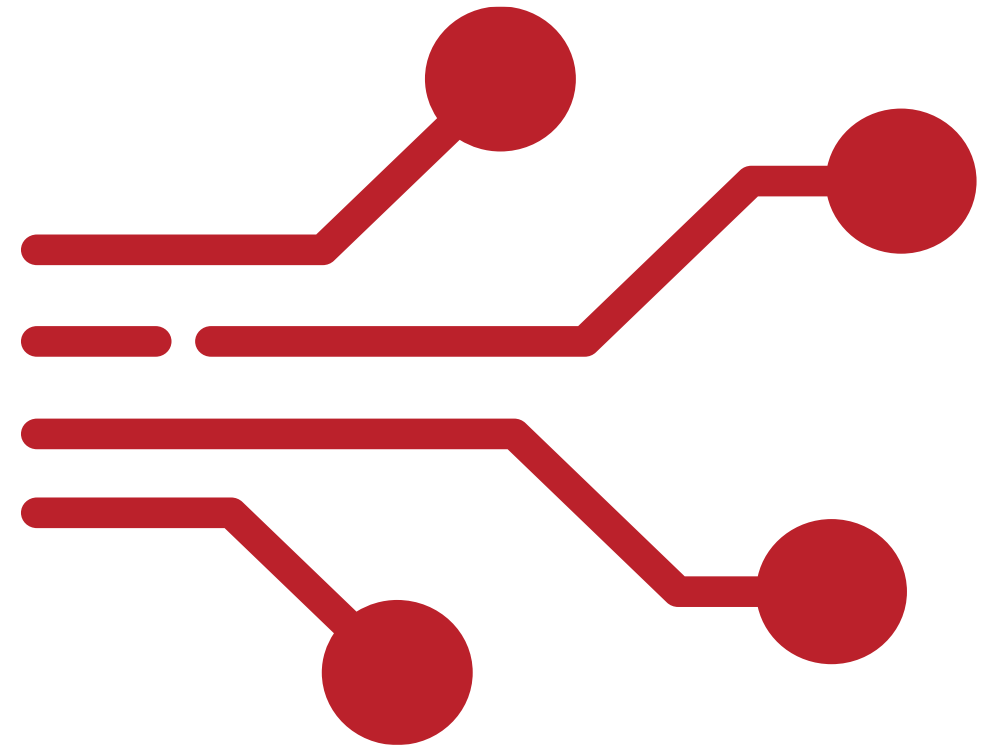
Zalety i ograniczenia baz relacyjnych

Relacyjne bazy danych oferują wiele istotnych zalet, z których najważniejszą jest wysoka spójność i integralność danych. Dzięki ściśle określonymu schematowi, kluczom oraz relacjom możliwe jest bezpieczne przechowywanie informacji i zapobieganie niespójnościom. Dodatkowo relacyjne bazy danych umożliwiają tworzenie złożonych zapytań, które pozwalają łączyć dane z wielu tabel, filtrować je i analizować w precyzyjny sposób. Sprawia to, że są one szczególnie dobrze przystosowane do systemów biznesowych i aplikacji, w których struktura danych jest stabilna i jasno określona.

Ograniczeniem baz relacyjnych może być jednak mniejsza elastyczność w sytuacjach, gdy schemat danych często się zmienia. Każda modyfikacja struktury tabel wymaga zazwyczaj dodatkowych operacji i może wpływać na istniejące dane oraz zapytania. Przy bardzo rozbudowanych modelach danych rośnie także złożoność projektu i utrzymania bazy, co może wpływać na wydajność i czas rozwoju systemu. Z tego powodu w niektórych zastosowaniach relacyjne bazy danych bywają uzupełniane lub zastępowane innymi rozwiązaniami.

10

Nierelacyjne bazy danych



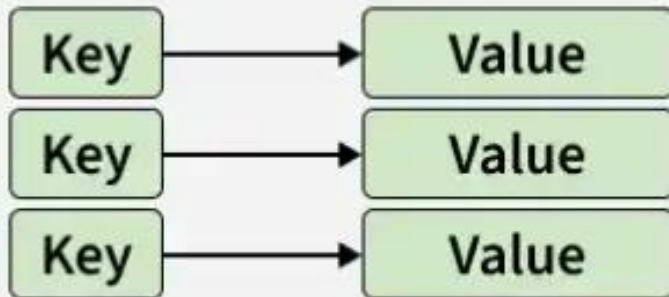
11

Czym są nierelacyjne bazy danych?

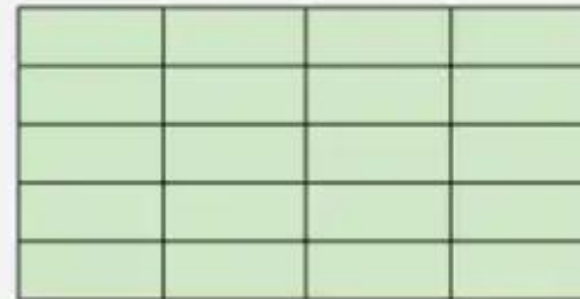
Bazy danych nierelacyjne, często określane mianem NoSQL, to systemy przechowywania danych, które nie opierają się na klasycznym modelu tabel, wierszy i kolumn. Zamiast tego dane mogą być zapisywane w różnych formach, takich jak dokumenty (np. w formacie JSON), pary klucz–wartość, grafy czy struktury kolumnowe. Taki sposób organizacji danych pozwala lepiej dopasować je do specyfiki aplikacji i rodzaju przetwarzanych informacji.

Główną zaletą baz nierelacyjnych jest duża elastyczność struktury danych. Nie wymagają one sztywno zdefiniowanego schematu, co ułatwia wprowadzanie zmian i rozwijanie aplikacji w czasie. Dzięki temu bazy NoSQL są często wykorzystywane w systemach o dynamicznie zmieniających się danych, dużej skali oraz w aplikacjach wymagających wysokiej wydajności i łatwego skalowania.

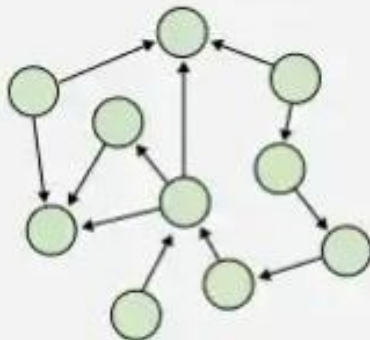
Key-Value



Column-Family



Graph



Document



Rys. 2. Przykłady
nierelacyjnych baz danych
[źródło:
<https://www.geeksforgeeks.org/dbms/types-of-nosql-databases/>]

13

Zastosowania baz NoSQL

Bazy NoSQL znajdują szerokie zastosowanie w aplikacjach, które muszą obsługiwać dużą liczbę użytkowników oraz przetwarzać ogromne wolumeny danych. Dzięki swojej architekturze są łatwe do skalowania poziomego, co pozwala na rozbudowę systemu poprzez dodawanie kolejnych serwerów. Sprawia to, że dobrze sprawdzają się w środowiskach o dynamicznie rosnącym obciążeniu.

Są one szczególnie przydatne w systemach, w których struktura danych często się zmienia lub nie jest jednoznacznie określona. Przykładami takich zastosowań są aplikacje społecznościowe, systemy analityczne, platformy przetwarzania danych w czasie rzeczywistym czy usługi gromadzące dane z wielu źródeł. Elastyczność modeli NoSQL umożliwia szybkie dostosowanie się do nowych wymagań bez konieczności przebudowy całej struktury bazy danych.

Porównanie baz relacyjnych i nierelacyjnych

Wybór pomiędzy bazą relacyjną a nierelacyjną zależy przede wszystkim od charakteru aplikacji, rodzaju danych oraz wymagań dotyczących spójności i wydajności. Bazy relacyjne sprawdzają się tam, gdzie kluczowe są silne mechanizmy spójności, jasno zdefiniowana struktura danych oraz złożone relacje między tabelami, na przykład w systemach finansowych czy biznesowych. Zapewniają one wysoką integralność danych i precyzyjne reguły ich przetwarzania.

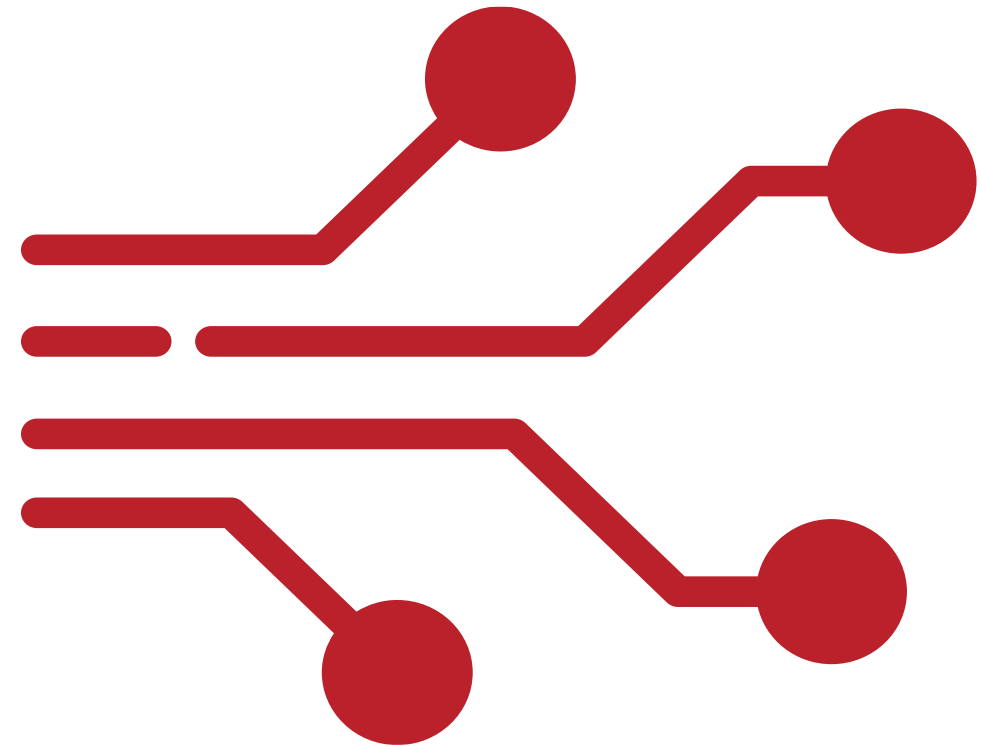
Bazy nierelacyjne natomiast oferują większą elastyczność w przechowywaniu danych oraz łatwiejszą skalowalność, co czyni je dobrym wyborem dla aplikacji o dynamicznie zmieniającej się strukturze danych lub bardzo dużej liczbie użytkowników. W praktyce wiele nowoczesnych systemów łączy oba podejścia, wykorzystując bazy relacyjne do danych krytycznych oraz bazy NoSQL do przechowywania danych pomocniczych, cache'owania czy analizy dużych wolumenów informacji.

CECHA	BAZA SQL	BAZA NOSQL
Rodzaj	Baza relacyjna	Baza nierelacyjna
Struktura	W bazie SQL dane są przechowywane w tabelach o określonej liczbie kolumn	Bazy nierelacyjne mogą mieć strukturę: dokumentową, grafową, klucz - wartość
Rodzaj struktury danych	Stała struktura danych	Zmienna struktura danych
Skalowalność	Wertykalna	Horyzontalna
Zapytania	Przeważnie jest to SQL	Sposób zapytań do bazy danych NoSQL zależy głównie od określonej bazy danych. Nie ma deklaratywnego języka zapytań.
Najważniejsze cechy	Integralność, spójność i stabilność danych	Skalowalność, elastyczność i szybkie zapytania

Rys. 3. Przykłady nierelacyjnych baz danych
[źródło: <https://nearshore-it.eu/pl/artykuly/nosql-vs-sql-bazy-danych/>]

16

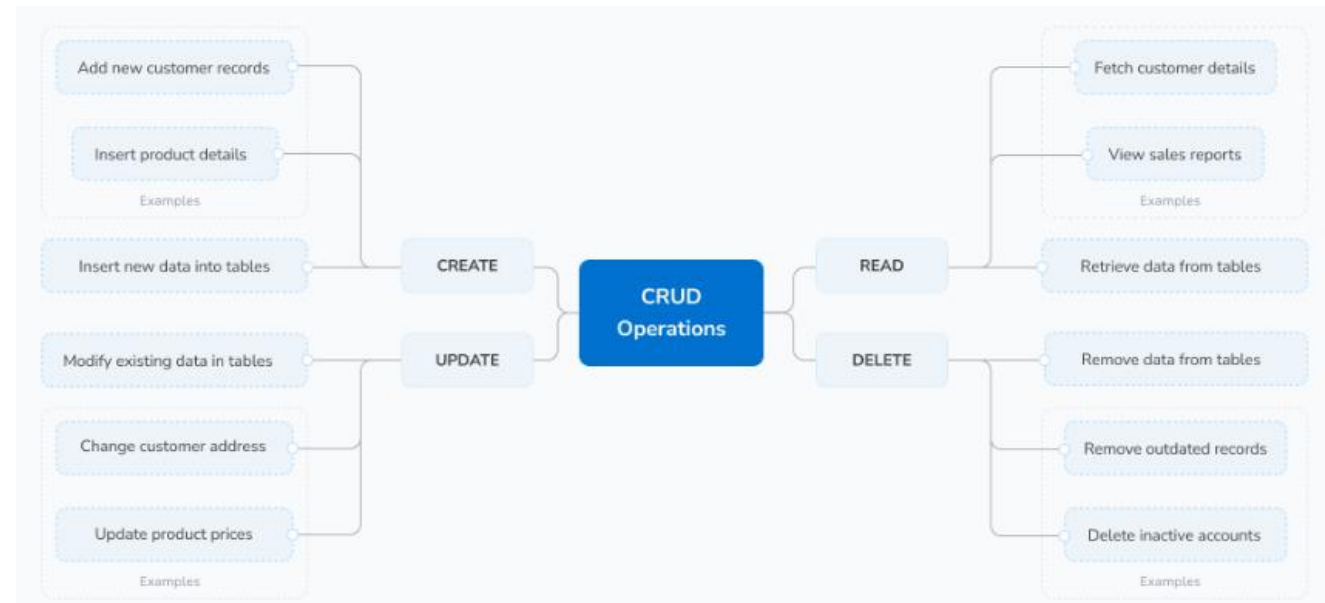
Zapytania do bazy danych i ich optymalizacja



17

Język zapytań i operacje na danych

Do komunikacji z relacyjnymi bazami danych wykorzystywany jest język SQL. Umożliwia on tworzenie struktur danych, dodawanie nowych rekordów, odczyt danych, ich aktualizację oraz usuwanie. Operacje te stanowią podstawę zarządzania danymi w aplikacjach webowych.

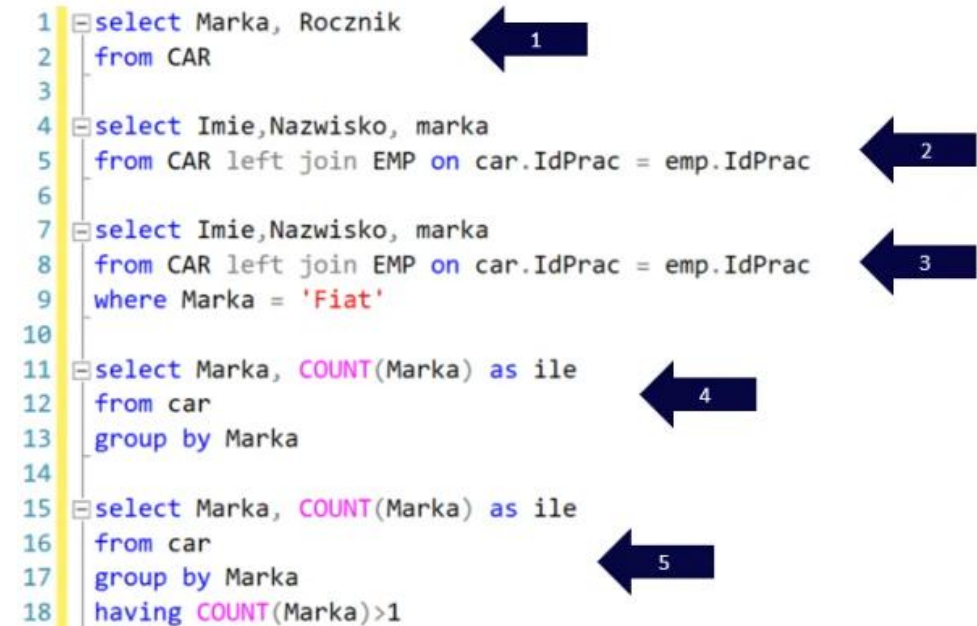


Rys. 4. Operacje CRUD [źródło: <https://www.devart.com/dbforge/sql/sqlcomplete/crud-operations-in-sql.html>]

Tworzenie zapytań SELECT

Zapytania SELECT służą do pobierania danych z bazy danych. Mogą być proste, zwracające wszystkie rekordy z tabeli, lub bardziej złożone, zawierające warunki, sortowanie czy łączenie danych z wielu tabel. Umiejętność tworzenia poprawnych zapytań SELECT jest kluczowa w pracy backend developera.

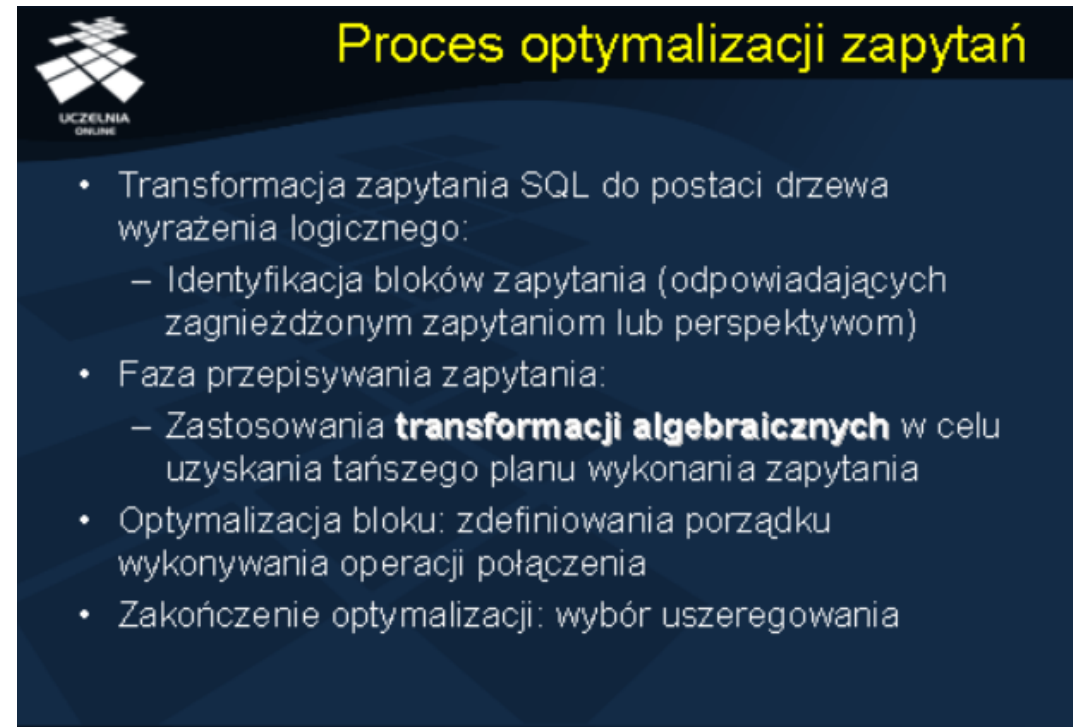
```
1 select Marka, Rocznik  
2 from CAR  
3  
4 select Imie, Nazwisko, marka  
5 from CAR left join EMP on car.IdPrac = emp.IdPrac  
6  
7 select Imie, Nazwisko, marka  
8 from CAR left join EMP on car.IdPrac = emp.IdPrac  
9 where Marka = 'Fiat'  
10  
11 select Marka, COUNT(Marka) as ile  
12 from car  
13 group by Marka  
14  
15 select Marka, COUNT(Marka) as ile  
16 from car  
17 group by Marka  
18 having COUNT(Marka) > 1
```



Rys. 5. Przykłady zapytań SELECT [źródło: <https://expose.pl/sql-co-to-jest-i-do-czego-sluzy/>]

Optymalizacja zapytań do bazy danych

Nieoptymalne zapytania mogą znacząco obniżyć wydajność aplikacji. Optymalizacja polega m.in. na ograniczaniu liczby pobieranych danych, stosowaniu indeksów oraz odpowiednim projektowaniu struktury tabel. Dobrze zoptymalizowane zapytania przyczyniają się do szybszego działania całego systemu.



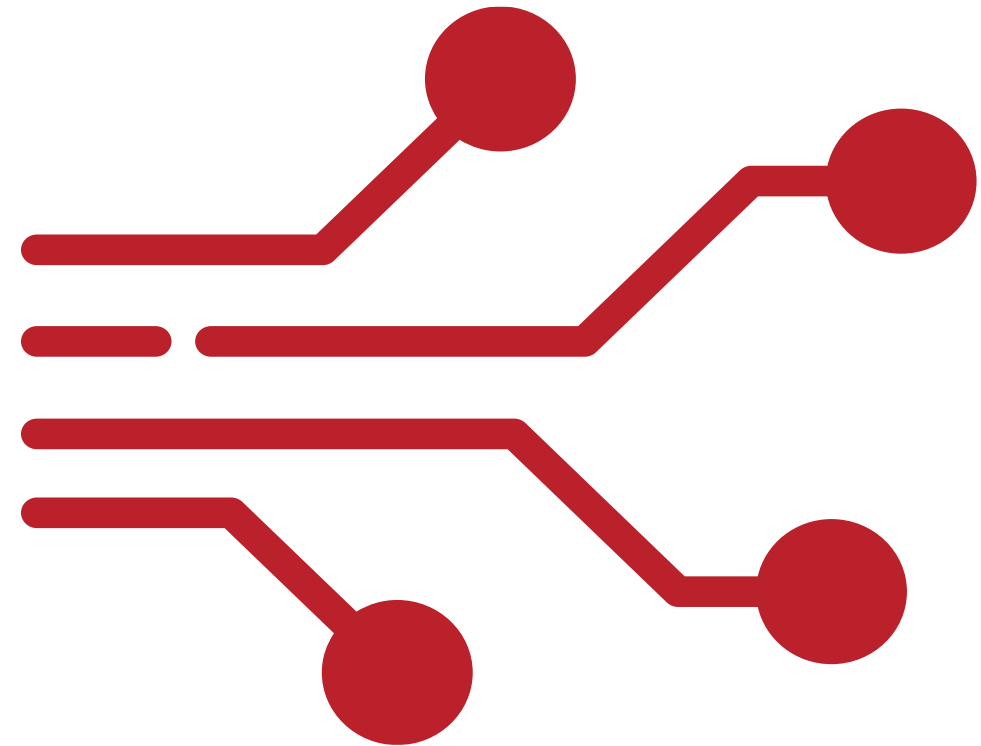
Proces optymalizacji zapytań

- Transformacja zapytania SQL do postaci drzewa wyrażenia logicznego:
 - Identyfikacja bloków zapytania (odpowiadających zagnieżdżonym zapytaniom lub perspektywom)
- Faza przepisywania zapytania:
 - Zastosowania **transformacji algebraicznych** w celu uzyskania tańszego planu wykonania zapytania
- Optymalizacja bloku: zdefiniowania porządku wykonywania operacji połączenia
- Zakończenie optymalizacji: wybór uszeregowania

Rys. 6. Proces optymalizacji zapytań [źródło: http://smurf.mimuw.edu.pl/external_slides/W12_Optymalizacja_zapytan_czesc_I/W12_Optymalizacja_zapytan_czesc_I.html]

20

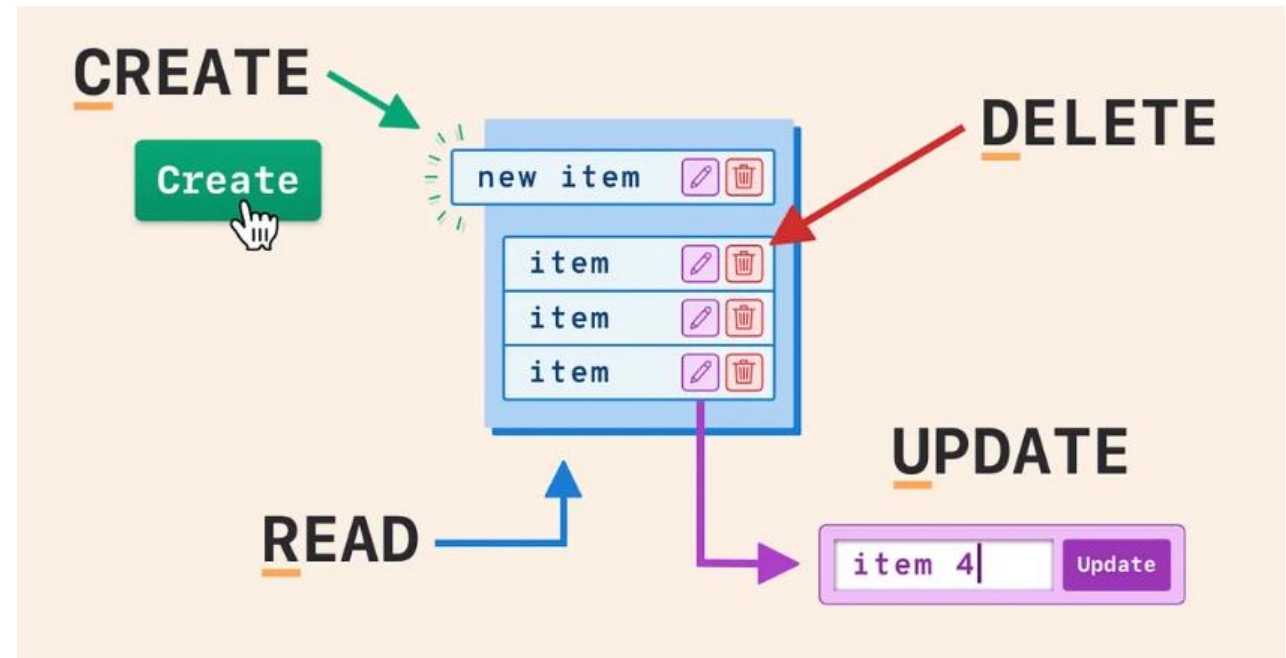
Aplikacje typu CRUD



21

Czym jest CRUD?

CRUD to akronim od czterech podstawowych operacji wykonywanych na danych: Create, Read, Update i Delete. Są one fundamentem większości aplikacji internetowych, ponieważ umożliwiają pełne zarządzanie danymi przechowywanymi w bazie.

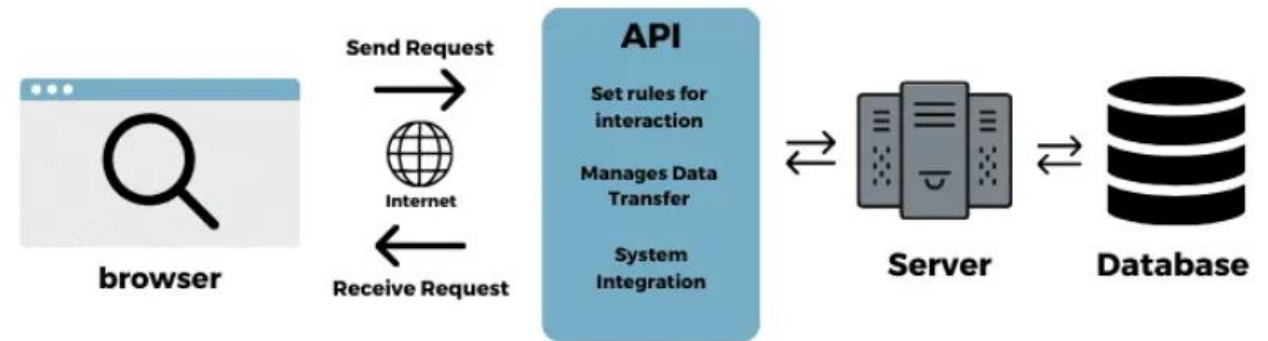


Rys. 7. Operacje CRUD [źródło:
<https://medium.com/geekculture/crud-operations-explained-2a44096e9c88>]

22

CRUD a aplikacje webowe

W aplikacjach webowych operacje CRUD są zazwyczaj realizowane poprzez backend i udostępniane klientowi w postaci API. Frontend wysyła żądania, a serwer wykonuje odpowiednie operacje na bazie danych, zachowując kontrolę nad poprawnością i bezpieczeństwem danych.

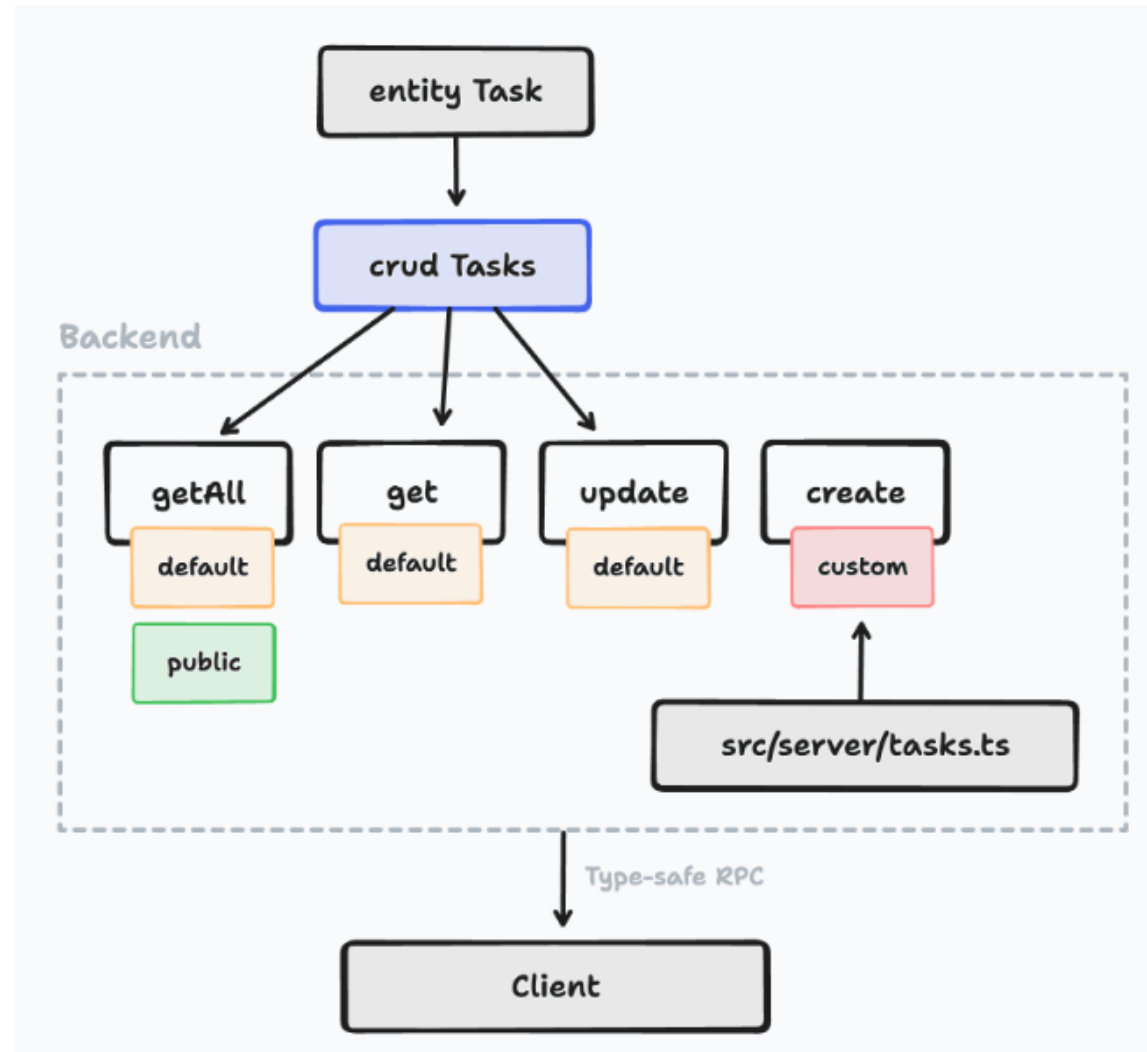


Rys. 8. Diagram integracji API [źródło: <https://kapucuonur.medium.com/integrating-frontend-and-backend-with-apis-a-comprehensive-guide-9d296eef2e33>]

Implementacja CRUD po stronie serwera

Implementacja CRUD po stronie serwera polega na zdefiniowaniu w backendzie zestawu funkcji lub endpointów API, które odpowiadają podstawowym operacjom na danych: tworzeniu (Create), odczytowi (Read), aktualizacji (Update) oraz usuwaniu (Delete). Każdy endpoint jest powiązany z konkretną akcją wykonywaną na bazie danych, na przykład dodaniem nowego rekordu, pobraniem listy zasobów, edycją istniejących informacji lub usunięciem wybranego elementu.

Takie podejście zapewnia przejrzystość i spójność działania systemu, ponieważ jasno określa, jakie operacje są dostępne oraz w jaki sposób można z nich korzystać. Dodatkowo backend może centralnie kontrolować walidację danych, obsługę błędów oraz uprawnienia użytkownika, co zwiększa bezpieczeństwo i ułatwia utrzymanie aplikacji wraz z jej rozwojem.



Rys. 9. Wizualizacja implementacji poleceń CRUD [źródło: <https://wasp.sh/docs/data-model/crud>]

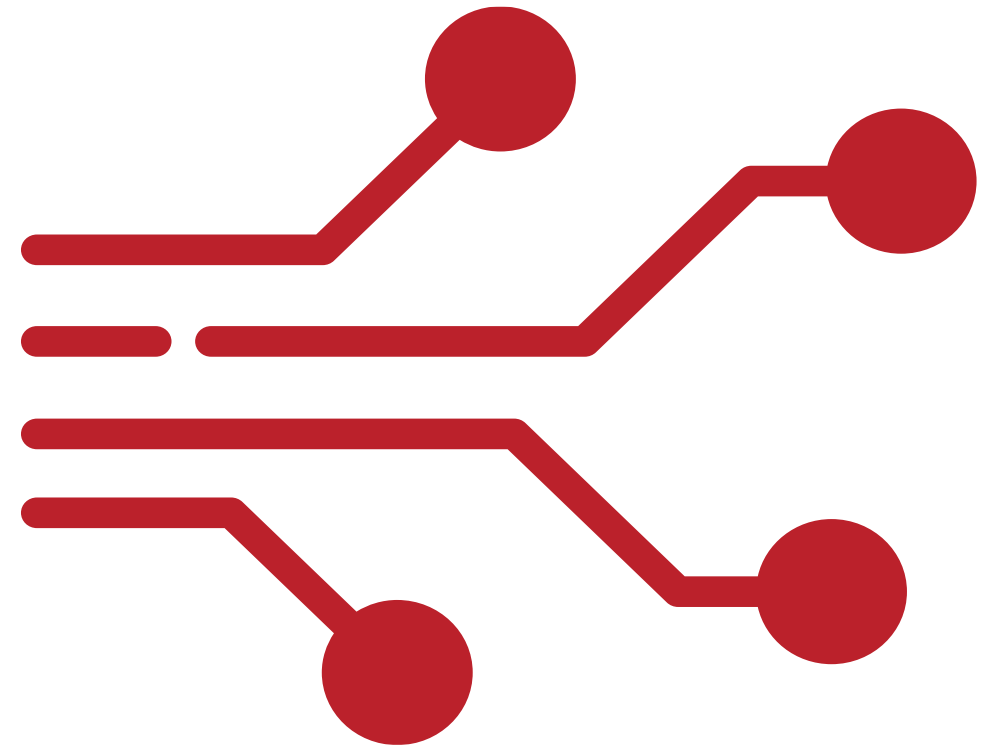
25

Walidacja i bezpieczeństwo operacji CRUD

Walidacja i bezpieczeństwo są niezbędne przy implementacji operacji CRUD, ponieważ każda z nich ingeruje w dane aplikacji. Zanim backend utworzy, zaktualizuje lub usunie rekord, powinien sprawdzić poprawność danych wejściowych: ich format, wymagane pola, zakres wartości oraz spójność z logiką biznesową. Pozwala to uniknąć zapisu błędnych informacji, które mogłyby później powodować problemy w działaniu systemu.

Równie istotna jest autoryzacja, czyli weryfikacja, czy użytkownik ma prawo wykonać daną operację na konkretnych danych. Backend powinien kontrolować uprawnienia, np. czy użytkownik może edytować tylko własne zasoby, czy ma dostęp administracyjny do danych innych osób. Takie podejście chroni aplikację przed nieautoryzowanym dostępem, ogranicza ryzyko nadużyć i zapewnia bezpieczeństwo całego systemu.

Podsumowanie



27

Integracja z bazami danych stanowi jeden z istotnych elementów tworzenia nowoczesnych aplikacji internetowych, ponieważ umożliwia trwałe przechowywanie i przetwarzanie informacji. Zarówno relacyjne, jak i nierelacyjne bazy danych oferują odmienne podejścia do organizacji danych, a wybór odpowiedniego rozwiązania zależy od charakteru aplikacji, jej skali oraz wymagań dotyczących spójności i elastyczności. W praktyce wiele systemów łączy oba typy baz, wykorzystując ich mocne strony w różnych obszarach.

Dla backend developera szczególnie istotna jest umiejętność projektowania struktur danych, tworzenia i optymalizacji zapytań oraz poprawnej implementacji operacji CRUD. Połączenie tych kompetencji z dbałością o walidację i bezpieczeństwo pozwala budować aplikacje webowe, które są wydajne, niezawodne, bezpieczne i gotowe na dalszy rozwój oraz skalowanie.

Literatura:

- [1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.
- [2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.
- [3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.
- [4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.
- [5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Dziękuję za uwagę



Wykład nr 12

Temat: Zarządzanie sesją i uwierzytelnianie

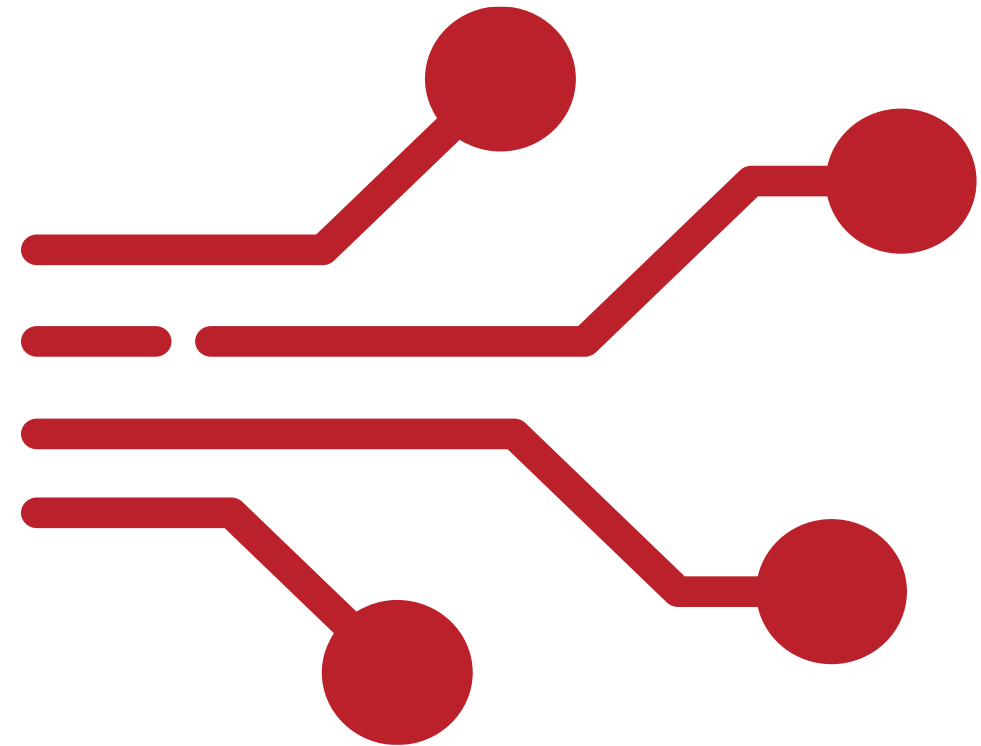
dr hab. inż. Michał Wydra, prof. Uczelni
Katedra Informatyki Stosowanej

Plan prezentacji

- Podstawy bezpieczeństwa i identyfikacji użytkownika
- Sesja użytkownika
- Mechanizmy uwierzytelniania
- Tokeny i nowoczesne podejścia
- Zagrożenia i dobre praktyki

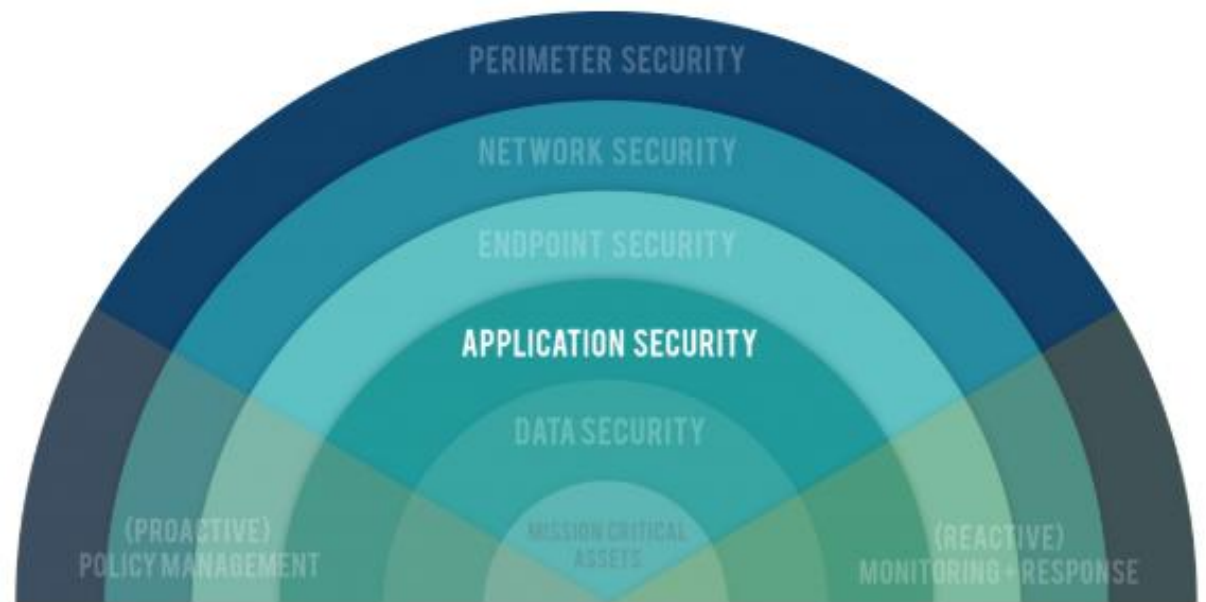


Podstawy bezpieczeństwa i identyfikacji użytkownika



Bezpieczeństwo jako element aplikacji webowej

Bezpieczeństwo aplikacji internetowych jest zagadnieniem przekrojowym, obejmującym wszystkie warstwy systemu – od interfejsu użytkownika, przez backend, aż po bazę danych i infrastrukturę sieciową. Jednym z najważniejszych aspektów bezpieczeństwa jest kontrola tego, kto korzysta z aplikacji oraz do jakich zasobów ma dostęp. Mechanizmy uwierzytelniania i zarządzania sesją pełnią w tym kontekście kluczową rolę.



Rys. 1. Bezpieczeństwo aplikacji – jedna z warstw bezpieczeństwa danych [źródło: <https://blog.cspire.com/7-layers-of-data-security-application>]

Uwierzytelnianie a autoryzacja

Uwierzytelnianie polega na sprawdzeniu, czy użytkownik jest tym, za kogo się podaje, najczęściej na podstawie loginu i hasła.

Autoryzacja natomiast określa, do jakich zasobów i operacji dany użytkownik ma dostęp.

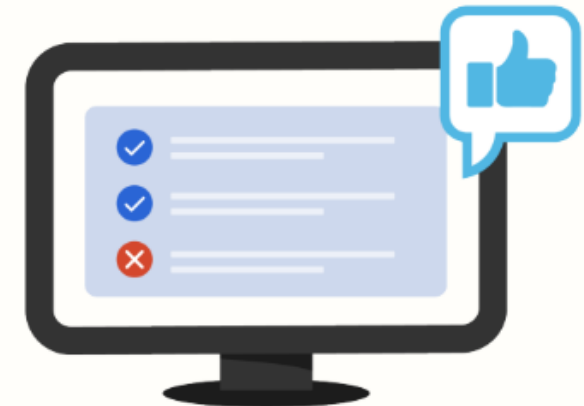
Rozróżnienie tych dwóch pojęć jest niezwykle istotne, ponieważ poprawne uwierzytelnienie użytkownika nie oznacza jeszcze, że ma on prawo do wykonania każdej operacji w systemie.

Authentication



Confirms users
are who they say they are.

Authorization

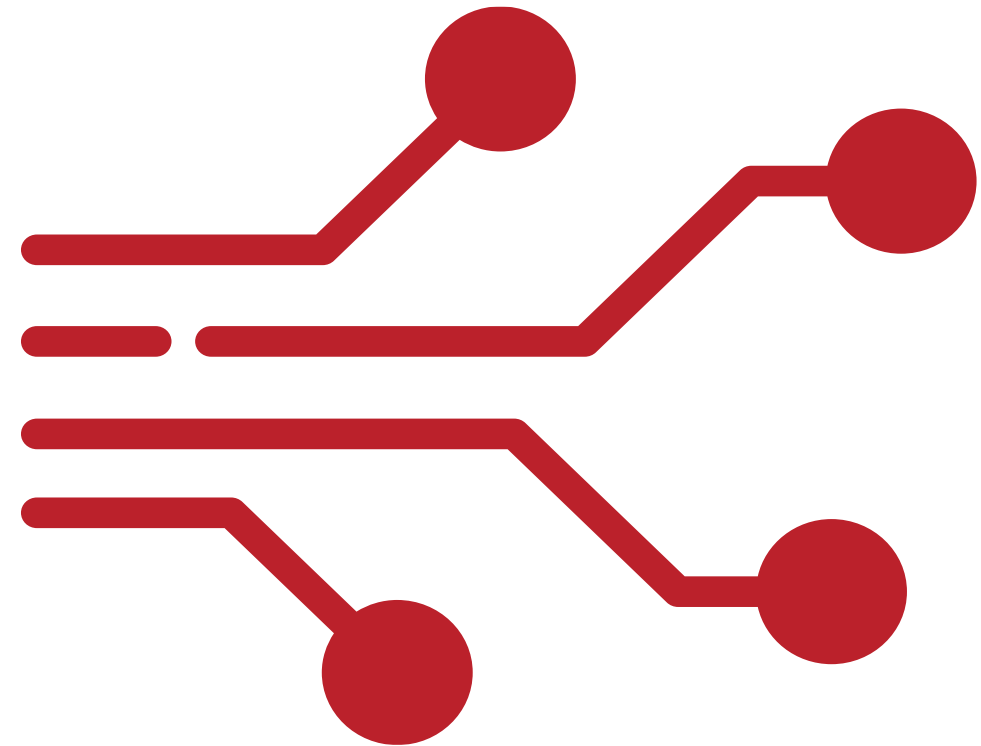


Gives users permission
to access a resource.

Rys. 2. Uwierzytelnienie a autoryzacja [źródło:
<https://www.okta.com/identity-101/authentication-vs-authorization/>]

5

Sesja użytkownika



6

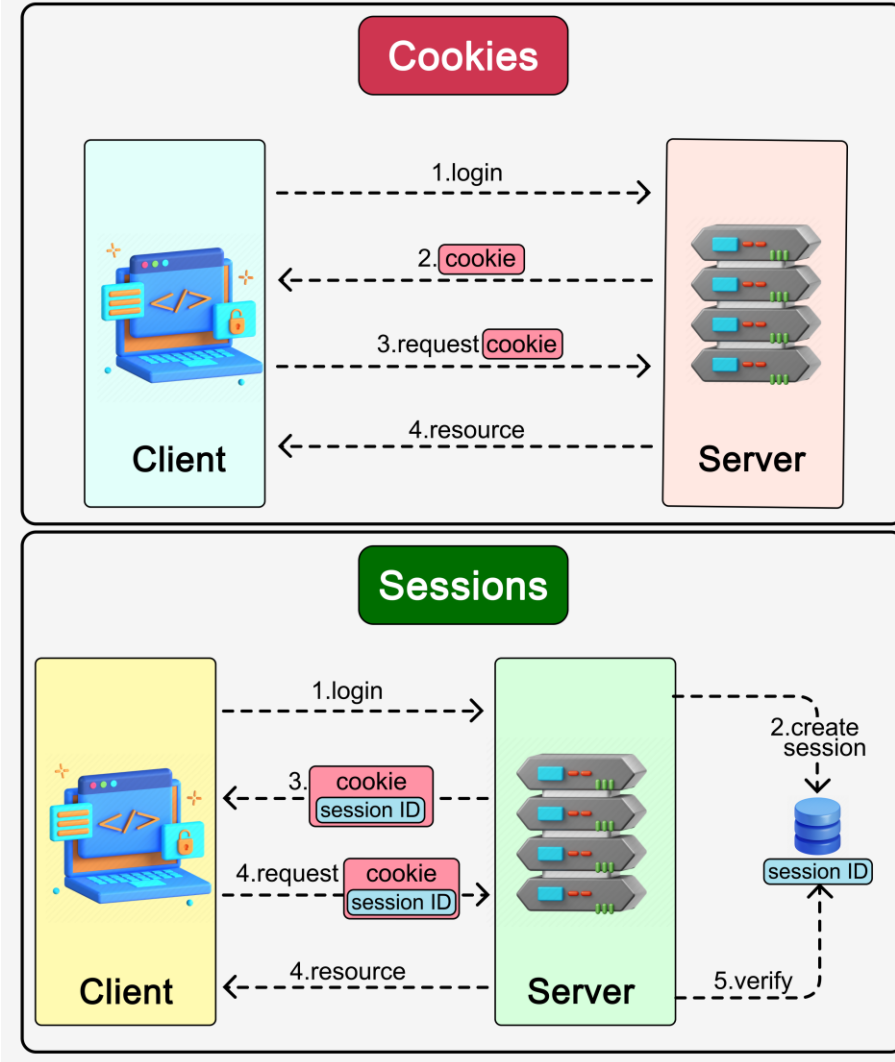
Czym jest sesja użytkownika?

Sesja użytkownika to mechanizm pozwalający serwerowi rozpoznawać kolejne żądania pochodzące od tego samego użytkownika. Ponieważ protokół HTTP jest bezstanowy, serwer nie przechowuje informacji o wcześniejszych żądaniach. Sesja umożliwia powiązanie wielu żądań w jeden logiczny ciąg, odpowiadający aktywności zalogowanego użytkownika.

Cookies (ciasteczka) to niewielkie dane zapisywane w przeglądarce użytkownika, które są automatycznie dołączane do kolejnych żądań HTTP wysyłanych do serwera. Najczęściej przechowują one identyfikator sesji, dzięki czemu serwer może rozpoznać użytkownika pomiędzy kolejnymi żądaniem. Cookies odgrywają kluczową rolę w utrzymywaniu sesji, personalizacji treści oraz realizacji mechanizmów uwierzytelniania w aplikacjach webowych.

Cookies vs Sessions

ByteByteGo



Rys. 3. Ciasteczka vs sesje [źródło: <https://bytebytego.com/guides/what-are-the-differences-between-cookies-and-sessions/>]

8

Rola sesji w aplikacjach internetowych

Dzięki sesji aplikacja może zapamiętać, że użytkownik jest zalogowany, jakie ma uprawnienia lub jakie operacje wykonał wcześniej. Sesja pozwala na personalizację interfejsu oraz kontrolę dostępu do zasobów. Bez mechanizmu sesji każda interakcja użytkownika musiałaby być traktowana jako całkowicie niezależna.

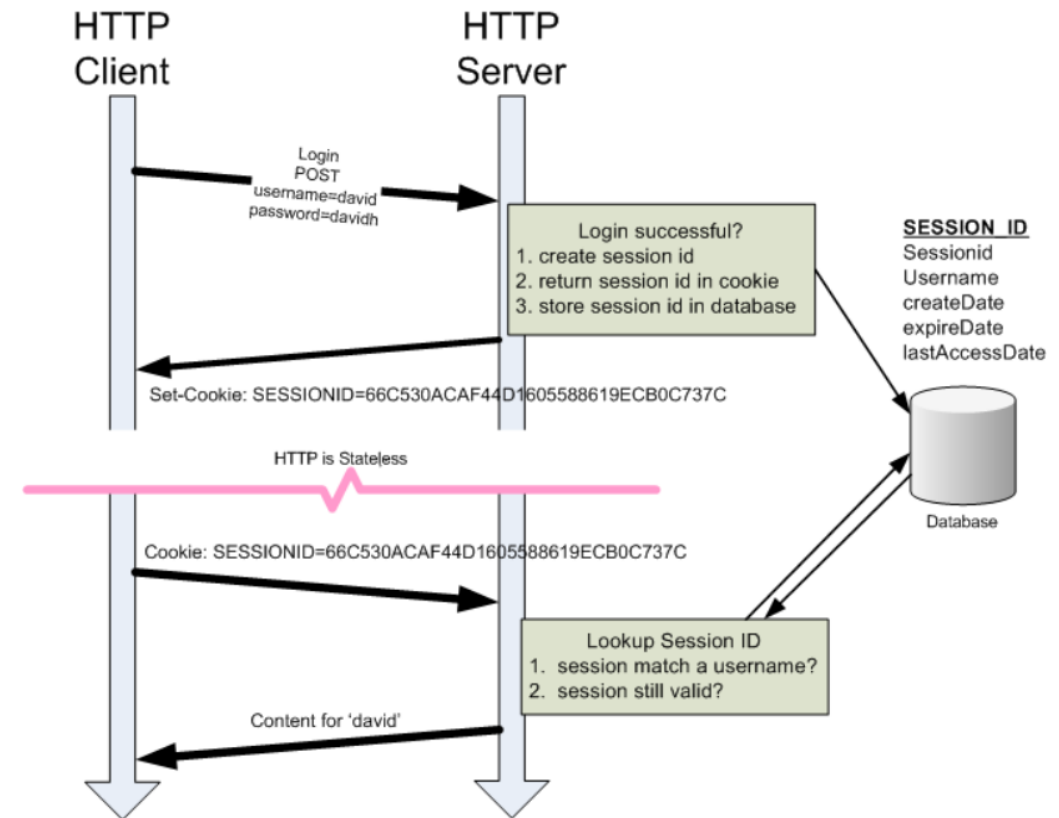


Rys. 4. Możliwości sesji [źródło:
<https://www.descope.com/learn/post/session-management>]

9

Identyfikator sesji

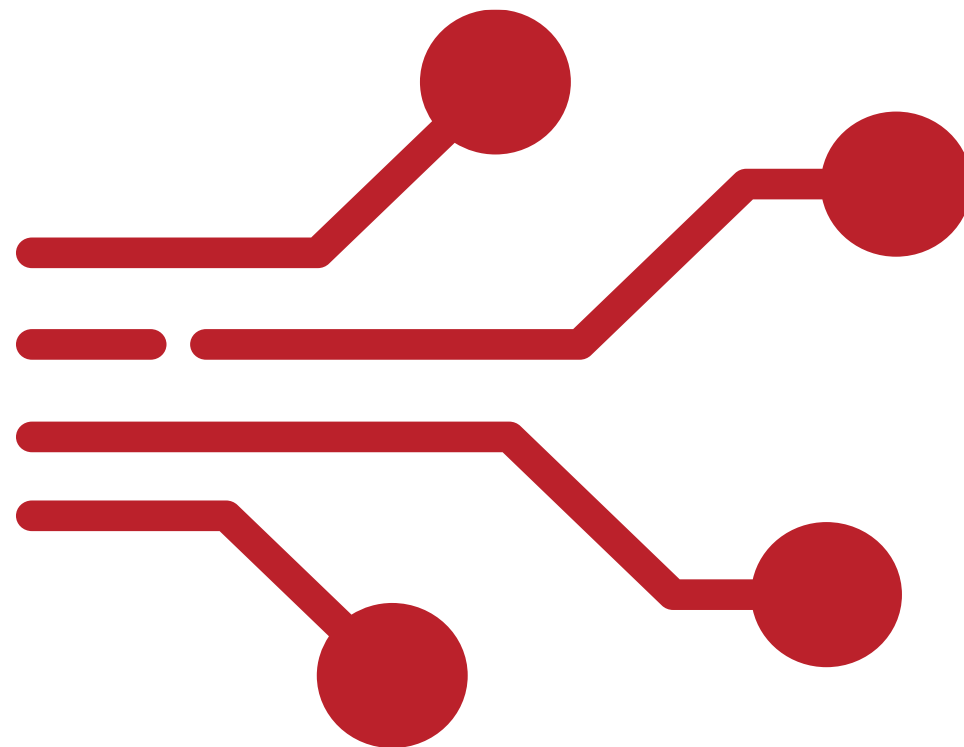
Każda sesja jest identyfikowana za pomocą unikalnego identyfikatora, zwanego identyfikatorem sesji. Jest on generowany przez serwer i przesyłany do klienta, najczęściej w postaci ciasteczka. Przy kolejnych żądaniach klient automatycznie przesyła identyfikator sesji, co pozwala serwerowi przypisać żądanie do konkretnego użytkownika.



Rys. 5. Identyfikacja sesji [źródło:
https://cscie12.dce.harvard.edu/lecture_notes/2018-fall/20181204/slide5.html]

10

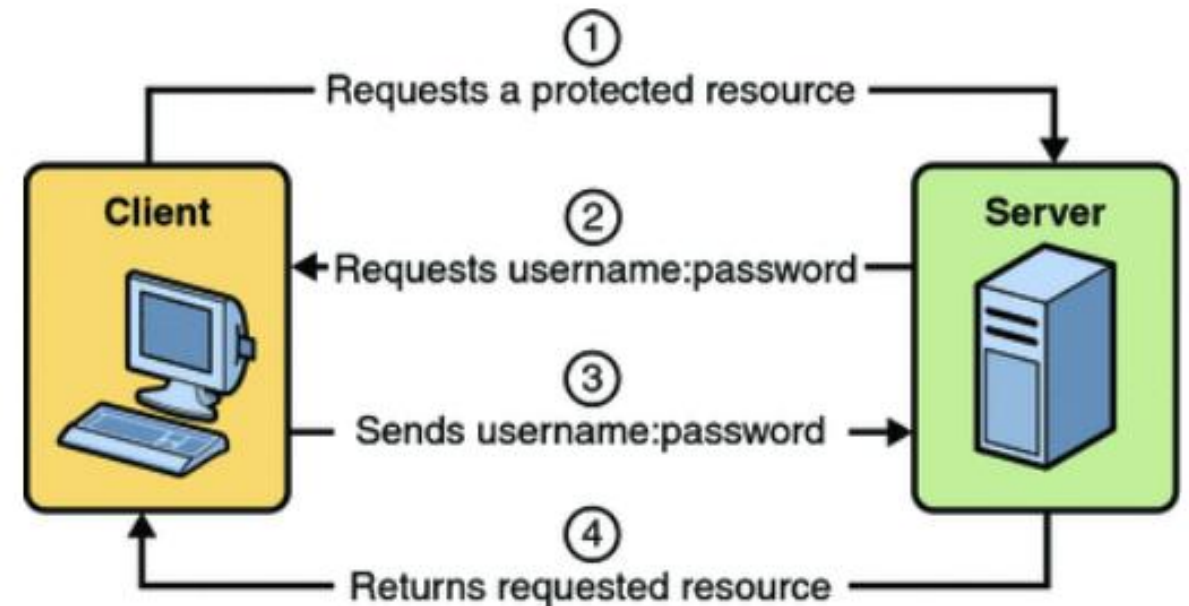
Mechanizmy uwierzytelniania



11

Uwierzytelnianie podstawowe (Basic Authentication)

Uwierzytelnianie podstawowe jest najprostszym mechanizmem uwierzytelniania w protokole HTTP. Polega na przesyłaniu danych uwierzytelniających w nagłówku żądania. Choć jest łatwe w implementacji, jego zastosowanie jest ograniczone ze względu na niskie bezpieczeństwo i brak mechanizmu utrzymywania stanu użytkownika.



Rys. 6. Uwierzytelnianie podstawowe [źródło: <https://docs.oracle.com/cd/E19879-01/819-3669/bncbo/index.html>]

12

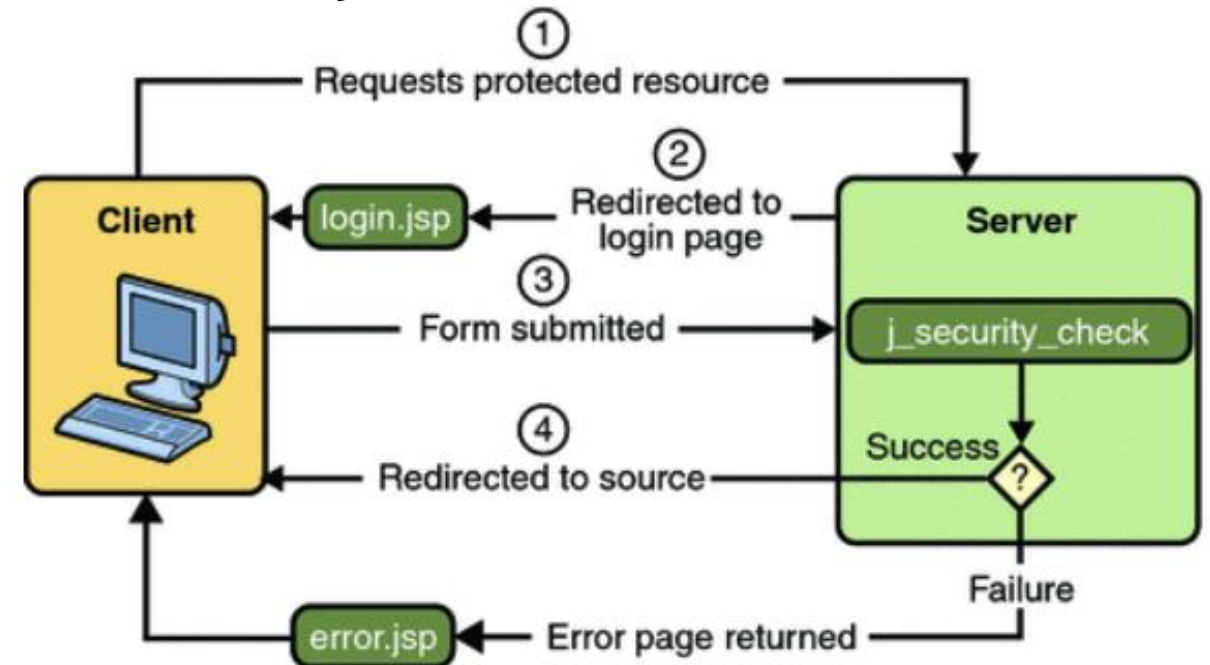
Ograniczenia uwierzytelniania podstawowego

Uwierzytelnianie podstawowe (basic authentication) wiąże się z istotnymi ograniczeniami, które sprawiają, że rzadko znajduje zastosowanie w nowoczesnych aplikacjach webowych. Dane uwierzytelniające użytkownika, czyli login i hasło, są przesyłane z każdym żądaniem HTTP, co oznacza brak mechanizmu sesji oraz brak wygody po stronie użytkownika. Każde zapytanie musi zawierać komplet informacji uwierzytelniających, niezależnie od wcześniejszych interakcji.

Nawet przy wykorzystaniu HTTPS rozwiązanie to pozostaje mało elastyczne i trudne do skalowania. Utrudnia wdrażanie dodatkowych mechanizmów bezpieczeństwa, takich jak zarządzanie sesją, wygaszanie logowania, role użytkowników czy uwierzytelnianie wieloskładnikowe. Z tego powodu basic authentication stosuje się głównie w prostych API, środowiskach testowych lub systemach wewnętrznych, a nie w pełnoprawnych aplikacjach webowych skierowanych do szerokiego grona użytkowników.

Uwierzytelnianie oparte na formularzu (Form-Based Authentication)

Form-Based Authentication to jeden z najczęściej stosowanych mechanizmów uwierzytelniania w aplikacjach internetowych. Polega on na logowaniu użytkownika za pomocą formularza, w którym podawane są dane uwierzytelniające, najczęściej login i hasło. Po przesłaniu formularza dane trafiają do backendu, gdzie są weryfikowane, a w przypadku poprawnego uwierzytelnienia tworzona jest sesja użytkownika.



4

Rys. 7. Uwierzytelnianie oparte na formularzu [źródło:
<https://docs.oracle.com/cd/E19879-01/819-3669/bncbq/index.html>]

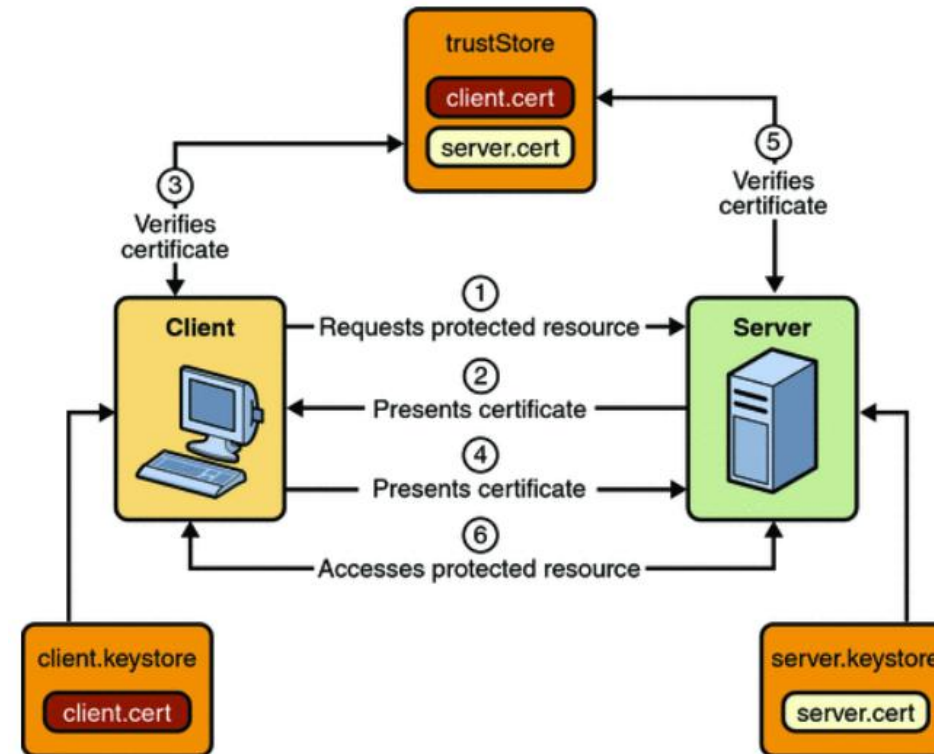
Uwierzytelnienie wzajemne (Mutual Authentication)

Mutual Authentication (uwierzytelnianie wzajemne) to mechanizm bezpieczeństwa, w którym obie strony komunikacji – klient i serwer – wzajemnie potwierdzają swoją tożsamość. W przeciwieństwie do klasycznego uwierzytelniania, gdzie tylko klient jest weryfikowany przez serwer, tutaj również klient ma pewność, że komunikuje się z zaufanym serwerem. Takie podejście jest szczególnie istotne w systemach o podwyższonych wymaganiach bezpieczeństwa.

15

Uwierzytelnienie wzajemne oparte na certyfikacie

W uwierzytelnianiu wzajemnym opartym na certyfikatach obie strony wykorzystują certyfikaty cyfrowe, najczęściej w ramach protokołu TLS. Serwer przedstawia klientowi swój certyfikat, a klient weryfikuje go na podstawie zaufanego urzędu certyfikacji. Następnie klient również prezentuje własny certyfikat, który jest sprawdzany przez serwer. Dopiero po poprawnej weryfikacji obu certyfikatów nawiązywane jest bezpieczne połączenie.

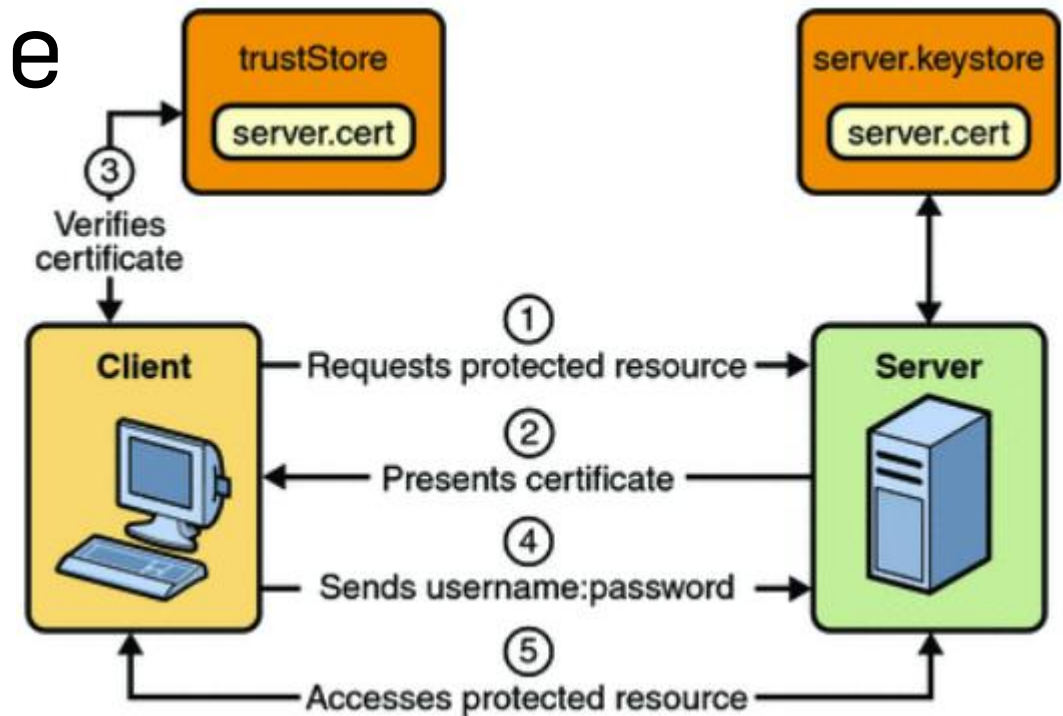


16

Rys. 8. Uwierzytelnienie wzajemne oparte na certyfikacie [źródło: <https://docs.oracle.com/cd/E19879-01/819-3669/bncbs/index.html>]

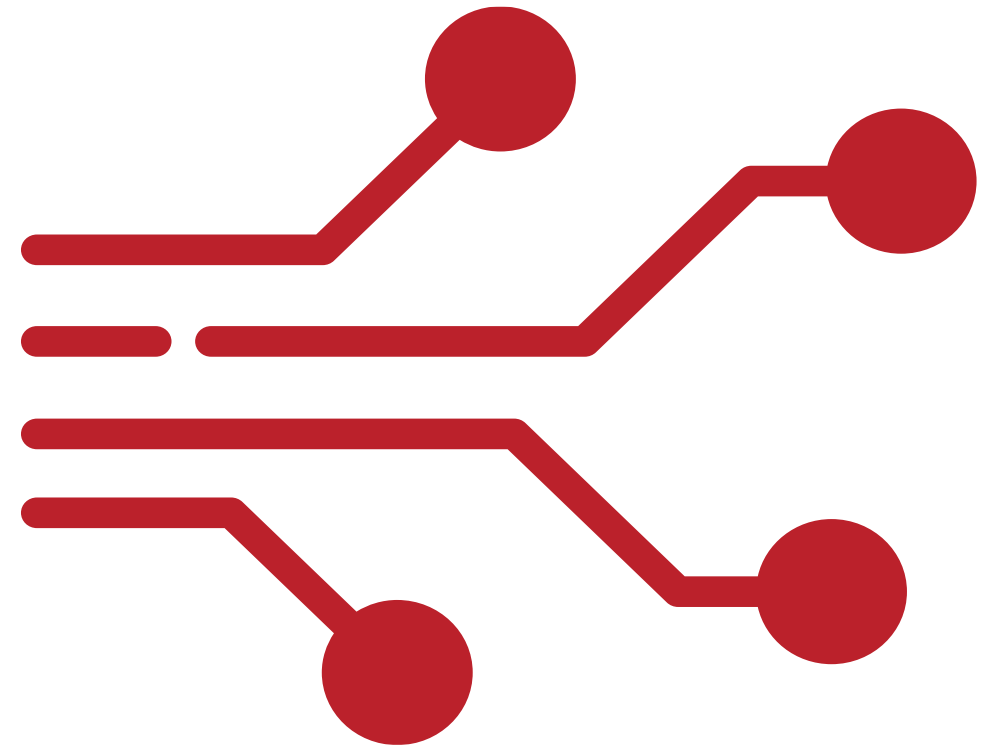
Uwierzytelnienie wzajemne oparte na nazwie użytkownika i haśle

Drugi wariant uwierzytelniania wzajemnego opiera się na loginie i haśle, jednak z rozszerzeniem klasycznego modelu. W tym podejściu klient uwierzytelnia się wobec serwera za pomocą danych logowania, a jednocześnie serwer musi udowodnić swoją tożsamość klientowi, na przykład poprzez bezpieczne połączenie HTTPS, tokeny lub dodatkowe mechanizmy kryptograficzne.



Rys. 9. Uwierzytelnienie wzajemne oparte na nazwie użytkownika i haśle [źródło: <https://docs.oracle.com/cd/E19879-01/819-3669/bncbs/index.html>] 17

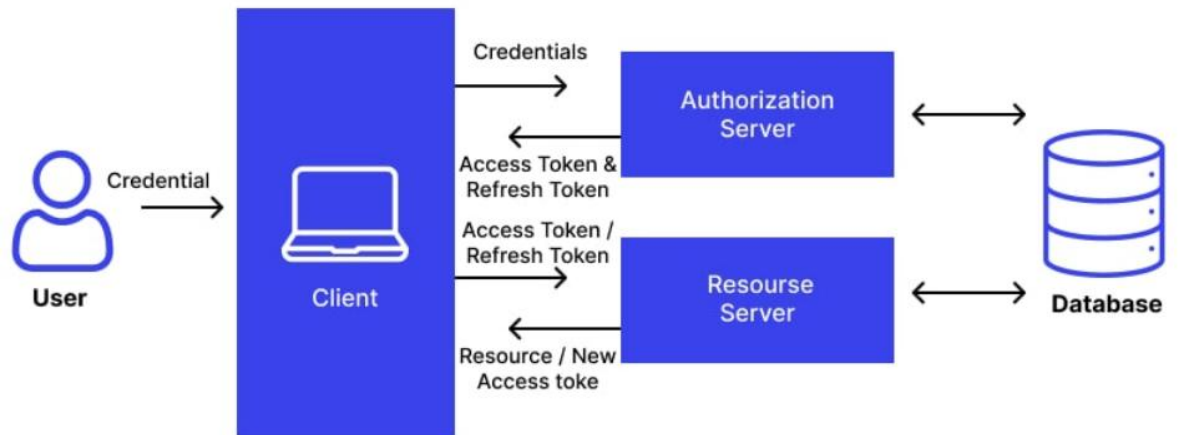
Tokeny i nowoczesne podejścia



18

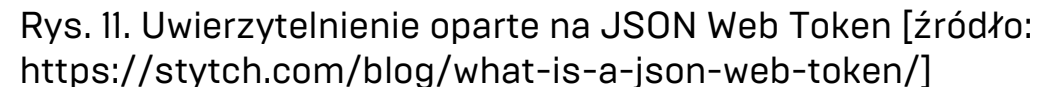
Uwierzytelnianie oparte na tokenach

W uwierzytelnianiu opartym na tokenach serwer po poprawnym zalogowaniu generuje token, który zawiera informacje o użytkowniku. Token jest przechowywany po stronie klienta i dołączany do kolejnych żądań. W przeciwieństwie do sesji serwer nie musi przechowywać stanu użytkownika, co zwiększa skalowalność systemu.



Rys. 10. Uwierzytelnienie oparte na tokenach [źródło: <https://www.wallarm.com/what/token-based-authentication>]

Jednym z najpopularniejszych standardów tokenów są JSON Web Tokens. JWT zawierają zakodowane informacje o użytkowniku oraz podpis kryptograficzny, który pozwala zweryfikować ich autentyczność. Dzięki temu serwer może sprawdzić poprawność tokenu bez konieczności przechowywania sesji.



Sesje a tokeny – porównanie

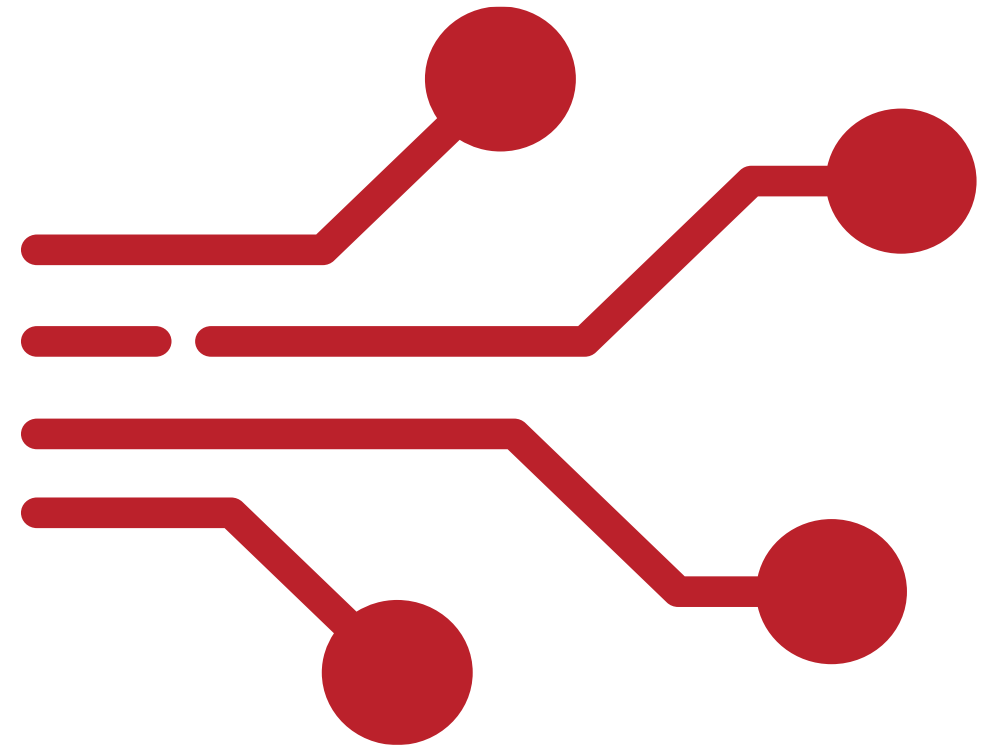
Sesje i tokeny realizują ten sam cel – umożliwiają rozpoznawanie użytkownika i kontrolę dostępu – jednak robią to w odmienny sposób. W modelu sesyjnym stan użytkownika jest przechowywany po stronie serwera, a klient przesyła jedynie identyfikator sesji. Takie podejście jest wygodne w klasycznych aplikacjach webowych, ale może utrudniać skalowanie systemu, ponieważ serwer musi zarządzać wieloma aktywnymi sesjami.

W przypadku uwierzytelniania tokenowego stan użytkownika przenoszony jest na stronę klienta w postaci tokenu, który jest dołączany do kolejnych żądań. Serwer nie musi przechowywać informacji o sesji, co zwiększa skalowalność i dobrze sprawdza się w aplikacjach opartych na API, SPA oraz systemach rozproszonych. Wybór pomiędzy sesjami a tokenami zależy od charakteru aplikacji, jej architektury, skali oraz wymagań dotyczących bezpieczeństwa i wydajności.

Feature	Session Authentication	Token Authentication
Storage Location	Session data stored on server; session ID in browser cookie	Token stored on client-side (e.g., localStorage or sessionStorage)
State	Stateful; server keeps track of sessions	Stateless; all info stored in the token
Scalability	Less scalable; server must manage session data	Highly scalable; server doesn't store session info
Login Handling	Handled via session creation and cookies	Token issued and stored by client after login
Logout Implementation	Easy to invalidate by clearing session on server	Requires custom logic to revoke tokens
Size	Small; only session ID sent in requests	Larger; token may contain user roles and permissions
Cross-Domain Support	Limited; cookies usually bound to one domain	Flexible; tokens can be sent to multiple domains
Security Risks	More vulnerable to CSRF attacks	More vulnerable to XSS attacks
Automatic Handling	Cookies sent automatically by the browser	Tokens must be manually added to HTTP headers
Best For	Traditional web apps (multi-page)	Modern apps, SPAs, mobile, and microservices

Rys. 12. Uwierzytelnienie – sesja vs tokeny [źródło: <https://www.theknowledgeacademy.com/blog/session-vs-token-authentication/>]

Zagrożenia i dobre praktyki



23

Zagrożenia związane z sesją

Nieprawidłowe zarządzanie sesją stanowi jedno z poważniejszych zagrożeń bezpieczeństwa w aplikacjach webowych. Jeśli identyfikator sesji zostanie ujawniony, atakujący może przejąć sesję użytkownika i podszyć się pod niego bez konieczności znajomości danych logowania. Tego typu ataki, określane jako przejęcie sesji, mogą prowadzić do nieautoryzowanego dostępu do kont, danych osobowych lub funkcji administracyjnych.

Szczególnie niebezpieczne są sytuacje, w których identyfikator sesji jest przesyłany w niezabezpieczony sposób, zapisywany w nieodpowiednich miejscach lub nie jest odpowiednio unieważniany po wylogowaniu. Brak wygasania sesji, podatność na ataki typu session fixation czy niewłaściwe ustawienia ciasteczek zwiększają ryzyko naruszenia bezpieczeństwa. Dlatego bezpieczne zarządzanie sesją, w tym stosowanie HTTPS, odpowiednich flag ciasteczek oraz mechanizmów wygaszania sesji, jest kluczowe dla ochrony użytkowników i aplikacji.

Bezpieczne przechowywanie danych uwierzytelniających

Bezpieczne przechowywanie danych uwierzytelniających jest jednym z podstawowych obowiązków backendu aplikacji webowej. Dane takie jak hasła nigdy nie powinny być zapisywane w postaci jawnej, ponieważ w przypadku naruszenia bezpieczeństwa mogłyby zostać natychmiast wykorzystane przez osoby nieuprawnione. Zamiast tego serwer powinien przechowywać jedynie ich bezpieczne skróty, generowane przy użyciu odpowiednich algorytmów kryptograficznych oraz mechanizmów takich jak sól, które utrudniają ataki słownikowe i brute force.

Równie istotne jest zabezpieczenie procesu przesyłania danych uwierzytelniających pomiędzy klientem a serwerem. Stosowanie bezpiecznych połączeń HTTPS zapewnia szyfrowanie transmisji i chroni dane przed podsłuchem oraz modyfikacją. Połączenie właściwych algorytmów kryptograficznych z bezpieczną komunikacją sieciową znacząco zwiększa poziom ochrony kont użytkowników i całej aplikacji.

25

Zarządzanie czasem życia sesji

Sesja użytkownika powinna mieć określony czas życia. Automatyczne wygaszanie sesji po okresie nieaktywności zmniejsza ryzyko nieautoryzowanego dostępu. Jest to szczególnie ważne w aplikacjach przetwarzających dane wrażliwe.

You will be logged out soon



For your security, we log you out automatically when you have not been active for a certain period of time.

Current time setting: 5 minutes

[Change time setting](#)

[Log out now](#)

[Continue session](#)

Rys. 13. Informacja o wylogowaniu z powodu braku aktywności
[źródło: <https://www.descope.com/learn/post/session-timeout-best-practices>] 26

Dobre praktyki uwierzytelniania

Stosowanie dobrych praktyk uwierzytelniania ma kluczowe znaczenie dla bezpieczeństwa aplikacji internetowych. Jednym z podstawowych elementów jest wykorzystywanie silnych mechanizmów uwierzytelniania, takich jak odpowiednia polityka haseł, uwierzytelnianie wieloskładnikowe czy bezpieczne zarządzanie sesją i tokenami. Rozwiązania te znacząco utrudniają przejęcie kont użytkowników.

Równie istotne jest ograniczanie liczby nieudanych prób logowania, co pozwala chronić system przed atakami typu brute force. Monitorowanie podejrzanych działań, takich jak nietypowe próby logowania czy nagłe zmiany zachowania użytkownika, umożliwia szybką reakcję na potencjalne zagrożenia. Odpowiednia konfiguracja i konsekwentne stosowanie mechanizmów uwierzytelniania znacząco zwiększają poziom bezpieczeństwa całej aplikacji.

Best Practices for Secure Login Implementation

Ensuring user authentication is both secure and user-friendly

1 Hide Passwords & Offer Visibility Toggle

Always mask passwords by default and provide a "Show/Hide Password" option for better usability.

2 Handle Errors Securely

Use proper error codes like 401 Unauthorized or 429 Too Many Requests from the server to classify failures. On the UI, show generic error messages like "invalid username or password"—no hints for attackers!



3 Restrict Login Attempts

Limit consecutive failed attempts and implement time-based lockouts. Use exponential backoff (e.g., 1 min → 5 min → 1 hour) to thwart brute force attacks.

4 CAPTCHA as a Defense-in-Depth Measure

Add CAPTCHA after multiple failures or unusual activity. While not foolproof, it significantly raises the cost of brute-force or DDOS attacks.

5 Monitor & Log Failures

Track failed login attempts, lockouts, and suspicious patterns. Log details like timestamps, IPs, and user-agents (securely!) for analysis and incident response.

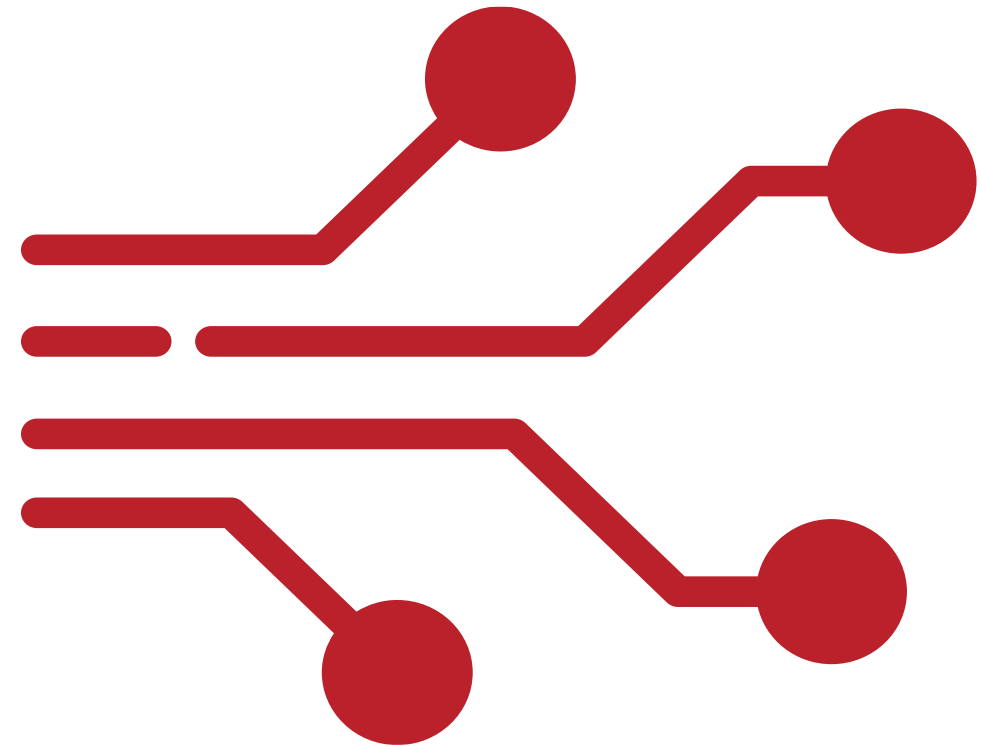
6 User Experience Meets Security

Balancing usability and security is key. Features like password visibility toggles and clear yet non-revealing error messages

Rys. 14. Dobre praktyki implementacji logowania [źródło: <https://javascript.plainenglish.io/6-best-practices-for-secure-login-implementation-in-2025-738d6b5cble6>]

28

Podsumowanie



29

Zarządzanie sesją i uwierzytelnianie stanowią fundament bezpiecznych aplikacji internetowych, ponieważ odpowiadają za jednoznaczną identyfikację użytkowników oraz kontrolę dostępu do zasobów systemu. Dzięki nim możliwe jest utrzymanie stanu zalogowania pomiędzy kolejnymi żądaniami oraz egzekwowanie uprawnień zgodnie z rolą użytkownika.

Właściwy dobór i poprawna implementacja mechanizmów sesyjnych oraz tokenowych mają bezpośredni wpływ na bezpieczeństwo, wydajność i skalowalność aplikacji. Świadome stosowanie dobrych praktyk w tym obszarze pozwala tworzyć systemy, które są nie tylko funkcjonalne i wygodne w użyciu, ale również odporne na nadużycia i zagrożenia bezpieczeństwa.

Literatura:

- [1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.
- [2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.
- [3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.
- [4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.
- [5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Dziękuję za uwagę

Wykład nr 13

Temat: Wdrażanie i optymalizacja aplikacji internetowych

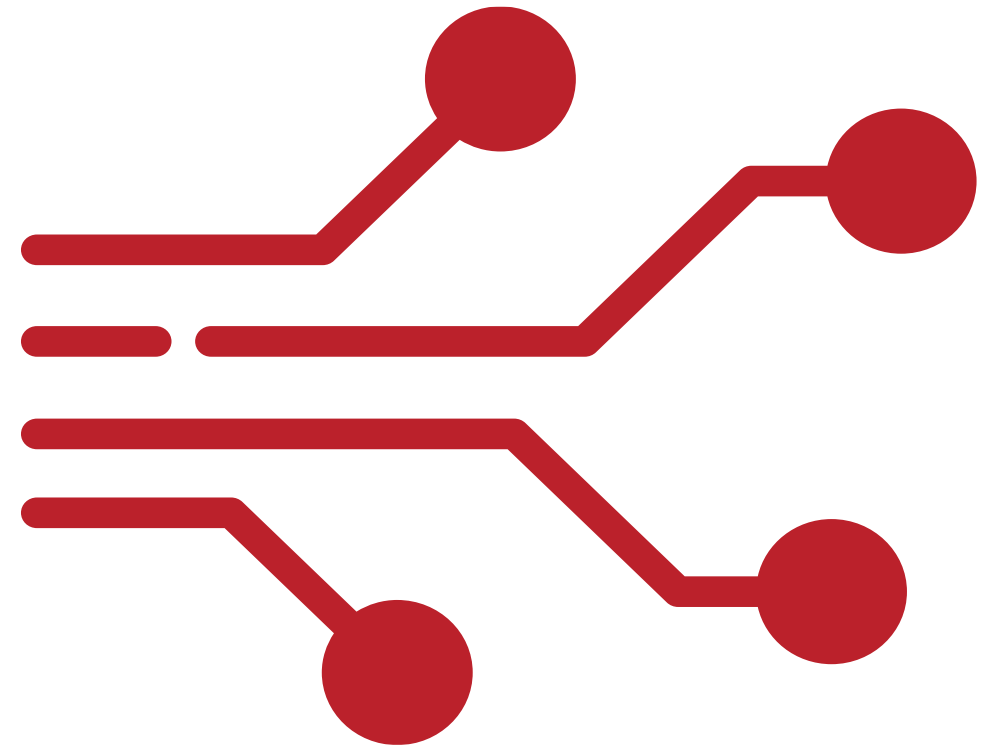
dr hab. inż. Michał Wydra, prof. Uczelni
Katedra Informatyki Stosowanej

Plan prezentacji

- Wydajność aplikacji internetowych
- Proces ładowania strony internetowej
- Metryki wydajności
- Narzędzia do pomiaru wydajności
- Techniki optymalizacji aplikacji



Wydajność aplikacji internetowych



Czym jest wydajność aplikacji webowej?

Wydajność aplikacji webowej określa, jak szybko i sprawnie system reaguje na działania użytkownika oraz jak efektywnie wykorzystuje dostępne zasoby. Obejmuje ona takie aspekty jak czas ładowania strony, szybkość odpowiedzi na interakcje, płynność działania interfejsu oraz stabilność aplikacji przy rosnącej liczbie użytkowników. Im krótszy czas reakcji i bardziej przewidywalne działanie systemu, tym lepsze wrażenia użytkownika.

Z punktu widzenia użytkownika wydajność jest jednym z kluczowych czynników wpływających na ocenę jakości aplikacji. Nawet funkcjonalnie poprawna aplikacja może zostać odebrana negatywnie, jeśli działa wolno lub niestabilnie. Dlatego wydajność stanowi istotny element projektowania i rozwoju aplikacji webowych, mający bezpośredni wpływ na satysfakcję użytkowników oraz sukces całego systemu.

Dlaczego wydajność jest tak ważna?

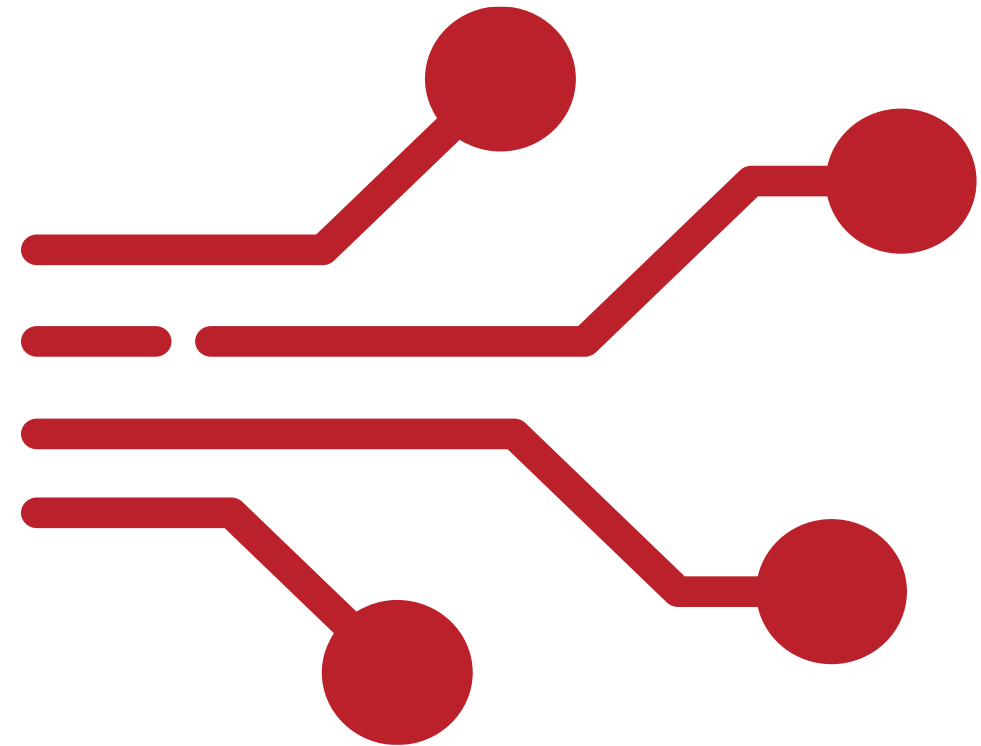
Długi czas ładowania strony negatywnie wpływa na doświadczenie użytkownika i zwiększa ryzyko opuszczenia witryny. Wydajność ma również znaczenie biznesowe, ponieważ wpływa na współczynnik konwersji oraz widoczność strony w wynikach wyszukiwania. Szybko działające aplikacje są lepiej oceniane zarówno przez użytkowników, jak i algorytmy wyszukiwarek.



Rys. 1. Podstawowe wskaźniki sieci [źródło:
<https://seomator.com/blog/load-speed-impacts-on-seo>]

5

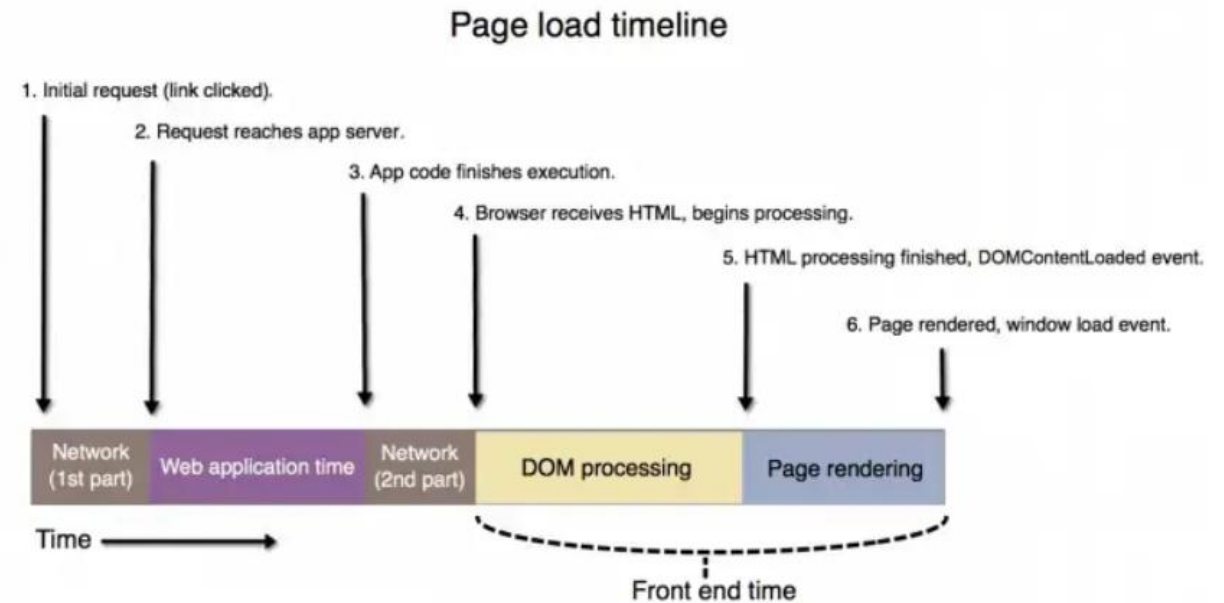
Proces ładowania strony internetowej



6

Etapy ładowania strony w przeglądarce

Proces ładowania strony internetowej składa się z wielu etapów, począwszy od wysłania żądania HTTP, poprzez nawiązanie połączenia z serwerem, aż po pobranie i przetworzenie zasobów. Każdy z tych etapów może mieć wpływ na końcowy czas ładowania strony. Zrozumienie tego procesu pozwala lepiej identyfikować miejsca wymagające optymalizacji.



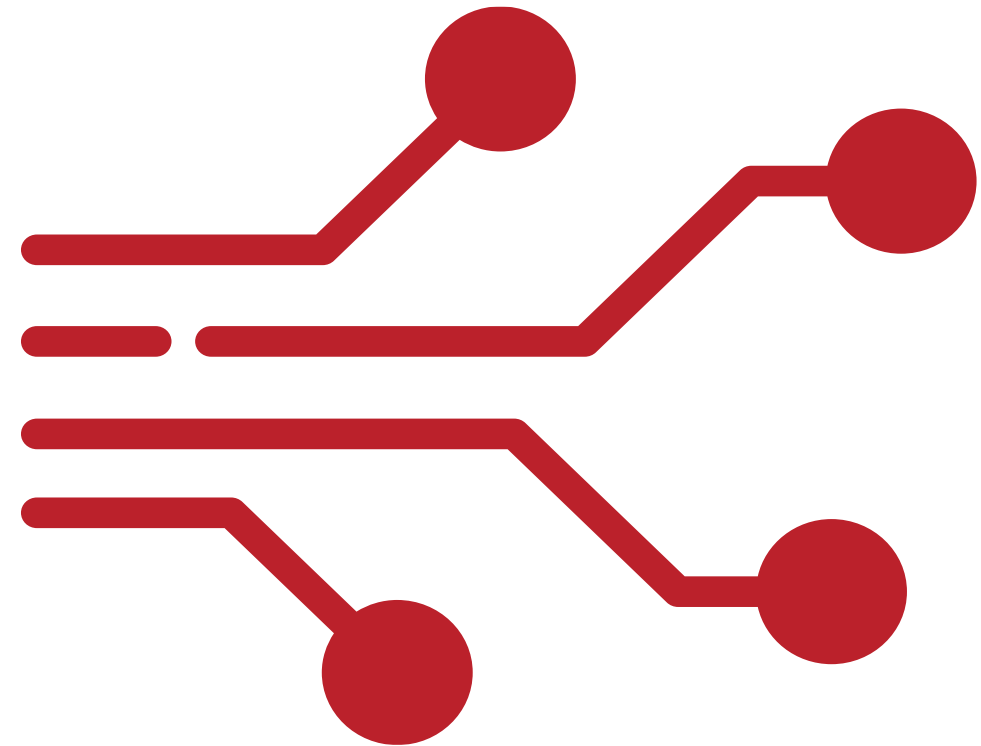
Rys. 2. Etapy ładowania strony [źródło: <https://docs.newrelic.com/docs/browser/new-relic-browser/page-load-timing-resources/page-load-timing-process/>]

Rola serwera w czasie odpowiedzi

Serwer odgrywa bardzo ważną rolę w czasie odpowiedzi aplikacji webowej, ponieważ to właśnie on odpowiada za przetworzenie żądania i wygenerowanie odpowiedzi dla klienta. Jeśli serwer potrzebuje dużo czasu na wykonanie operacji, takich jak dostęp do bazy danych, walidacja danych czy realizacja logiki biznesowej, użytkownik odczuwa to jako opóźnienie w działaniu aplikacji. Nawet najlepiej zoptymalizowany frontend nie zrekompensuje wolnej odpowiedzi po stronie serwera.

Wydajność serwera zależy od wielu czynników, w tym jego konfiguracji, dostępnych zasobów sprzętowych oraz aktualnego obciążenia. Istotny jest również sposób przetwarzania danych, jakość zapytań do bazy danych oraz architektura aplikacji. Odpowiednia optymalizacja backendu, skalowanie infrastruktury i efektywne zarządzanie zasobami pozwalają skrócić czas odpowiedzi i zapewnić płynne działanie aplikacji nawet przy dużej liczbie użytkowników.

Metryki wydajności

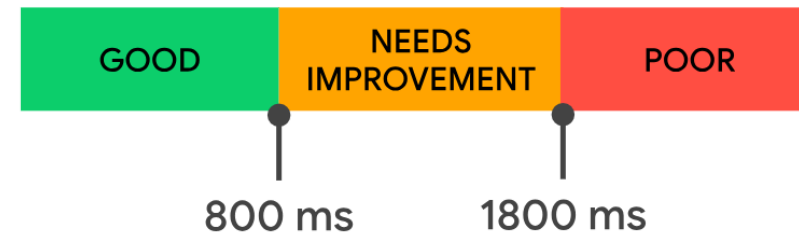


Czas do pierwszego bajtu (TTFB)

TTFB, czyli Time To First Byte, określa czas od wysłania żądania do momentu otrzymania pierwszego bajtu odpowiedzi z serwera. Jest to istotny wskaźnik jakości komunikacji z serwerem oraz jego wydajności. Wysoka wartość TTFB może świadczyć o problemach po stronie backendu lub infrastruktury sieciowej.

TTFB

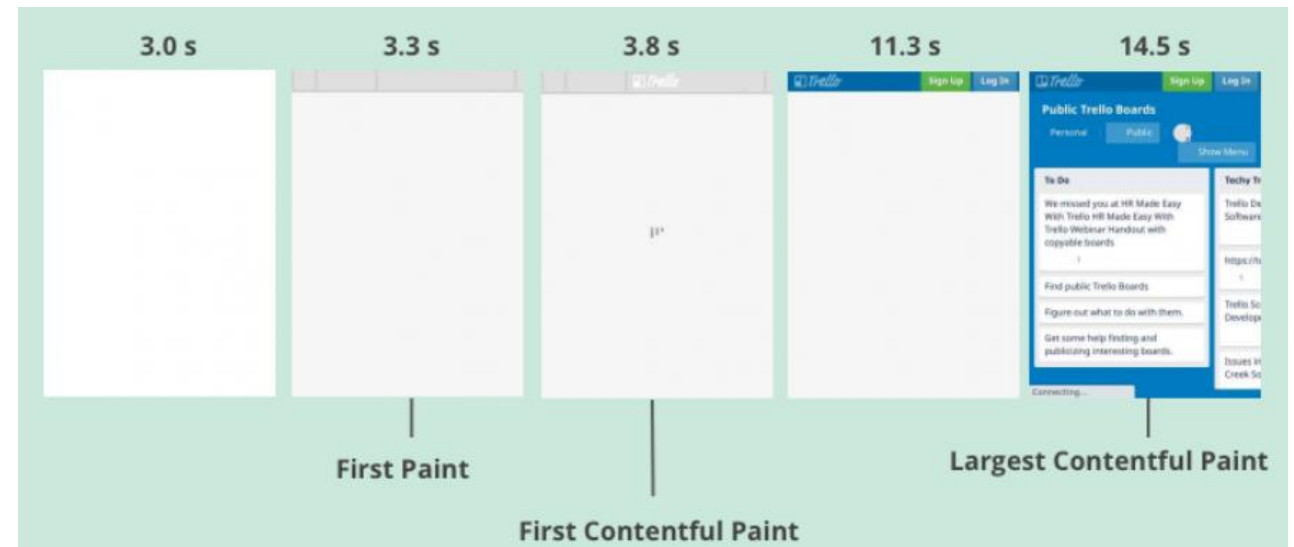
Time To First Byte



Rys. 3. Time To First Byte [źródło:
<https://web.dev/articles/optimize-ttfb>]

Pierwsze renderowanie i widoczna treść

Pierwsze renderowanie strony ma ogromne znaczenie dla tego, jak użytkownik postrzega szybkość działania aplikacji. Metryki takie jak First Paint (FP) oraz First Contentful Paint (FCP) opisują moment, w którym w przeglądarce pojawia się pierwsza widoczna zawartość, na przykład tło strony, tekst lub obraz. Nawet minimalna zmiana wizualna sygnalizuje użytkownikowi, że strona zaczęła się ładować.



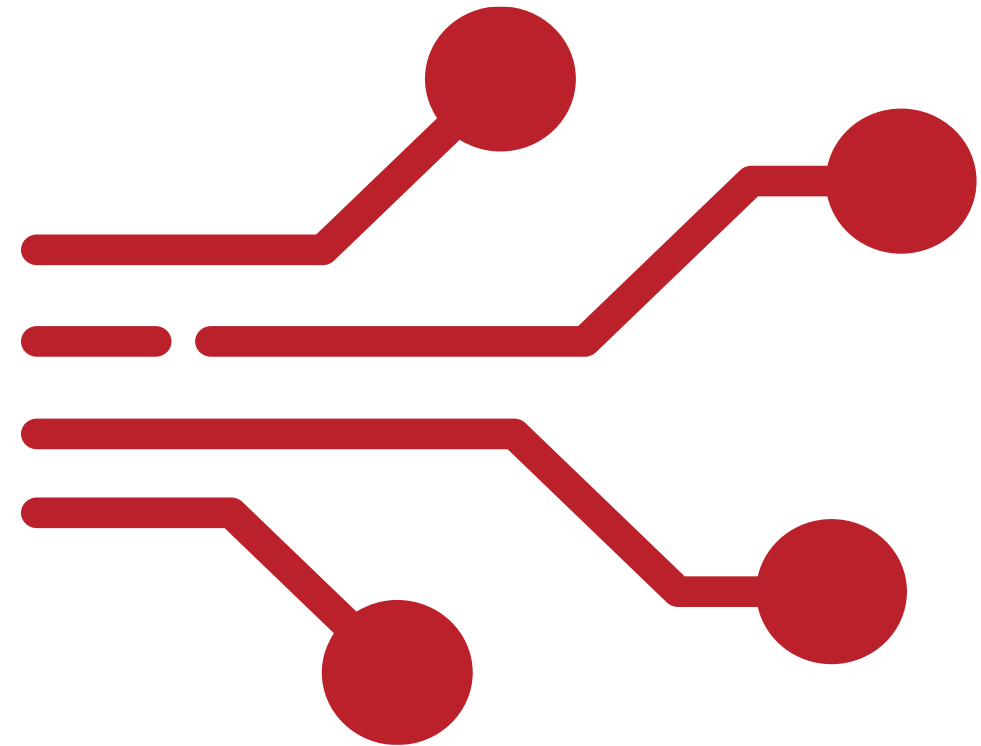
Rys. 4. Renderowanie strony [źródło: <https://www.debugbear.com/docs/metrics/first-contentful-paint>]

Interaktywność strony

Time to Interactive określa moment, w którym strona jest w pełni załadowana i gotowa do interakcji. Jest to szczególnie ważne w przypadku aplikacji internetowych, które zawierają dynamiczne elementy i logikę po stronie klienta. Strona może być widoczna, ale dopiero po osiągnięciu tego momentu użytkownik może z niej swobodnie korzystać.

W przypadku nowoczesnych aplikacji internetowych, które intensywnie wykorzystują JavaScript i dynamiczne komponenty, osiągnięcie krótkiego czasu do pełnej interaktywności jest szczególnie istotne. Strona może być widoczna wcześniej, jednak dopóki nie osiągnie stanu pełnej interaktywności, użytkownik może doświadczać opóźnień lub braku reakcji interfejsu. Optymalizacja TTI pozwala poprawić płynność działania aplikacji i znacząco zwiększa komfort jej użytkowania.

Narzędzia do pomiaru wydajności



13

PageSpeed Insights

PageSpeed Insights to narzędzie umożliwiające analizę wydajności strony internetowej oraz ocenę jej zgodności z wytycznymi Core Web Vitals. Narzędzie to generuje raporty zarówno dla wersji mobilnej, jak i desktopowej, a także sugeruje możliwe usprawnienia.



Rys. 5. Logo PageSpeed Insights
[źródło: <https://pagespeed.web.dev/>]

Chrome DevTools

Chrome DevTools oferuje rozbudowane możliwości analizy wydajności aplikacji. Zakładki takie jak Network czy Performance pozwalają śledzić czas ładowania zasobów, kolejność wykonywania operacji oraz potencjalne wąskie gardła. Jest to jedno z podstawowych narzędzi pracy web developera.



Rys. 6. Logo Chrome for developers [źródło:
<https://developer.chrome.com/docs/devtools?hl=pl>]

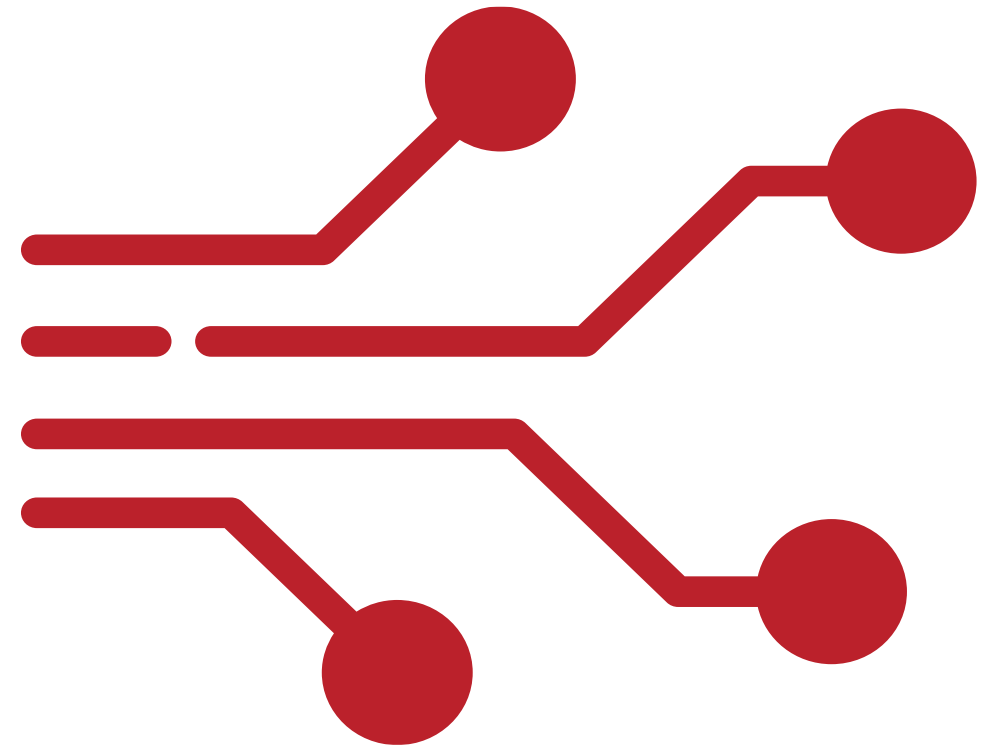
Lighthouse

Lighthouse to narzędzie audytowe, które poza wydajnością analizuje również dostępność, SEO oraz dobre praktyki. Pozwala ono kompleksowo ocenić jakość aplikacji internetowej i wskazuje obszary wymagające poprawy jeszcze przed wdrożeniem produkcyjnym.



Rys. 7. Logo Lighthouse [źródło:
<https://chromewebstore.google.com/detail/lighthouse/blipmdconlkpinefehnmjammfjpmpbjk?hl=pl>]

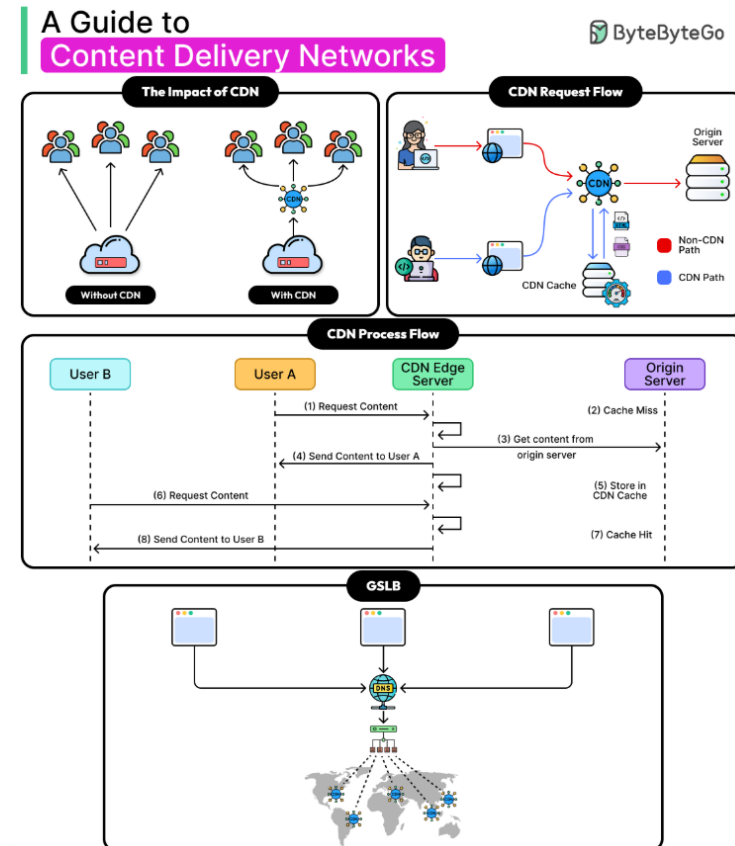
Techniki optymalizacji aplikacji



17

Optymalizacja połączeń i wykorzystanie CDN

Jednym ze sposobów poprawy wydajności jest skrócenie czasu przesyłania danych. Content Delivery Networks umożliwiają dostarczanie zasobów z serwerów znajdujących się bliżej użytkownika. Dzięki temu zmniejsza się opóźnienie wynikające z fizycznej odległości pomiędzy klientem a serwerem.



Rys. 8. Content Delivery Networks [źródło: <https://blog.bytebytego.com/p/a-detailed-guide-to-content-delivery>]

Cache po stronie serwera

Cache po stronie serwera polega na tymczasowym przechowywaniu wyników często wykonywanych operacji, takich jak zapytania do bazy danych, obliczenia biznesowe czy generowane odpowiedzi API. Zamiast każdorazowo przetwarzać to samo żądanie od początku, serwer może skorzystać z wcześniej zapisanych danych w pamięci podręcznej, co znacząco przyspiesza obsługę kolejnych zapytań.

Stosowanie cache pozwala istotnie skrócić czas odpowiedzi aplikacji oraz zmniejszyć obciążenie bazy danych i zasobów serwera. Ma to szczególne znaczenie w systemach obsługujących dużą liczbę użytkowników lub intensywnie korzystających z tych samych danych. Odpowiednio zaprojektowany mechanizm cache'owania poprawia wydajność i stabilność aplikacji, jednocześnie zwiększając jej skalowalność.

Cache po stronie klienta

Cache po stronie klienta polega na przechowywaniu zasobów aplikacji bezpośrednio w przeglądarce użytkownika. Przeglądarki internetowe potrafią zapisywać pliki takie jak obrazy, arkusze CSS czy pliki JavaScript, aby przy kolejnych wizytach na stronie nie było konieczności ponownego ich pobierania z serwera. Dzięki temu czas ładowania strony ulega znacznemu skróceniu, a ilość przesyłanych danych zostaje ograniczona.

Proces cache'owania po stronie klienta jest kontrolowany za pomocą odpowiednich nagłówków HTTP, takich jak Cache-Control czy Expires. Poprawna konfiguracja tych nagłówków pozwala określić, jak długo zasoby mogą być przechowywane lokalnie oraz kiedy powinny zostać odświeżone. Odpowiednie wykorzystanie cache po stronie klienta poprawia wydajność aplikacji, zmniejsza obciążenie serwera i znacząco podnosi komfort użytkowania.

Optymalizacja zasobów statycznych

Zasoby statyczne, takie jak obrazy, arkusze stylów CSS czy pliki JavaScript, mają istotny wpływ na czas ładowania strony internetowej. Duże lub nieoptymalnie przygotowane pliki zwiększają ilość danych przesyłanych do przeglądarki, co wydłuża czas oczekiwania użytkownika. Dlatego optymalizacja tych zasobów jest jednym z kluczowych elementów poprawy wydajności aplikacji webowej.

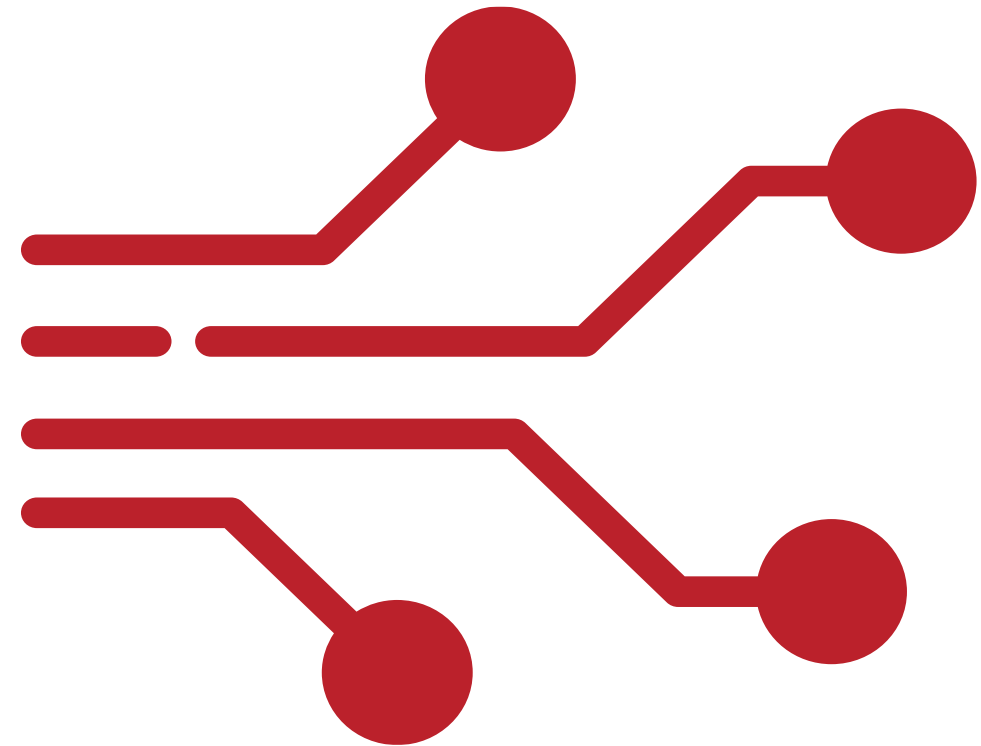
Optymalizacja polega między innymi na wyborze odpowiednich formatów plików, zmniejszaniu ich rozmiaru oraz stosowaniu kompresji danych. Przykładowo, obrazy mogą być zapisywane w nowoczesnych formatach, a pliki CSS i JavaScript mogą być minimalizowane i łączone. Dobrze zoptymalizowane zasoby statyczne skracają czas ładowania strony, poprawiają płynność działania aplikacji i znacząco zwiększają komfort korzystania z niej.

Lazy loading i preloading

Lazy loading i preloading to techniki, które pozwalają świadomie zarządzać momentem pobierania zasobów w aplikacji webowej. Lazy loading polega na opóźnieniu ładowania elementów do chwili, gdy faktycznie są one potrzebne, na przykład gdy użytkownik przewija stronę i dany fragment interfejsu staje się widoczny. Dzięki temu początkowy czas ładowania strony jest krótszy, a przeglądarka nie pobiera zbędnych danych.

Preloading działa w odwrotny sposób – umożliwia wcześniejsze pobranie zasobów, które z dużym prawdopodobieństwem będą potrzebne w kolejnym kroku, na przykład po kliknięciu przycisku lub przejściu na następną podstronę. Obie techniki pozwalają lepiej rozłożyć obciążenie sieci w czasie, poprawiają postrzeganą wydajność aplikacji i zwiększają płynność interakcji użytkownika z systemem.

Podsumowanie



23

Wdrażanie i optymalizacja aplikacji internetowych to proces ciągły, który wymaga analizy działania systemu, testowania różnych rozwiązań oraz świadomego podejmowania decyzji technicznych. Kluczowe znaczenie ma regularny pomiar wydajności przy użyciu odpowiednich metryk i narzędzi, ponieważ pozwala on identyfikować wąskie gardła, wykrywać problemy oraz oceniać realny wpływ wprowadzanych usprawnień.

Zastosowanie sprawdzonych technik, takich jak cache po stronie serwera i klienta, wykorzystanie sieci CDN, optymalizacja zasobów statycznych czy mechanizmy lazy loading i preloading, umożliwia znaczną poprawę szybkości i stabilności działania aplikacji. Dzięki temu możliwe jest tworzenie aplikacji webowych, które nie tylko spełniają wymagania funkcjonalne, ale są również szybkie, skalowalne i przyjazne dla użytkownika, co bezpośrednio przekłada się na ich pozytywny odbiór i sukces.

Literatura:

- [1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.
- [2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.
- [3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.
- [4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.
- [5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Dziękuję za uwagę

Wykład nr 14

Temat: Wprowadzenie do bezpieczeństwa aplikacji internetowych

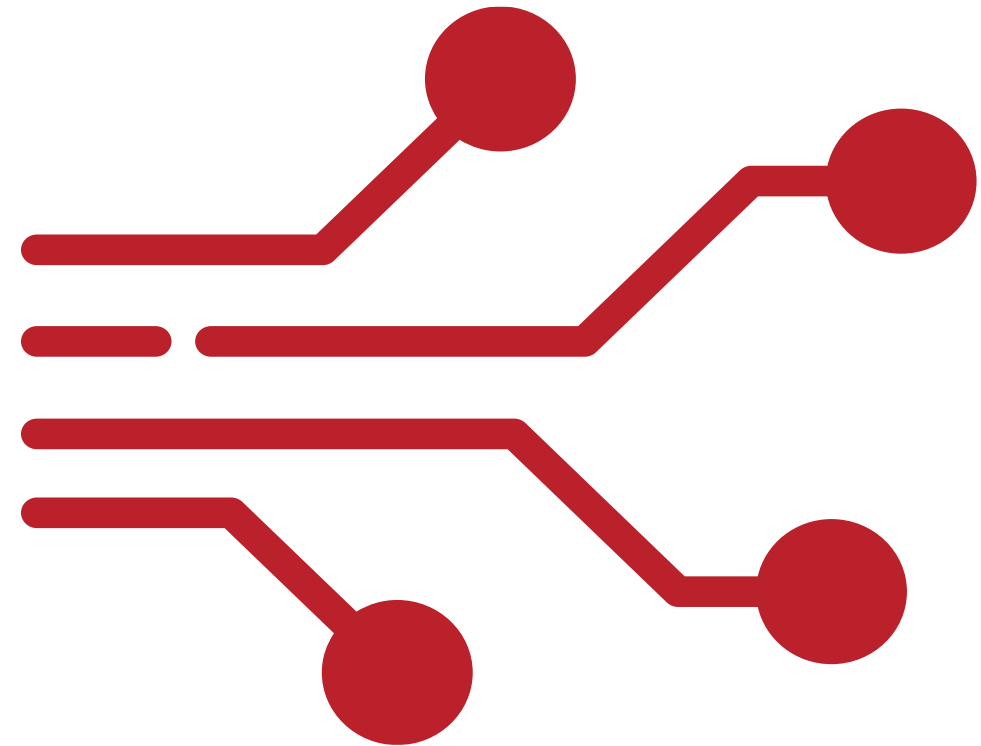
dr hab. inż. Michał Wydra, prof. Uczelni
Katedra Informatyki Stosowanej

Plan prezentacji

- Bezpieczeństwo jako cecha aplikacji webowej
- Podatności w aplikacjach internetowych
- Kontrola dostępu i konfiguracja
- Ochrona danych i komunikacji



Bezpieczeństwo jako cecha aplikacji webowej

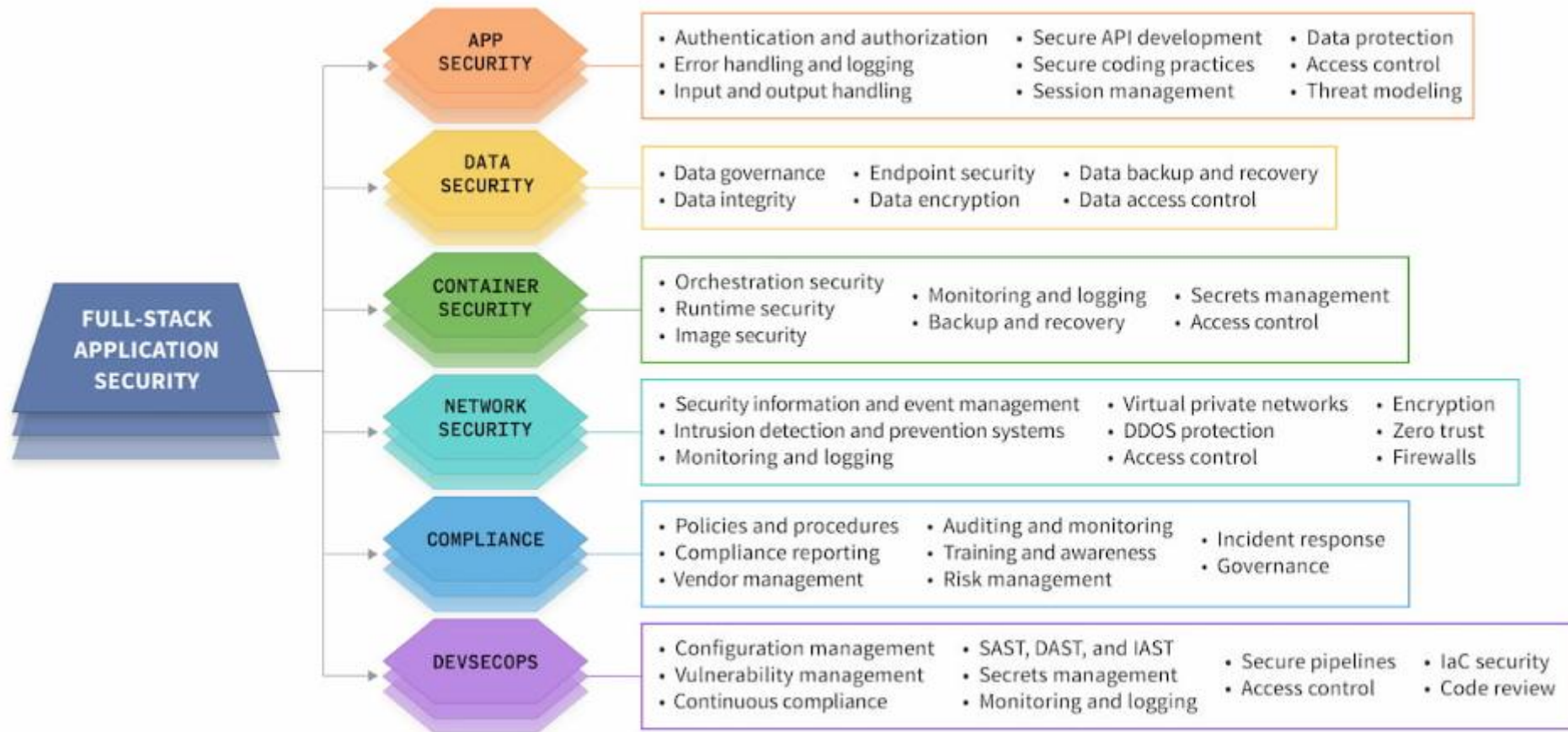


3

Bezpieczeństwo jako zagadnienie przekrojowe

Bezpieczeństwo aplikacji internetowych jest zagadnieniem przekrojowym, ponieważ obejmuje wszystkie warstwy systemu – od interfejsu użytkownika, przez logikę backendu, aż po bazę danych oraz infrastrukturę serwerową i sieciową. Każda z tych warstw realizuje inne zadania, ale jednocześnie każda może stać się potencjalnym celem ataku. Z tego powodu bezpieczeństwo nie może być traktowane jako pojedyncza funkcjonalność ani dodatek na końcu projektu.

Jest to raczej zbiór zasad, dobrych praktyk i mechanizmów, które muszą być uwzględniane na każdym etapie tworzenia aplikacji – od projektowania architektury, przez implementację kodu, aż po wdrożenie i utrzymanie systemu. Zaniedbanie bezpieczeństwa w jednej warstwie, na przykład brak walidacji danych w backendzie lub niewłaściwa konfiguracja serwera, może prowadzić do poważnych konsekwencji dla całego systemu. Dlatego skuteczne podejście do bezpieczeństwa wymaga spójnej strategii oraz świadomości zagrożeń na wszystkich poziomach aplikacji webowej.



Rys. 1. Elementy bezpieczeństwa aplikacji [źródło: <https://dzone.com/articles/full-stack-security-guide-best-practices-challenges>]

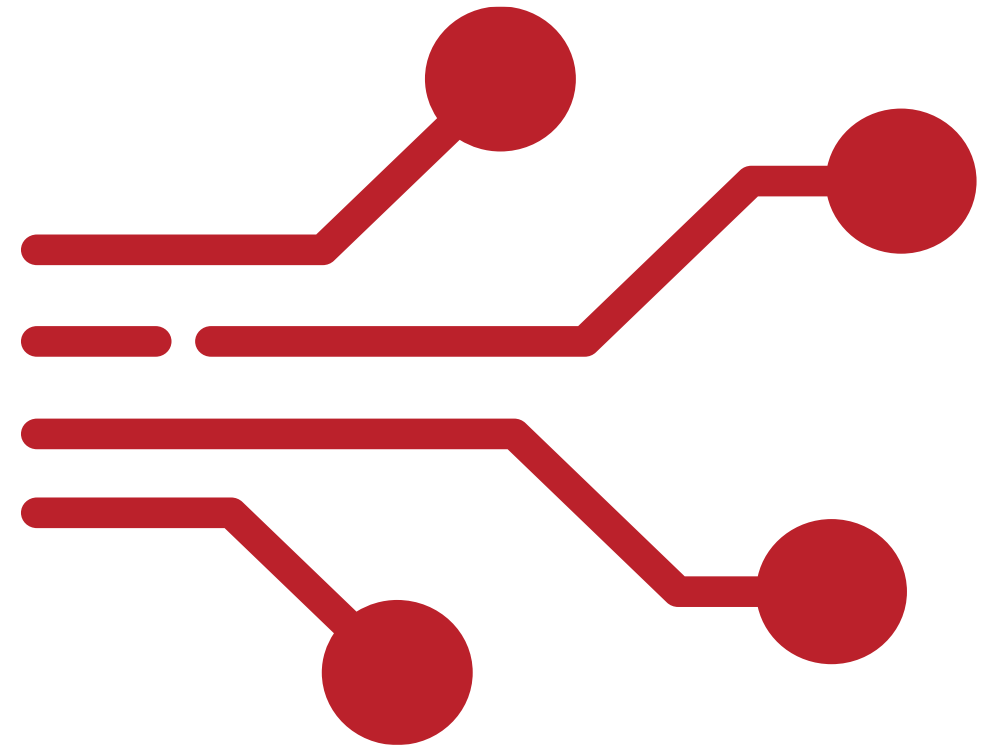
Dlaczego bezpieczeństwo aplikacji jest tak ważne?

Bezpieczeństwo aplikacji internetowych jest niezwykle istotne, ponieważ bardzo często przetwarzają one dane wrażliwe, takie jak dane osobowe użytkowników, hasła, adresy e-mail czy informacje finansowe. Naruszenie poufności tych danych, ich wyciek lub nieautoryzowana modyfikacja mogą prowadzić do poważnych konsekwencji zarówno dla użytkowników, jak i dla właścicieli systemu. Skutkiem takich incydentów bywają straty finansowe, kradzież tożsamości czy nadużycia związane z dostępem do kont.

Brak odpowiednich zabezpieczeń wpływa również negatywnie na reputację organizacji i zaufanie użytkowników. Utrata wiarygodności może skutkować odpływem klientów oraz długotrwałymi konsekwencjami wizerunkowymi. Dodatkowo wiele aplikacji podlega regulacjom prawnym dotyczącym ochrony danych, co oznacza, że naruszenia bezpieczeństwa mogą prowadzić do sankcji prawnych i kar finansowych. Z tego powodu zapewnienie bezpieczeństwa aplikacji webowych jest nie tylko wyzwaniem technicznym, ale także ważnym obowiązkiem organizacyjnym i prawnym.

6

Podatności w aplikacjach internetowych



Czym są podatności bezpieczeństwa?

Podatności bezpieczeństwa to słabości w aplikacji internetowej, jej kodzie lub konfiguracji, które mogą zostać wykorzystane przez atakującego do nieautoryzowanego działania. Mogą one dotyczyć zarówno logiki aplikacji, sposobu przetwarzania danych, jak i ustawień serwera czy infrastruktury. Same w sobie nie oznaczają jeszcze, że doszło do ataku, jednak znacząco zwiększają ryzyko jego wystąpienia.

Najczęściej podatności wynikają z niewłaściwej walidacji danych wejściowych, błędnej konfiguracji systemu, braku aktualizacji oprogramowania lub nieznajomości podstawowych zasad bezpieczeństwa. Ich istnienie stwarza realne zagrożenie dla poufności, integralności i dostępności danych. Dlatego kluczowe znaczenie ma ich wczesne wykrywanie oraz eliminowanie na etapie projektowania, implementacji i utrzymania aplikacji.

OWASP i znaczenie standardów bezpieczeństwa

OWASP (Open Web Application Security Project) to międzynarodowa, otwarta inicjatywa, której celem jest podnoszenie poziomu bezpieczeństwa aplikacji internetowych. Organizacja ta tworzy materiały edukacyjne, dokumentację oraz narzędzia, które pomagają programistom i zespołom technicznym projektować oraz rozwijać bezpieczne systemy webowe.

Jednym z najważniejszych osiągnięć OWASP jest lista OWASP Top 10, opisująca najczęściej występujące i najbardziej krytyczne zagrożenia bezpieczeństwa aplikacji internetowych. Zestawienie to stanowi punkt odniesienia dla programistów, testerów i zespołów bezpieczeństwa, pomagając identyfikować ryzyka oraz wdrażać odpowiednie zabezpieczenia. Stosowanie standardów i rekomendacji OWASP znacząco zwiększa poziom bezpieczeństwa aplikacji i pozwala ograniczyć podatności już na etapie projektowania i implementacji systemu.

Top 10 CI/CD Security Risks



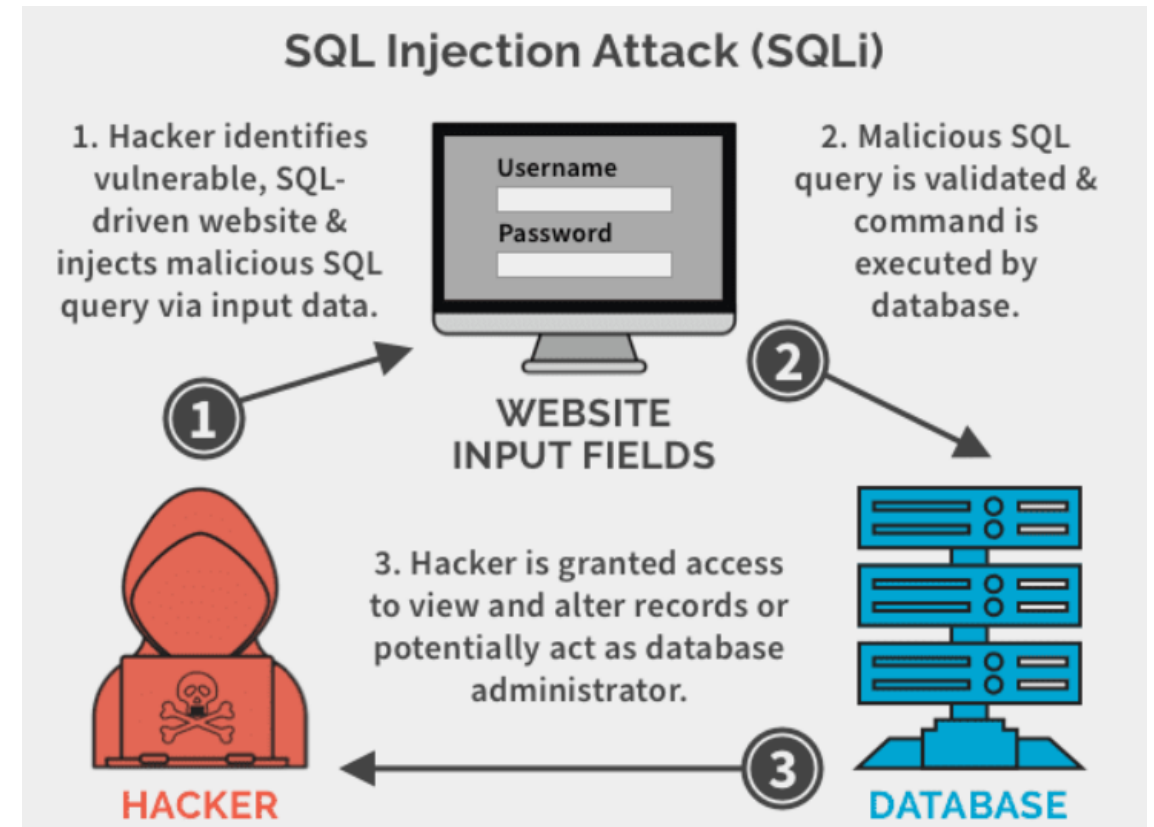
CICD-SEC-1	Insufficient Flow Control Mechanisms
CICD-SEC-2	Inadequate Identity and Access Management
CICD-SEC-3	Dependency Chain Abuse
CICD-SEC-4	Poisoned Pipeline Execution (PPE)
CICD-SEC-5	Insufficient PBAC (Pipeline-Based Access Controls)
CICD-SEC-6	Insufficient Credential Hygiene
CICD-SEC-7	Insecure System Configuration
CICD-SEC-8	Ungoverned Usage of 3rd Party Services
CICD-SEC-9	Improper Artifact Integrity Validation
CICD-SEC-10	Insufficient Logging and Visibility

Rys. 2. Lista OWASP Top 10 [źródło: <https://owasp.org/www-project-top-10-ci-cd-security-risks/>]

10

Ataki typu Injection

Ataki typu injection polegają na wstrzyknięciu złośliwego kodu do danych przesyłanych przez użytkownika. Jeśli serwer nie sprawdza poprawności tych danych, może dojść do wykonania nieautoryzowanych poleceń, na przykład zapytań SQL. Jest to jedno z najczęściej spotykanych i najbardziej niebezpiecznych zagrożeń w aplikacjach webowych.

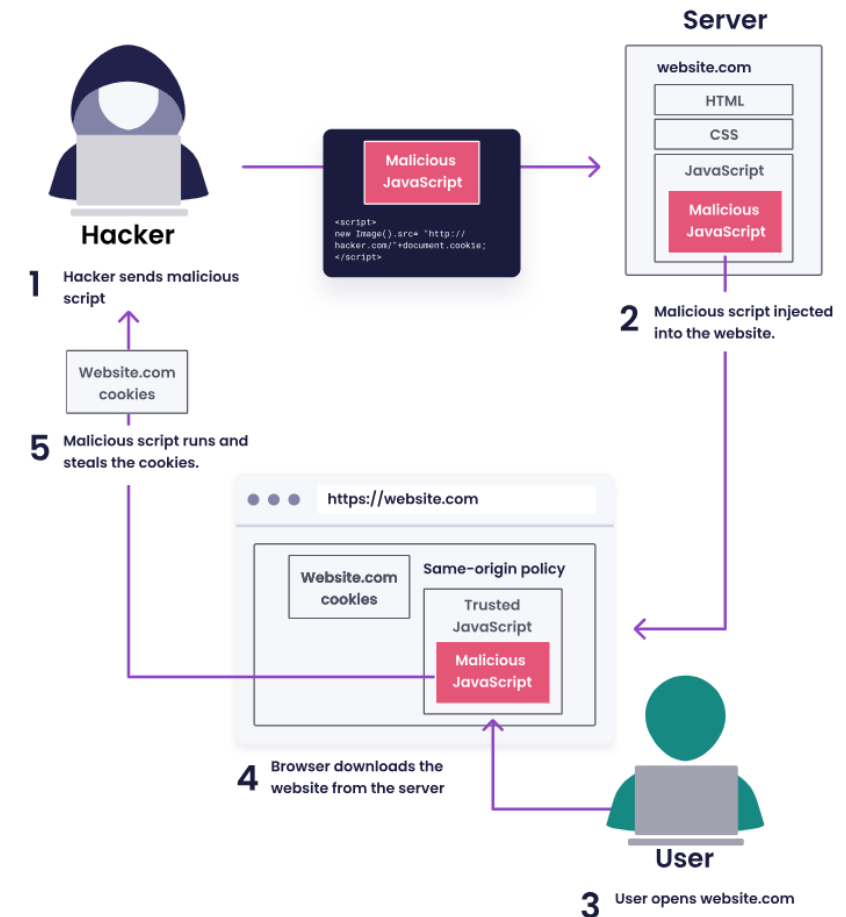


Rys. 3. Atak SQL Injection [źródło: <https://www.spanning.com/blog/sql-injection-attacks-web-based-application-security-part-4/>]

11

Cross-Site Scripting (XSS)

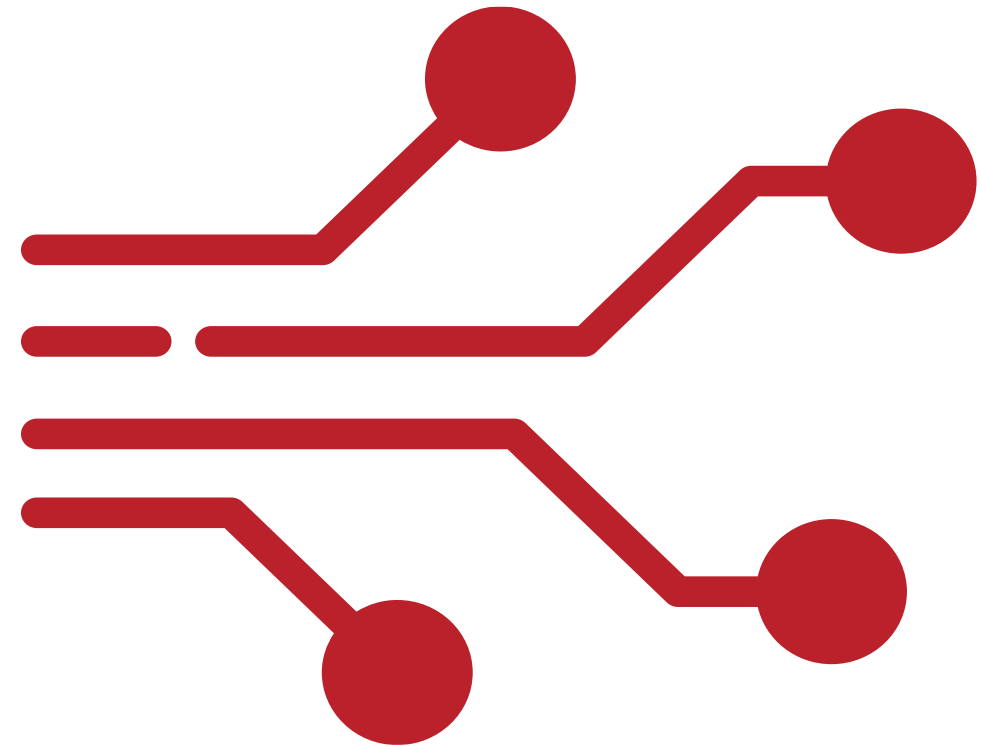
Cross-Site Scripting to atak polegający na wstrzyknięciu złośliwego kodu JavaScript do strony internetowej. Kod ten jest następnie wykonywany w przeglądarkach innych użytkowników. XSS może prowadzić do kradzieży danych, przejęcia sesji lub manipulowania zawartością strony.



Rys. 4. Schemat działania ataku XSS [źródło: <https://learn.snyk.io/lesson/xss/>]

12

Kontrola dostępu i konfiguracja



13

Broken Authentication

Broken Authentication to kategoria podatności związanych z błędami w mechanizmach logowania, uwierzytelniania oraz zarządzania sesją użytkownika. Występuje wtedy, gdy te mechanizmy są nieprawidłowo zaprojektowane lub zaimplementowane, co może umożliwić atakującym przejęcie kont użytkowników bez konieczności poznania ich danych logowania.

Do typowych problemów należą słabe hasła, brak ochrony przed wielokrotnymi próbami logowania, niewłaściwe zarządzanie identyfikatorami sesji czy brak ich unieważniania po wylogowaniu. Broken authentication jest szczególnie niebezpieczne w aplikacjach przetwarzających dane wrażliwe lub oferujących funkcje administracyjne, ponieważ przejęcie jednego konta może prowadzić do nieautoryzowanego dostępu do całego systemu. Dlatego poprawna implementacja mechanizmów uwierzytelniania i sesji ma kluczowe znaczenie dla bezpieczeństwa aplikacji webowych.

Broken Access Control

Broken Access Control to podatność bezpieczeństwa polegająca na niewłaściwym egzekwowaniu zasad autoryzacji w aplikacji webowej. W takiej sytuacji użytkownik może uzyskać dostęp do zasobów, danych lub funkcji, do których nie powinien mieć uprawnień, na przykład poprzez zmianę identyfikatora w adresie URL lub wywołanie niedozwolonego endpointu API.

Problem ten bardzo często wynika z braku odpowiednich sprawdzeń po stronie serwera lub z polegania wyłącznie na ograniczeniach po stronie klienta. Skutkiem broken access control mogą być nieautoryzowane operacje, takie jak podgląd danych innych użytkowników, modyfikacja cudzych zasobów czy uzyskanie dostępu do funkcji administracyjnych. Dlatego kontrola dostępu powinna być zawsze realizowana i weryfikowana po stronie backendu, niezależnie od interfejsu użytkownika.

15

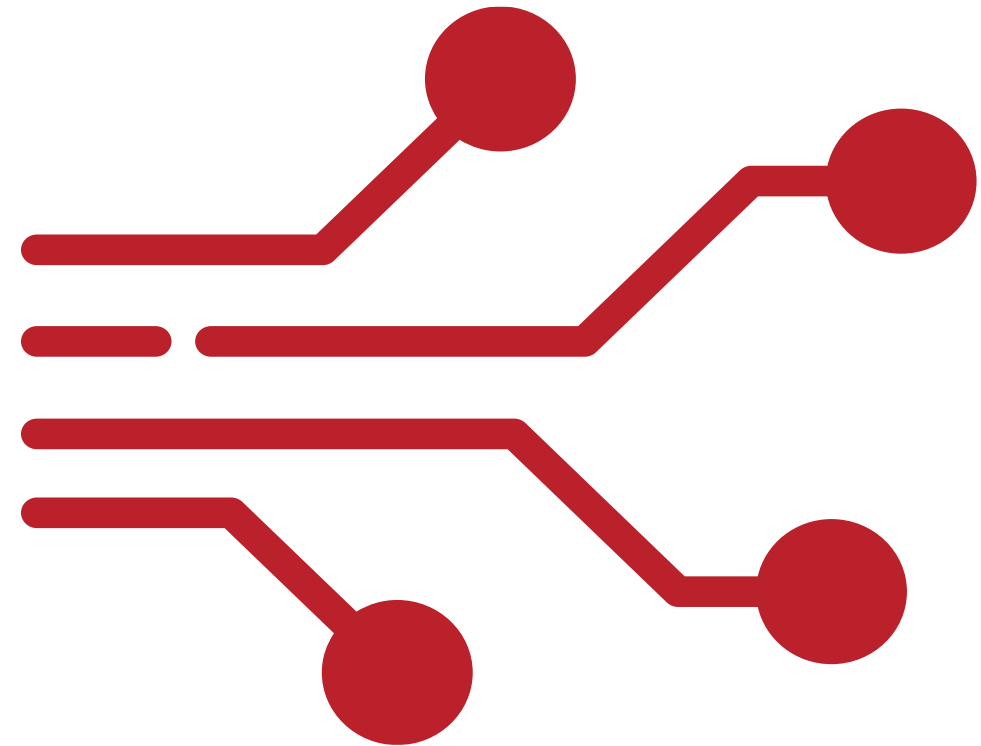
Security Misconfiguration

Security Misconfiguration to kategoria podatności wynikająca z nieprawidłowej konfiguracji aplikacji, serwera lub infrastruktury, a nie bezpośrednio z błędów w kodzie. Może ona prowadzić do ujawnienia informacji technicznych, danych konfiguracyjnych lub dodatkowych punktów wejścia, które ułatwiają przeprowadzenie ataku. Często dotyczy to ustawień domyślnych, które nie zostały odpowiednio zmienione przed wdrożeniem systemu do środowiska produkcyjnego.

Przykładami takich błędów są pozostawione konta testowe, włączone tryby debugowania, zbyt szczegółowe komunikaty błędów czy niepotrzebnie otwarte porty i usługi. Security misconfiguration jest szczególnie niebezpieczne, ponieważ może dostarczać atakującemu cennych informacji o architekturze systemu i ułatwiać dalsze ataki. Dlatego właściwa konfiguracja środowiska, regularne audyty oraz stosowanie zasad minimalnych uprawnień są kluczowymi elementami bezpiecznego wdrażania aplikacji webowych.

16

Ochrona danych i komunikacji



17

Ochrona danych wrażliwych

Ochrona danych wrażliwych jest jednym z najważniejszych aspektów bezpieczeństwa aplikacji internetowych. Informacje takie jak dane osobowe, hasła, numery kart płatniczych czy dane medyczne powinny być przechowywane i przetwarzane w sposób, który minimalizuje ryzyko ich nieautoryzowanego ujawnienia. Dotyczy to zarówno danych zapisanych w bazach danych, jak i informacji przesyłanych pomiędzy klientem a serwerem.

Ogromne znaczenie ma stosowanie szyfrowania danych w trakcie transmisji, na przykład poprzez bezpieczne połączenia HTTPS, oraz zabezpieczanie danych przechowywanych na serwerze przy użyciu odpowiednich mechanizmów kryptograficznych. Równie istotne jest ograniczanie ilości przetwarzanych danych do absolutnego minimum oraz kontrola dostępu do nich. Takie podejście zmniejsza potencjalne skutki ewentualnego naruszenia bezpieczeństwa i znacząco podnosi poziom ochrony informacji w aplikacjach webowych.

Szyfrowanie i HTTPS

HTTPS to bezpieczna wersja protokołu HTTP, która wykorzystuje szyfrowanie do ochrony danych przesyłanych pomiędzy przeglądarką użytkownika a serwerem. Dzięki zastosowaniu mechanizmów kryptograficznych, opartych na protokole TLS, informacje przesyłane przez sieć są chronione przed podsłuchem, przechwyceniem oraz nieautoryzowaną modyfikacją. Zapewnia to poufność i integralność komunikacji.

Stosowanie HTTPS pozwala również na uwierzytelnienie serwera, co daje użytkownikowi pewność, że łączy się z właściwą i zaufaną stroną. Z tego powodu HTTPS jest obecnie standardem w nowoczesnych aplikacjach internetowych i stanowi podstawowy element bezpieczeństwa. Jego brak może nie tylko narażać dane użytkowników, ale także negatywnie wpływać na wiarygodność aplikacji oraz jej ocenę przez przeglądarki i wyszukiwarki.

Podstawy kryptografii w aplikacjach webowych

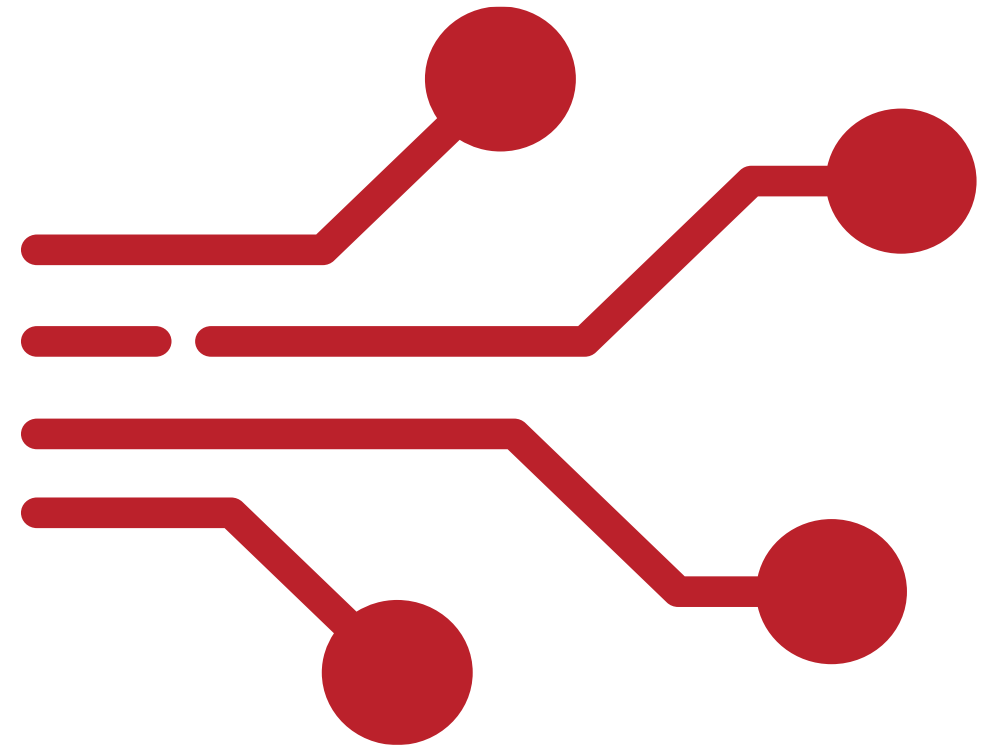
Kryptografia stanowi podstawę bezpieczeństwa w aplikacjach webowych, ponieważ umożliwia bezpieczne szyfrowanie i odszyfrowywanie danych. Jej głównym celem jest ochrona poufności informacji, zapewnienie ich integralności oraz potwierdzenie tożsamości stron komunikacji. Dzięki kryptografii dane mogą być przesyłane przez niezaufane sieci, takie jak Internet, bez ryzyka ich podsłuchania lub zmodyfikowania.

W aplikacjach internetowych stosuje się zarówno algorytmy symetryczne, jak i asymetryczne. Algorytmy symetryczne wykorzystują ten sam klucz do szyfrowania i odszyfrowywania danych, co czyni je bardzo wydajnymi i odpowiednimi do szyfrowania dużych ilości informacji. Algorytmy asymetryczne operują natomiast parą kluczy – publicznym i prywatnym – i są wykorzystywane m.in. do bezpiecznej wymiany kluczy oraz uwierzytelniania stron.

Połączenie obu podejść pozwala osiągnąć wysoki poziom bezpieczeństwa i wydajności. W praktyce algorytmy asymetryczne służą do ustanowienia bezpiecznego połączenia i wymiany klucza, a algorytmy symetryczne do dalszej, szybkiej komunikacji. Takie rozwiązanie jest fundamentem mechanizmów stosowanych w HTTPS i stanowi podstawę ochrony poufnych informacji w nowoczesnych aplikacjach webowych.

20

Bezpieczeństwo po stronie przeglądarki i serwera



21

Same-Origin Policy i CORS

Same-Origin Policy (SOP) to jedna z podstawowych zasad bezpieczeństwa przeglądarek internetowych. Ogranicza ona możliwość dostępu skryptów do zasobów pochodzących z innej domeny (origin), czyli różniących się protokołem, domeną lub portem. Dzięki temu złośliwa strona nie może swobodnie odczytywać danych z innych witryn, na przykład informacji sesyjnych czy odpowiedzi API, co znacząco zwiększa bezpieczeństwo użytkownika.

W praktyce jednak nowoczesne aplikacje webowe często wymagają komunikacji pomiędzy różnymi domenami, na przykład gdy frontend i backend są hostowane oddzielnie. W takich sytuacjach stosuje się mechanizm CORS (Cross-Origin Resource Sharing), który pozwala w kontrolowany sposób złagodzić ograniczenia SOP. CORS musi być świadomie skonfigurowany po stronie serwera i polega na określeniu, które domeny, metody HTTP oraz nagłówki są dozwolone. Poprawna konfiguracja CORS umożliwia bezpieczną współpracę pomiędzy systemami, natomiast błędna może prowadzić do poważnych luk bezpieczeństwa.

22

Content Security Policy (CSP)

Content Security Policy (CSP) to zaawansowany mechanizm bezpieczeństwa przeglądarki, który pozwala precyzyjnie określić, jakie zasoby mogą być ładowane i wykonywane w ramach danej strony internetowej. Polityka CSP jest definiowana po stronie serwera, najczęściej za pomocą odpowiednich nagłówków HTTP, i wskazuje m.in. dozwolone źródła skryptów, stylów, obrazów czy ramek iframe.

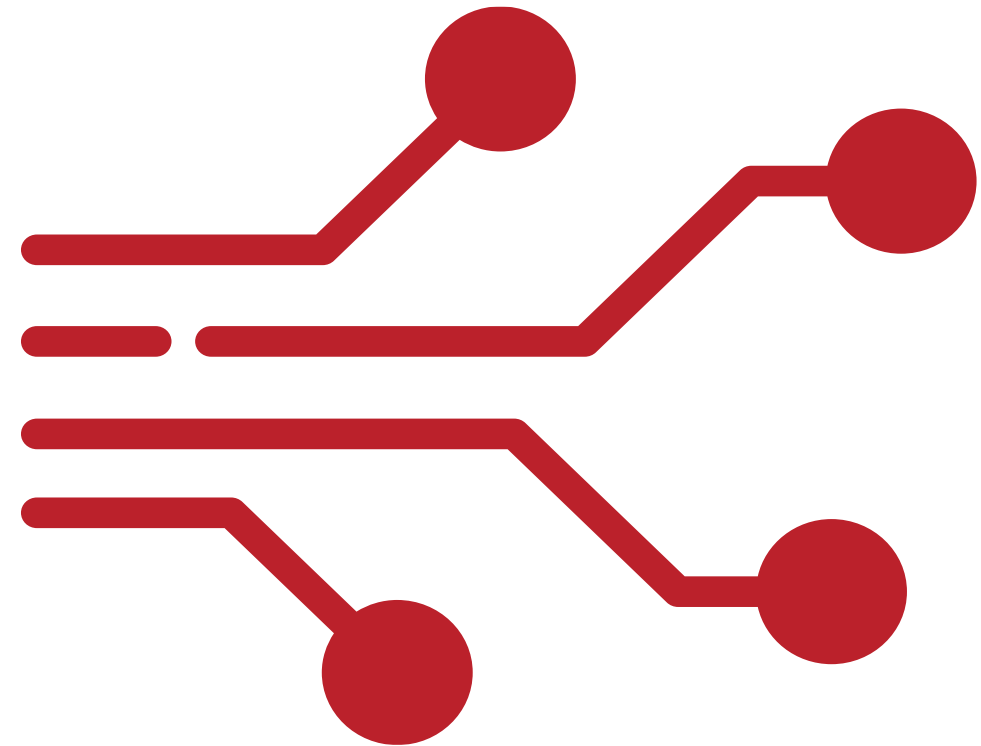
Dzięki CSP aplikacja zyskuje dodatkową warstwę ochrony przed atakami typu Cross-Site Scripting. Nawet jeśli do strony zostanie wstrzyknięty złośliwy kod, przeglądarka zablokuje jego wykonanie, jeśli nie pochodzi on z zaufanego źródła określonego w polityce bezpieczeństwa. W rezultacie CSP znacząco ogranicza skuteczność ataków XSS i pomaga chronić użytkowników przed nieautoryzowanym uruchamianiem kodu w przeglądarce.

Monitorowanie i aktualizacje

Bezpieczeństwo aplikacji internetowej nie kończy się w momencie jej wdrożenia do środowiska produkcyjnego. Jest to proces ciągły, który wymaga regularnej kontroli i reagowania na zmieniające się zagrożenia. Aktualizacje zależności, bibliotek oraz środowiska uruchomieniowego są niezbędne, ponieważ wiele podatności bezpieczeństwa jest odkrywanych dopiero po pewnym czasie i usuwanych w kolejnych wersjach oprogramowania.

Równie istotne jest monitorowanie logów systemowych i aplikacyjnych, które pozwala wykrywać nietypowe lub podejrzane zachowania, takie jak próby nieautoryzowanego dostępu, błędy uwierzytelniania czy nagłe skoki aktywności. Brak monitoringu może sprawić, że ataki pozostaną niewykryte przez długi czas, zwiększając skalę potencjalnych strat. Świadome podejście do monitorowania i aktualizacji stanowi kluczowy element utrzymania długoterminowego bezpieczeństwa aplikacji webowych.

Podsumowanie



25

Bezpieczeństwo aplikacji internetowych jest zagadnieniem złożonym i wielowymiarowym, które wymaga uwzględnienia wielu aspektów technicznych i organizacyjnych. Obejmuje ono ochronę danych wrażliwych, bezpieczną komunikację pomiędzy klientem a serwerem, poprawną implementację mechanizmów uwierzytelniania i autoryzacji oraz właściwą konfigurację aplikacji i infrastruktury. Każdy z tych elementów ma wpływ na ogólny poziom bezpieczeństwa systemu.

Znajomość najczęstszych zagrożeń, takich jak ataki typu injection, XSS czy problemy z kontrolą dostępu, pozwala świadomie projektować i rozwijać aplikacje webowe. Stosowanie sprawdzonych standardów, regularne aktualizacje oraz ciągłe monitorowanie systemu sprawiają, że aplikacje stają się bardziej odporne na ataki i bezpieczniejsze dla użytkowników.

Literatura:

- [1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.
- [2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.
- [3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.
- [4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.
- [5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Dziękuję za uwagę