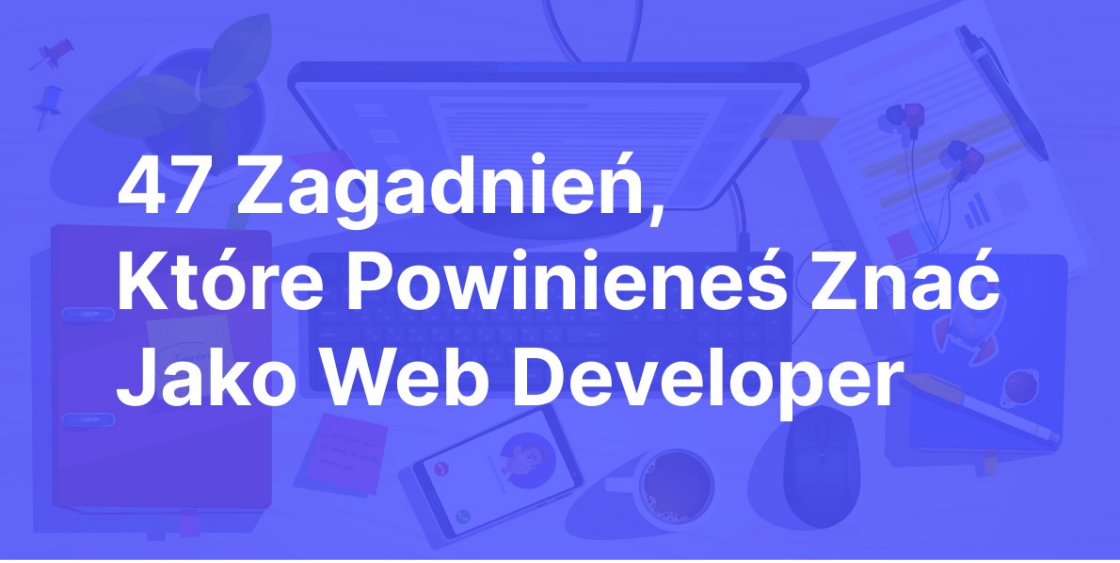

47 Zagadnień, Które Powinieneś Znać Jako Web Developer

- W jaki sposób działają **DNSy**?
- Co oznaczają **liczby** oddzielone kropkami w wersjach oprogramowania?
- Czym różnią się **biblioteki** od **frameworków**?
- Co to są bezgłowe **CMSy**?
- Do czego wykorzystuje się **brokerów wiadomości**?
- Jak tworzyć natywne **web komponenty**?
- Co łączy **RESTful**, **GraphQL** i **RPC**?
- Czym różni się **autentykacja** od **autoryzacji**?
- Co to jest **CI/CD**?

Odpowiedzi na te i wiele innych pytań znajdziesz **w tym ebooku**.

Adrian Bienias
adrianbienias.pl

Spis treści

Web Development w Pigułce

47 Zagadnień, Które Powinieneś Znać Jako Web Developer

- W jaki sposób działają **DNSy**?
- Co oznaczają **liczby** oddzielone kropkami w wersjach oprogramowania?
- Czym różnią się **biblioteki** od **frameworków**?
- Co to są bezgłowe **CMSy**?
- Do czego wykorzystuje się **brokerów wiadomości**?
- Jak tworzyć natywne **web komponenty**?
- Co łączy **RESTful**, **GraphQL** i **RPC**?
- Czym różni się **autentykacja** od **autoryzacji**?
- Co to jest **CI/CD**?

Odpowiedzi na te i wiele innych pytań znajdziesz **w tym ebooku**.

Adrian Bienias
adrianbienias.pl

O Autorze

Adrian Bienias, full stack web developer z ponad 20 letnim doświadczeniem w tworzeniu stron internetowych i ponad 10 letnim doświadczeniem w tworzeniu szkoleń z zakresu web developmentu.

Były inżynier oprogramowania Atlassian. Rozwijał oprogramowanie Jira, używane na całym świecie przez wielkie korporacje jak np. Twitter, Uber, Lufthansa oraz agencje kosmiczne SpaceX i NASA.

Twórca serwisu Codisity, serwisu do nauki projektowania i tworzenia stron internetowych.

Podstawy

1. Praca w IT

Definicja IT

IT to skrót z języka angielskiego od słów *Information Technology*.

W języku polskim funkcjonuje w kilku wariantach jako Technika/Technologia Informacyjna/Informatyczna.

Branża IT to nie tylko programowanie. To wszystkie działy powiązane z tworzeniem, przetwarzaniem, odbieraniem, przechowywaniem i wymianą wszelkiego rodzaju informacji (danych).

Jak każda inna branża, oprócz osób technicznych, potrzebuje również stanowisk związanych z zarządzaniem, marketingiem czy obsługą klientów.

Bardzo dobre zarobki

Ze względu na niematerialny charakter produktów, branża IT ma dużą marżowość.

Dostarczanie produktów klientom wiąże się z bardzo niskimi kosztami logistyki (w porównaniu do produktów fizycznych).

Międzynarodowy zasięg działania ma też istotny wpływ na wyższe wynagrodzenia w krajach z niższą siła nabywczą rodzimej waluty.

Możliwość pracy zdalnej

Dodatkowym atutem pracy w IT jest możliwość pracy zdalnej.

Okres pandemii znacząco przyspieszył rozwój technologi i rozwiązań ułatwiających zdalną komunikację i zarządzanie.

Nieograniczony rozwój

Ze względu na dużą ilość używanych technologi oraz ich stałą i dynamiczną ewolucję, pracując w IT, można, a nawet trzeba stale się rozwijać.

Ilość tematów, w które można się zagłębić jest ogromna i stale pojawiają się nowe. Zawsze znajdzie się miejsce na innowacje i szukanie coraz to lepszych rozwiązań.

Podział na stanowiska pracy

Gałąź branży IT związana z tworzeniem stron/aplikacji internetowych klasyfikuje zazwyczaj techniczne stanowiska na poniższe warianty:

UX/UI Designer

Projektuje doświadczenia i interfejs użytkowników.

W skrócie praca projektantów UX/UI polega na zaspokajaniu potrzeb poprzez tworzoną stronę/aplikację (np. zamawianie taksówek) w możliwie najbardziej intuicyjny sposób, zapewniając jednocześnie pozytywne doznania wizualne.

Frontend Developer

Za pomocą kodu realizuje projekty wykonane w procesie designu UX/UI.

Kod, który pisze jest finalnie uruchamiany bezpośrednio w przeglądarce końcowego użytkownika.

Backend Developer

Tworzy wszelkie funkcjonalności strony/aplikacji, które uruchamiane są na serwerze.

Oznacza to pisanie oprogramowania np. związanego z korzystaniem z baz danych, szyfrowaniem haseł użytkowników, a finalnie tworzeniem API dla Frontendu.

DevOps

DevOps to skrót od połączenia dwóch wyrazów *Development* i *Operations*.

Dział/stanowisko łączące wytwarzanie oprogramowania z dostarczaniem go klientom i końcowym użytkownikom.

Zajmuje się zapewnieniem infrastruktury dla działania strony/aplikacji oraz sprawnego jej wytwarzania.

Administruje serwerami, dba o monitorowanie zasobów (czy np. nie brakuje mocy obliczeniowej dla obsługiwań zapytań użytkowników), czy zapewnia płynny proces wdrażania zmian w kodzie aplikacji.

Tester (QA/QE)

Dba o jakość tworzonej strony/aplikacji poprzez testowanie jej działania.

Wykrywa powstałe błędy w oprogramowaniu zanim owe oprogramowanie trafi do końcowego użytkownika.

Inne (np. stanowiska menadżerskie)

Podobnie jak w innych branżach, aspekt biznesowy musi być kierowany odgórnie, aby tworzona strona/aplikacja spełniała nie tylko potrzeby użytkowników, ale też i wymogi inwestorów.

Oferty pracy oraz raporty wynagrodzeń

Sprawdź jak wyglądają aktualne oferty pracy z widełkami wynagrodzeń w poniższych serwisach:

JustJoinIT

- Just Join IT: #1 Job Board for tech industry in Europe
- Raport wynagrodzeń w IT za rok 2021

NoFluffJobs

- Portal z ofertami pracy IT | Praca IT dla Ciebie | No Fluff Jobs
- Kalkulator zarobków na No Fluff Jobs

BulldogJob

- Oferty pracy z IT
- Badanie Społeczności IT 2022 - Raport.

SOLID.jobs

- SOLID.Jobs – Platforma rekrutacyjna dla specjalistów

Dodatkowe linki

- Obowiązki w pracy na stanowisku UX/UI Designer

2. Dokumentacje

Każda z technologi związana z web developmentem ma swoją oficjalną dokumentację.

Dokumentacje są instrukcjami użytkowania. Czytając je uczysz się, jak korzystać z danej technologi.

Kupując np. pralkę, otrzymujesz instrukcję użytkowania. Nie zaglądając do instrukcji, możesz wydedukować sposób korzystania z urządzenia na podstawie dotychczasowych doświadczeń z innymi urządzeniami.

Im jednak urządzenie jest bardziej skomplikowane, tym większa szansa, że użyjesz go w sposób niewłaściwy. Prawdopodobnie nie poznasz też wielu funkcji ukrytych pod kombinacjami np. kilku przycisków, lub zaniedbasz kwestie związane z konserwacją.

Podobnie jest w przypadku technologi webowych. Mając już jakieś doświadczenie z korzystania z kilku technologi można naiwnie założyć, że sami lepiej zrozumiemy działanie.

Dokumentacje mogą być oczywiście lepsze lub gorsze. Nawet, gdy swoją formą zniechęcają do korzystania z nich, są najbardziej wartościowym źródłem wiedzy o danej technologi.

Jako web developer musimy mieć to na uwadze i kierować wszelkie swoje wątpliwości co do działania danej technologi, prosto do dokumentacji.

Specyfikacja, implementacja, dokumentacja

... taka sytuacja.

Specyfikacja to zbiór zasad, które powinny być przestrzegane podczas implementacji danej technologi. Implementacja to stworzenie oprogramowania działającego według zasad opisanych w specyfikacji.

Pisząc własne oprogramowanie np. stronę internetową, tworzysz ją zgodnie ze swoją specyfikacją, albo specyfikacją klienta.

Przykładowo: blog z opcją wizualnego tworzenia i edycji wpisów oraz z systemem komentarzy. To jest specyfikacja. Bardzo ogólna, ale nakreśla ramy, w których powinieneś ująć działanie kodu.

Kod, który napiszesz, będzie implementacją specyfikacji.

Instrukcja, którą przekażesz klientowi, w której wyjaśnisz jak korzystać z Twojej implementacji będzie dokumentacją.

Czytanie dokumentacji

Przed rozpoczęciem pracy z nową technologią (językiem, frameworkiem, biblioteką), zapoznaj się z jej dokumentacją, aby lepiej zrozumieć jej działanie i sposób użycia.

Korzystaj z indeksów, spisów treści, aby szybko nawigować po interesujących Cię informacjach. Wiele dokumentacji technicznych zawiera również wyszukiwarkę, która pozwala przyspieszyć odnajdywanie informacji.

Poznając nową technologię, warto zapoznać się od razu z cała dokumentacją w pobieżny sposób. Pozwala to szybko spojrzeć na całą technologię z góry, niczym na mapę nieznanego miejsca, które planujemy w najbliższym czasie odwiedzić.

Nie chodzi o to, żeby “wykuć” cała dokumentację na pamięć. Ważne jest by mieć ogólny zarys możliwości i sposobów działania danej technologi, aby w przyszłości wiedzieć jak i gdzie odnajdywać szczegółowe informacje na dany temat.

Tworzenie dokumentacji

Dokumentowanie kodu, który piszemy jest bardzo ważne. Dobra dokumentacja ułatwia korzystanie innym ze stworzonych przez nas rozwiązań. Pomaga również nam, gdy wracamy po jakimś czasie do kodu, który napisaliśmy jakiś czas temu.

Tworzenie dokumentacji jest podobne do pisania instrukcji. Wyobraź sobie, że oprogramowanie, które tworzysz jest rzeczą fizyczną. Osoby, które będą korzystały z tego oprogramowania (również Ty sam w przyszłości) potrzebują instrukcji.

Czytanie i analizowanie kodu jest znacznie wolniejsze niż czytanie dokumentacji.

Wyobraź sobie, że pierwszy raz w życiu masz styczność z urządzeniem *T-1000*.

Możesz rozebrać cała konstrukcję starając się zrozumieć jak elementy się ze sobą łączą, jakie jest przeznaczenie urządzenia i jakie ma funkcjonalności.

Możesz też przeczytać instrukcję producenta, aby łatwiej zrozumieć działanie owego urządzania.

Pamiętaj, że jako web developer, jesteś właśnie takim producentem.

Dodatkowe linki

- Jak czytać dokumentację?
- Jak tworzyć dokumentację?

3. Internet

Internet to globalna sieć komputerowa, która składa się z wielu różnych komputerów i sieci połączonych ze sobą za pomocą różnych technologii komunikacyjnych, takich jak kable telefoniczne, światłowodowe, satelitarne i bezprzewodowe. Dzięki internetowi możliwe jest wymienianie się informacjami i danymi między komputerami na całym świecie w czasie rzeczywistym.

Dodatkowe linki

- Jak działa internet?
- How does the Internet work? | Cloudflare
- The internet, explained - Vox
- How Does the Internet Work?
- How does the Internet work? - Learn web development | MDN
- How does the internet work?
- How Does the Internet Work? - Glad You Asked S1 - YouTube

4. Protokół HTTP

Protokół HTTP (Hypertext Transfer Protocol) to standardowy protokół sieciowy, który jest używany do przesyłania danych między komputerami w Internecie. HTTP jest głównie używany do przesyłania stron internetowych z serwerów do przeglądarek internetowych, ale może być również używany do przesyłania innych rodzajów plików, takich jak obrazy, pliki audio i video, itp.

Działanie protokołu opiera się o parę klient-serwer, co oznacza, że istnieją dwa główne rodzaje komputerów w tej architekturze: klienci i serwery. Klienci to komputery, które żądają dostępu do zasobów, takich jak strony internetowe, a serwery to komputery, które są odpowiedzialne za przechowywanie i udostępnianie tych zasobów.

HTTP działa poprzez wysyłanie żądań z klienta do serwera i odpowiedzi z serwera na te żądania. Żądanie może obejmować różne rodzaje operacji, takich jak pobieranie strony internetowej, wysyłanie danych formularza do serwera lub wysyłanie pliku do serwera. Serwer odpowiada na żądanie przez wysłanie odpowiedzi, która może zawierać dane, takie jak strona internetowa lub informacja o błędzie.

Protokół HTTP jest często używany w połączeniu z językiem HTML (Hypertext Markup Language) do tworzenia stron internetowych, ale może być również używany z innymi językami, takimi jak XML lub JSON.

HTTP jest protokołem bezstanowym, co oznacza, że każde żądanie klienta jest niezależne od poprzednich i następnych żądań. Nie ma stałego połączenia pomiędzy klientem a

serwerem i każde żądanie jest przetwarzane niezależnie. Protokół HTTP jest protokołem niezabezpieczonym (w przeciwieństwie do protokołu HTTPS), co oznacza, że dane przesyłane przez daną sieć są w formie nieszyfrowanej i mogą być widoczne dla każdego w tej sieci.

Dodatkowe linki

- What is HTTP? | Cloudflare
- HTTP | MDN

5. DNSy

DNS (Domain Name System) to system nazw domenowych, który służy do mapowania nazw domen na adresy IP. W prostych słowach, DNS pozwala ludziom na korzystanie z przyjaznych dla użytkownika nazw domen, takich jak google.com, zamiast trudnych do zapamiętania adresów IP, takich jak 216.58.212.142.

Kiedy użytkownik wprowadza nazwę domeny do przeglądarki internetowej, przeglądarka wysyła żądanie do serwera DNS, aby uzyskać odpowiedni adres IP dla tej nazwy domeny. Serwer DNS jest podobny do książki telefonicznej, która mapuje nazwy domen na adresy IP. Gdy serwer DNS otrzyma żądanie, sprawdzi swoją bazę danych i zwróci odpowiedni adres IP. Następnie przeglądarka wysyła żądanie do serwera o podanym adresie IP, a serwer zwraca stronę internetową, która jest wyświetlana użytkownikowi.

Serwery DNS są rozmieszczone na całym świecie i działają wspólnie, aby umożliwić ludziom dostęp do stron internetowych z każdego miejsca. Jeśli ktoś wprowadzi nazwę domeny, która nie jest zarejestrowana w systemie DNS, otrzyma komunikat o błędzie, ponieważ serwer DNS nie będzie w stanie zwrócić odpowiedniego adresu IP.

Dodatkowe linki

- What is DNS? | How DNS works | Cloudflare
- How DNS works
- Mess with DNS

6. Przeglądarki internetowe

Przeglądarka internetowa to oprogramowanie, które pozwala użytkownikom przeglądać strony internetowe. Aby to zrobić, przeglądarka łączy się z serwerem, na którym znajduje się strona internetowa, a następnie pobiera plik HTML, który jest kodem źródłowym strony internetowej. Przeglądarka analizuje ten kod HTML i wyświetla treść strony internetowej na ekranie użytkownika.

Przeglądarka internetowa może również wyświetlać inne rodzaje plików, takie jak obrazy, pliki CSS i JavaScript.

Aby użytkownik mógł przeglądać różne strony internetowe, przeglądarka internetowa musi mieć możliwość przełączania się pomiędzy różnymi adresami URL. Użytkownik może to zrobić, wpisując adres URL w pole wyszukiwania lub klikając na odnośnik na innej stronie internetowej. Po wprowadzeniu adresu URL lub kliknięciu odnośnika, przeglądarka internetowa ponownie łączy się z serwerem i pobiera nową stronę internetową.

Przeglądarki internetowe różnią się między sobą funkcjami i wyglądem, ale ich podstawowa funkcja pozostaje ta sama: umożliwienie użytkownikom przeglądania stron internetowych.

Narzędzia deweloperskie

Narzędzia deweloperskie to zestaw narzędzi, które są dostępne w przeglądarce internetowej i służą do debugowania, testowania i tworzenia stron internetowych.

Narzędzia deweloperskie zazwyczaj zawierają takie funkcje, jak:

- **Inspektor elementów**: pozwala przeglądać kod HTML i CSS strony internetowej oraz modyfikować go na bieżąco, aby zobaczyć, jak zmiany wpłyną na wygląd strony
- **Konsola**: pozwala wyświetlać komunikaty i błędy związane z kodem strony internetowej oraz wykonywać polecenia JavaScript na stronie
- **Narzędzie do profilowania i optymalizacji**: pozwala monitorować wydajność strony internetowej i identyfikować miejsca, w których kod jest słabo wydajny
- **Sieć**: pozwala podejrzeć ruch sieciowy wygenerowany przez połączenie się ze stroną internetową
- **Narzędzie do testowania responsywności**: pozwala sprawdzić, jak strona internetowa wygląda na różnych urządzeniach i rozdzielczościach ekranu

Dodatkowe linki

- How browsers work
- Role of Rendering Engines in Browsers | BrowserStack
- Populating the page: how browsers work - Web performance | MDN
- How Do Web Browsers Work? - YouTube
- Web technology for developers | MDN
- 15 DevTool Secrets for JavaScript Developers

7. Hosting

Istnieje wiele rodzajów usług hostingowych, które różnią się między sobą dostępnymi funkcjami, poziomem obsługi i ceną.

Hosting współdzielony

Najtańszy rodzaj usługi hostingowej, w której wiele stron internetowych dzieli jeden serwer. Hosting współdzielony jest odpowiedni dla małych stron internetowych, które nie wymagają dużo mocy obliczeniowej ani dużej przestrzeni dyskowej.

Hosting VPS (Virtual Private Server)

Rodzaj usługi hostingowej, w której strona internetowa dzieli serwer z innymi stronami, ale ma do dyspozycji więcej mocy obliczeniowej i przestrzeni dyskowej niż w przypadku hosting współdzielonego. Hosting VPS jest odpowiedni dla stron internetowych o średnim ruchu.

Hosting dedykowany

Rodzaj usługi hostingowej, w której strona internetowa ma do dyspozycji cały serwer. Jest to najdroższy rodzaj usługi hostingowej, ale zapewnia najwięcej mocy obliczeniowej i przestrzeni dyskowej. Hosting dedykowany jest odpowiedni dla dużych stron internetowych o dużym ruchu.

Hosting chmurowy

Rodzaj usługi hostingowej, w której strona internetowa jest hostowana na kilku serwerach w chmurze. Dzięki temu możliwe jest szybkie skalowanie mocy obliczeniowej i przestrzeni dyskowej w zależności od potrzeb strony internetowej. Hosting chmury jest odpowiedni dla stron internetowych o dużym i zmiennym ruchu.

Inne

Oprócz powyższych rodzajów usług hostingowych istnieją również inne, takie jak hosting plików, hosting e-mail i hosting aplikacji. Ważne jest, aby dobrać odpowiedni rodzaj usługi hostingowej do potrzeb strony internetowej.

8. Domeny

Domena internetowa to nazwa, pod którą można odwiedzić witrynę internetową lub usługi online. Domeny są używane do tworzenia adresów URL, które są używane do dostępu do stron internetowych.

Domeny składają się z etykiet, które rozdzielane są kropkami. Domeny internetowe składają się z nazwy głównej oraz końcówki/rozszerzenia. Końcówka domeny (np. *com*, *pl*, *org*) nazywana jest też domeną najwyższego poziomu (skrót z ang. *TLD* od *Top Level Domain*).

Domeny można rejestrować u usługodawców świadczących usługi rejestracji domen. Wiąże się z koniecznością wniesienia opłaty za okres dzierżawy. Standardowym okresem jest 1 rok, ale może być dłuższy np. kilkuletni.

Domeny nie możemy wykupić na stałe. Możemy jedynie otrzymać prawo do korzystania z danej domeny na ustalony okres, łącznie z prawem przedłużenia tego okresu.

Brak wniesienia opłaty za kolejny okres wiąże się z przekazaniem domeny do puli domen dostępnych do rejestracji.

Sub domeny

Mając zarejestrowaną domenę, możemy za darmo tworzyć dowolną ilość sub domen, czyli etykiet kolejnego poziomu.

Przykładowo dzierżawiąc domenę `domena.pl`, możemy utworzyć dowolną ilość nowych adresów typu

```
subdomena.domena.pl.
```

Dodatkowe linki

- What is a domain name? | Domain name vs. URL | Cloudflare
- What is a Domain Name? - Learn web development | MDN

9. Programowanie

Programowanie to proces tworzenia i implementacji kodu informatycznego, który określa sposób działania komputera lub innego urządzenia elektronicznego.

Programy komputerowe są pisane w różnych językach programowania, które umożliwiają zapisywanie instrukcji w formie, którą komputer jest w stanie zrozumieć i wykonać.

Warstwy abstrakcji

Procesory w typowych komputerach są w stanie wykonywać operacje matematyczne wyłącznie w systemie binarnych tj. na zerach i jedynkach. Działania te są możliwe dzięki algebrze Boole’a.

Języki programowania wprowadzając warstwy abstrakcji, ułatwiające programowanie działania komputerów. Zamiast pisać zera i jedynki, możemy posługiwać się łatwiejszym do zrozumienia dla nas kodem.

Finalnie wszystko co wprowadzamy do komputera lub z niego odczytujemy (teksty, dźwięki, obrazy, gesty, itp.) musi być zamienione na ciąg zer i jedynek, umownie reprezentujących konkretny rodzaj danych.

System binarny

System binarny to system liczbowy, w którym liczby są zapisywane za pomocą dwóch cyfr: 0 i 1.

Jest to system używany przez komputery, ponieważ urządzenia elektroniczne mogą rozpoznawać jedynie dwa stany przepływu prądu: “włączony” lub “wyłączony”.

System binarny jest więc idealnym rozwiązaniem do użytku w komputerach. Umożliwia on przetwarzanie informacji w sposób bardzo prosty i szybki.

Przykładowo, w systemie binarnym (dwójkowym), ciąg `101` można zapisać za pomocą równania
`1*2^2 + 0*2^1 + 1*2^0 = 4 + 0 + 1`

W systemie dziesiętnym `5` można zapisać analogicznie, używając jako podstawy potęgowania liczby 10 (system dziesiętny.
`5*10^0`

Struktury danych

Jedną z warstw abstrakcji wprowadzaną w programowaniu są struktury danych.

Za pomocą struktur możemy łatwo przechowywać zbiory danych oraz wykonywać na nich operacje.

Przykładowo, połączenia ludzi w serwisach społecznościowych, czy punktów trasy na mapie możemy przedstawić za pomocą struktury grafowej.

Listę zadań do zrobienia możemy ująć w formie tablicy (ang. *array*), listy, a jeśli zależy nam na zachowaniu kolejności to w formie listy łańcuchowej (linkowanej), stosu lub kolejki.

Strukturę podstron, czy hierarchię stanowisk pracy, możemy przechowywać w strukturze drzewa.

Każda ze struktur danych jest przeznaczona do przechowywania danych w sposób umożliwiający szybki dostęp do nich oraz wykonywanie różnych operacji na nich, takich jak wyszukiwanie, sortowanie i usuwanie.

Algorytmy

Algorytm to po prostu opis kroków, które należy wykonać, aby osiągnąć określony cel. Algorytmy są często stosowane w informatyce do rozwiązywania różnych problemów, ale mogą być również używane w innych dziedzinach, takich jak matematyka, nauki przyrodnicze i inżynieria.

Algorytmy są ważne, ponieważ pozwalają nam na rozwiązywanie problemów w sposób systematyczny i logiczny. Dzięki algorytmom możemy również wyjaśnić innym osobom, jak coś zrobić, a także umożliwić komputerom wykonanie zadania za pomocą kodu.

Aby być skutecznym, algorytm musi być dobrze zaprojektowany i musi być jasno opisany. Musi również być wystarczająco szczegółowy, aby można go było zaimplementować w kodzie informatycznym lub wykonać ręcznie. Ważne jest również, aby algorytm był efektywny i szybki w działaniu oraz aby zużywał niewielką ilość zasobów, takich jak pamięć lub moc obliczeniowa.

Notacja dużego O (Big O)

Notacja Big O to sposób opisywania, jak szybko rośnie liczba operacji wykonywanych przez algorytm wraz ze wzrostem rozmiaru danych wejściowych. Jest to ważne, ponieważ pozwala nam ocenić, jak efektywny jest dany algorytm w przetwarzaniu dużych ilości danych.

Big O używamy do określenia górnej granicy tego, ile operacji będzie wymagane dla najgorszego możliwego przypadku. Na przykład, jeśli mówimy, że algorytm ma złożoność O(n), oznacza to, że liczba operacji wykonywanych przez algorytm rośnie liniowo wraz ze wzrostem rozmiaru danych wejściowych.

Inne przykłady notacji Big O to:

- O(1) - stała złożoność, oznaczająca, że liczba operacji jest niezależna od rozmiaru danych wejściowych
- O(log n) - złożoność logarytmiczna, oznaczająca, że liczba operacji rośnie wolniej niż liniowo wraz ze wzrostem rozmiaru danych wejściowych
- O(n^2) - złożoność kwadratowa, oznaczająca, że liczba operacji rośnie wykładniczo wraz ze wzrostem rozmiaru danych wejściowych

Notacja Big O jest używana do porównywania różnych algorytmów i określania, który z nich jest najlepszy w danym zastosowaniu.

Algorytmy o niższej złożoności Big O są zazwyczaj bardziej wydajne niż algorytmy o wyższej złożoności, chociaż inne czynniki, takie jak pamięć i inne zasoby, również mogą mieć wpływ na wydajność algorytmu.

Wzorce projektowe

Innym z rodzajów abstrakcji używanych podczas programowania są wzorce projektowe. Umożliwiają one strukturyzowanie kodu według określonych zasad, ułatwiając tym samym utrzymywanie oraz rozbudowę kodu.

Przykładowo, łącząc dwa systemy, jeden który odbiera dane, a drugi który dane wysyła, możemy skorzystać z wzorca projektowego typu *adapter*. Dzięki niemu nie splątamy sposobu korzystania z przychodzących danych w systemie odbierającym i będziemy mogli w łatwy sposób przenieść się na innym system wysyłający dane, bez drastycznego przebudowywania systemu odbierającego.

Dodatkowe linki

- System binarny
- Podstawy systemu binarnego
- Algebra Boole'a – Wikipedia, wolna encyklopedia
- Computer Science Roadmap: Curriculum for the self taught developer
- Big O Cheat Sheet – Time Complexity Chart
- Time Complexity – Big O Notation Course

10. Formatowanie kodu

Formatowanie kodu to proces uporządkowania kodu informatycznego tak, aby był on łatwiejszy do czytania i zrozumienia dla ludzi. Formatowanie obejmuje takie rzeczy jak dodawanie odstępów i wcięć, aby zwiększyć czytelność kodu oraz używanie standardowych konwencji nazewnictwa, aby ujednolicić styl kodowania.

Dobrze sformatowany kod jest łatwiejszy do czytania i zrozumienia, co ułatwia pracę zespołową i umożliwia łatwiejsze debugowanie i utrzymanie kodu. Większość języków programowania ma swoje własne zalecenia dotyczące formatowania kodu, które należy przestrzegać, aby zapewnić czytelność i spójność kodu.

Formatowanie może być ręcznie lub automatycznie. Większość edytorów kodu posiada funkcję automatycznego formatowania, która może sformatować kod zgodnie z zaleceniami. Można również użyć specjalnych narzędzi do formatowania kodu, jak np. *Prettier*, czy *ESLint*, aby zapewnić spójność formatowania w obrębie całego projektu.

Dodatkowe linki

- Prettier · Opinionated Code Formatter
- ESLint - Pluggable JavaScript Linter

11. Debugowanie i testowanie

Debugowanie to proces usuwania błędów (tzw. “bugów”) z kodu informatycznego. Błędy mogą prowadzić do nieoczekiwanego zachowania programu lub do tego, że program w ogóle nie działa. Debugowanie jest ważnym elementem procesu tworzenia oprogramowania, ponieważ pozwala na naprawienie błędów i zapewnienie, że program działa prawidłowo.

Istnieje wiele różnych metod i narzędzi do debugowania kodu. Może to obejmować czytanie kodu i szukanie błędów, używanie debuggera, który pozwala na śledzenie wykonania kodu i sprawdzanie wartości zmiennych, oraz testowanie programu z różnymi danymi wejściowymi, aby sprawdzić, jak program sobie z nimi radzi.

Debugowanie może być czasochłonne i trudne, szczególnie w przypadku bardziej skomplikowanych programów. Wymaga to cierpliwości, dokładności i zdolności do rozwiązywania problemów. Jednak zapewnienie, że program działa poprawnie, jest ważne dla jego jakości i niezawodności.

Metoda gumowej kaczki

Metoda debugowania za pomocą gumowej kaczki (ang. *rubber duck debugging*) to sposób na rozwiązywanie problemów z kodem informatycznym, w którym opisujemy swoje myśli i rozwiązania problemu głośno, tak jakbyśmy to robili dla gumowej kaczki lub innej zabawki.

Idea polega na tym, że mówienie o tym, co robimy i dlaczego, pomaga nam lepiej zrozumieć problem i znaleźć rozwiązanie. Może to również pomóc w identyfikacji błędów w naszym myśleniu lub w naszym rozumowaniu, które mogą prowadzić do błędów w kodzie.

Metoda ta jest szczególnie przydatna, gdy pracujemy samodzielnie i nie mamy nikogo, z kim moglibyśmy porozmawiać o problemie. Możemy po prostu opisać swoje myśli gumowej kaczce i mówić o tym, co robimy i dlaczego. Bardzo często pomaga to w znalezieniu rozwiązania i naprawieniu błędów w kodzie.

Podobny efekt można uzyskać opisując problem np. na forum dyskusyjnym. Dokładne opisanie problemu, tak by inni go zrozumieli, często potrafi pomóc w lepszym zrozumieniu istoty problemu. Nie jest niespodzianką, gdy jeszcze przed opublikowaniem postu na forum znajdujemy rozwiązanie naszego problemu.

Dodatkowe linki

- What is Debugging? How to Debug Your Code for Beginners
- Metoda gumowej kaczuszki – Wikipedia, wolna encyklopedia

12. Kontrola wersji

System kontroli wersji (ang. *Version Control System*, skrót *VCS*) to oprogramowanie, które służy do zarządzania zmianami w kodzie informatycznym i dochowywania historii zmian. Pozwala on zachować wersje kodu i śledzić zmiany wprowadzane przez różne osoby w czasie.

System kontroli wersji jest bardzo przydatny w pracy zespołowej, ponieważ pozwala na współpracę nad tym samym kodem i unikanie konfliktów. Może również służyć do tworzenia wersji rozwojowych i produkcyjnych kodu oraz do łatwego powrotu do poprzednich wersji w razie potrzeby.

Istnieje wiele różnych systemów kontroli wersji, takich jak Git, Subversion, Mercurial i CVS. Każdy z nich ma swoje własne cechy i sposoby działania, ale wszystkie służą do tego samego celu: zarządzania zmianami w kodzie i dochowywania historii zmian.

Wersjonowanie

W celu ułatwienia zrozumienia typu wprowadzanych zmian w oprogramowaniu, stosuje się wersjonowanie.

Popularnym schematem wersjonowanie jest zwiększanie numeracji w obrębie 3 oddzielonych od siebie kropkami wartości.

Standardowy schemat wersji zapisywany jest za pomocą 3 liczb, `major.minor.patch`.

Major oznacza wersję główną, *minor* oznacza wersję drugorzędną/mniejszą, a *patch* wersję łaty.

Zwiększenie nr wersji `major` oznacza, że w oprogramowaniu wystąpiły zmiany, które nie są kompatybilne z poprzednimi wersjami `major`. Uaktualnienie oprogramowania w którym zmieniła się wersja `major` oznacza, że sposób w jaki do tej pory używaliśmy oprogramowania został znacząco zmieniony.

Jeśli korzystamy z zewnętrznej biblioteki w naszym oprogramowaniu i ta biblioteka została wydana w nowej wersji `major`, to zaktualizowanie jej najprawdopodobniej doprowadzi do zepsucia działania naszego oprogramowania.

Przed aktualizacją oprogramowania, które jest zależnością naszego oprogramowania musimy uważać, aby nie zaktualizować go do nowej wersji `major` bez uprzedniego zapoznania się z listą zmian (tzw. `breaking changes`) i odpowiedniego zaadresowania tych zmian w naszym oprogramowaniu.

Wersja `minor` oznacza aktualizacje usprawniające działanie oprogramowania. W aktualizacji tej wersji mogą być dodane np. nowe funkcjonalności lub ulepszone zostało działanie obecnych. Aktualizacja wersji `minor` oznacza jednak, że nie zostały wprowadzone żadne zmiany, które naruszałyby kompatybilność wsteczną tj. nie jest wymagane od nas żadne dodatkowe działanie w celu zapewnienia poprawności działania naszego oprogramowania, wykorzystującego zaktualizowaną wersję oprogramowania zależnego.

Wersja `patch` oznacza wprowadzenie poprawek czyli załatanie błędów w istniejącym oprogramowaniu. Podobnie jak wersja `minor`, zmiana nr tej wersji oznacza, że nie zostały

wprowadzone żadne drastyczne zmiany, a jedynie naprawione zostały błędy w aktualnej wersji `major.minor`.

Przykładowo, tworząc oprogramowanie i wypuszczając je w formie dostępnej do użytku przez innych zazwyczaj rozpoczynamy wersjonowanie od wersji *major* `0`, *minor* `1`, *patch* `0`. Pierwsza ogólnodostępna wersja oprogramowania będzie więc oznaczona nr `0.1.0`. Możemy też pominąć wersję *patch* i zapisać nr wersji w takim przypadku jako `0.1`.

Wprowadzając nowe ulepszenia, które są kompatybilne z poprzednim działaniem programu, podbijamy wersję *minor*, wypuszczając tym samym kolejną wersję oprogramowania z nr `0.2`.

Jeśli wykryjemy jakiś błąd w wersji `0.2` to wypuszczamy nową wersję z łatą, którą opatrujemy nr `0.2.1`.

Jeśli ten sam błąd występował w wersji `0.1` to możemy również naprawić go w tamtej wersji oprogramowania i wypuścić wersję `0.1.1`. Nie jest to niezbędne, chyba że wyraźnie zakomunikowaliśmy świadczenie długoterminowego wsparcia dla wersji `0.1`, tzw. *LTS* (z ang. *Long-Term Support*).

Gdy wprowadzimy jakąś przełomową zmianę, która zmieni działanie oprogramowania, tak, że nie będzie można go już używać w sposób zgodny z poprzednią wersją, wypuszczamy wersję `1.0`.

Systemy kontroli wersji (jak np. Git), umożliwiają łatwe przełączanie się pomiędzy wersjami kodu.

Wykrywając błąd np. w wersji `5.11.4` możemy odtworzyć cała bazę naszego kodu z momentu, w którym

wypuszczaliśmy tą wersję, a następnie dokonać poprawek i wypuścić wersję `5.11.4`.

Aktualna wersja oprogramowania może być zupełnie inna, np. `6.1` i nie zawierać już błędu, który był w wersji `5.11`, z uwagi na przełomowe zmiany, które wprowadziliśmy między wersją `5.x.` a `6.x.`.

Git

Git to system kontroli wersji, który umożliwia zarządzanie zmianami w plikach i śledzenie historii zmian w projektach. Pozwala on również na tworzenie kopi zapasowych plików i zarządzanie wieloma wersjami projektu jednocześnie. Git jest szczególnie przydatny w przypadku projektów oprogramowania, ponieważ pozwala wielu ludziom współpracować nad tym samym projektem jednocześnie i łatwo integrować ich wkład. Git jest obecnie jednym z najczęściej używanych systemów kontroli wersji na świecie.

Inne systemy kontroli wersji

Git to nie jedyny system kontroli wersji. Jest jednak najbardziej popularnym rozwiązaniem.

Innymi systemami kontroli wersji jest np. Subversion (SVN), Mercurial, lub Piper (używany przez Google).

GitHub, GitLab, Bitbucket

GitHub, GitLab, Bitbucket (oraz inne platformy tego typu) są oparte na systemie kontroli wersji Git. Umożliwiają scentralizowane przechowywanie i śledzenie zmian w plikach oraz łatwą współpracę nad projektami zespołowymi.

Serwisy tego typu są często używane do hostowania i zarządzania projektami oprogramowania, ale mogą być również używane do zarządzania dowolnym rodzajem plików (głównie tekstowych) i projektów.

GitHub, GitLab czy Bitbucket oferują również narzędzia do zarządzania zadaniami, bugami, zgłoszeniami, czy nawet do tworzenia sekwencji zdarzeń, które mają nastąpić w momencie wprowadzenia zmian w projekcie (jak np. przetestowanie wprowadzonych zmian).

Rewizja kodu (Code review)

Popularną praktyką podczas korzystania z systemu kontroli wersji jest tzw. *code review*.

Wspomniane wcześniej scentralizowane serwisy pozwalają na łatwą rewizję kodu przez innych programistów, przed ostatecznym wprowadzeniem zmian w kodzie do projektu.

Celem *code review* jest poprawienie jakości i wychwytywanie błędów w kodzie, zanim trafią do oprogramowania

W ramach procesu rewizji kodu, inni programiści mogą przeglądać i komentować kod, aby zapewnić, że jest on zgodny z wymaganiami i dobrymi praktykami programistycznymi.

Dodatkowe linki

- Czym jest Git?
- GitHub Git Ściąga - GitHub Cheatsheets
- Git
- Git - Book

- Version control - Wikipedia
- System kontroli wersji – Wikipedia, wolna encyklopedia
- Semantic Versioning 2.0.0 | Semantic Versioning
- Code review - Wikipedia
- Subversion – Wikipedia, wolna encyklopedia

13. Bezpieczeństwo

Dbanie o bezpieczeństwo w informatyce to proces zapewnienia, że dane i systemy informatyczne są chronione przed nieautoryzowanym dostępem, użyciem, zniszczeniem lub modyfikacją. Obejmuje to różne działania, takie jak zapewnianie zabezpieczeń sieciowych, instalowanie oprogramowania antywirusowego, tworzenie kopi zapasowych danych, stosowanie silnych haseł i ograniczenie dostępu do danych i systemów tylko upoważnionym osobom.

Bezpieczeństwo informatyczne jest ważne w każdej organizacji, ponieważ chroni dane i systemy przed zagrożeniami zewnętrznymi i wewnętrznymi, takimi jak hakerzy, wirusy i błędy ludzkie.

Brak odpowiednich zabezpieczeń i niedbałość w kwestii tworzenia oprogramowania może prowadzić np. do wycieku danych użytkowników. Sprawy tego typu są rozpatrywane indywidualnie, ale mogą narażać organizację na kary finansowe.

Innym z zagrożeń może być np. zaszyfrowanie ważnych danych i udostępnienie klucza odszyfrowującego po dokonaniu opłaty. Przypomina to fizyczne porwanie. W świecie cyfrowym zakładnikiem stają się dane, które zostają uwolnione po zapłaceniu okupu.

Dodatkowe linki

- Web security | MDN

- OWASP Top 10:2021
- Safe and secure
- SecLists/Passwords at master · danielmiessler/SecLists
- Ransomware – Wikipedia, wolna encyklopedia

14. Wydajność

Wydajność w informatyce oznacza szybkość i efektywność działania systemów informatycznych i oprogramowania. Może być mierzona za pomocą różnych mierników, takich jak czas działania, ilość przetwarzanych danych lub liczba obsługiwanych użytkowników.

Wydajność jest ważna, ponieważ lepsza wydajność oznacza szybsze i bardziej niezawodne działanie systemów, co z kolei może prowadzić do większej wydajności i efektywności organizacji.

Istnieje wiele sposobów poprawy wydajności systemów informatycznych, takich jak optymalizacja oprogramowania, rozszerzenie zasobów sprzętowych (np. poprzez dodanie więcej pamięci RAM lub szybszych dysków twardych) lub zmiana sposobu korzystania z systemu (np. poprzez ograniczenie liczby wykonywanych jednocześnie zadań). Wydajność jest ważnym aspektem zarządzania systemami informatycznymi i jest często brana pod uwagę przy podejmowaniu decyzji dotyczących oprogramowania i sprzętu.

Dodatkowe linki

- PageSpeed Insights
- Web performance | MDN
- Measuring performance - Learn web development | MDN
- Optimization and performance - Developer guides | MDN
- Metrics
- Fast load times

- Measure page quality

15. Licencjonowanie

Licencjonowanie oprogramowania to sposób umożliwienia użytkownikom legalnego korzystania z oprogramowania. W ramach licencji użytkownik otrzymuje prawo do instalowania i korzystania z oprogramowania na określonych warunkach. Licencje mogą być różne i mogą obejmować różne ograniczenia w zakresie korzystania z oprogramowania. Na przykład licencja może pozwalać na instalowanie oprogramowania tylko na jednym komputerze lub udostępniać oprogramowanie tylko na określony okres czasu.

Głównym celem licencjonowania oprogramowania jest zapewnienie twórcom oprogramowania wynagrodzenia za ich pracę i umożliwienie im kontynuowania rozwoju oprogramowania. Licencjonowanie oprogramowania również pomaga chronić prawa autorskie do oprogramowania i zapobiega nielegalnemu rozpowszechnianiu i kopiowaniu oprogramowania.

Otwarte oprogramowanie

Wiele organizacji decyduje się na udostępnienie źródła swojego oprogramowania z użyciem licencji pozwalającej na dowolne modyfikowanie i rozpowszechnianie oprogramowania. Zachęca to innych programistów do dobrowolnego rozwijania oprogramowania dopasowując go do własnych potrzeb oraz naprawiając błędne funkcjonowanie.

Otwarte oprogramowanie (możliwość odczytu kodu źródłowego) nie oznacza automatycznie, że kod można kopiować, dowolnie modyfikować, czy rozpowszechniać. Podobnie jak w przypadku innych prac autorskich (jak np. zdjęcia), trzeba upewnić się, że licencja pozawala nam na dany użytek.

Dodatkowe linki

- Otwarte oprogramowanie – Wikipedia, wolna encyklopedia
- Licencja MIT – Wikipedia, wolna encyklopedia

16. Aplikacje mobilne

Aplikacje mobilne (zwane również aplikacjami na telefon lub po prostu aplikacjami) to programy, które można zainstalować i uruchomić na urządzeniach mobilnych, takich jak smartfony lub tablety. Aplikacje te służą do różnych celów, takich jak komunikacja, rozrywka, zakupy, bankowość i wiele innych. Można je pobierać z internetu za pomocą specjalnych sklepów z aplikacjami, takich jak App Store dla urządzeń z systemem iOS lub Google Play dla urządzeń z systemem Android.

Ich przewagą nad aplikacjami uruchamianymi z poziomu przeglądarki jest możliwość korzystania z natywnych funkcjonalności urządzenia jeśli urządzenie takowe posiada np. dostęp do GPSa, żyroskopu, kamery, mikrofonu, odczyt lub wysyłanie wiadomości sms, wykonywania połączeń, odczyt lub zapis kontaktów w książce adresowej , itp.

PWA

Specyficznym typem aplikacji mobilnych są aplikacje typu PWA (skrót z ang. od *Progressive Web Application*).

PWA umożliwiają utworzenie aplikacji dostępnej z poziomu przeglądarki, ale jednocześnie z możliwościami aplikacji mobilnej. PWA mogą być używane w trybie offline (bez dostępu do internetu), mogą być zapisane na ekranie urządzenia w formie osobnej ikony, otwierając tym samym aplikację w trybie aplikacji, bez interfejsu przeglądarki (jak np. pasek adresu).

Aplikacje typu desktop

Innym wariantem aplikacji, o którym warto wspomnieć, są aplikacje uruchamiane bezpośrednio w systemach operacyjnych typu desktop.

Zasada tworzenia oprogramowania w takim przypadku jest taka sama jak w przypadku urządzeń mobilnych. Tworząc oprogramowanie typu desktop musimy umożliwić wykonanie programu bezpośrednio przez system operacyjny.

Narzędzia typu *Electron* pozwalają tworzyć aplikacje desktopowe za pomocą technologi webowych, tworząc aplikację która będzie uruchamiana w systemie operacyjnym za pomocą przeglądarki internetowej. Interfejs przeglądarki zostaje jedynak ukryty i finalnie wygląd aplikacji nie wskazuje na to, że jest to aplikacja działająca wewnątrz przeglądarki.

Dodatkowe linki

- Mobile Web Development - Developer guides | MDN
- Progressive Web Apps
- What are Progressive Web Apps?
- Progressive web apps (PWAs) | MDN
- Learn PWA
- PWA audits
- React Native · Learn once, write anywhere
- Build cross-platform desktop apps with JavaScript, HTML, and CSS | Electron

Frontend

17. HTML

HTML (Hypertext Markup Language) to język znaczników, który służy do tworzenia stron internetowych. HTML pozwala na tworzenie struktury i formatowanie treści strony internetowej, takiej jak nagłówki, akapity, listy, linki, obrazy i inne elementy.

HTML składa się z zestawu znaczników, które są używane do oznaczenia różnych elementów na stronie. Te znaczniki są zawarte w nawiasach kątowych, np. `<html>`, `<head>`, `<body>`, `<p>`, `<img>`.

W połączeniu z innymi językami, takimi jak CSS i JavaScript, HTML jest używany do tworzenia dynamicznych i interaktywnych stron internetowych.

Semantyka

Kiedyś strony były budowane w oparciu o tabele (znaczniki typu `<table>`,`<tr>`,`<td>`). To idealny przykład używania niewłaściwej semantyki, która za pomocą znaczników, błędnie określa treść dokumentu.

Z technicznego punktu widzenia mógłbyś w swoim kodzie używać wyłącznie `div`ów i `span`ów i byłbyś w stanie wizualnie stworzyć każdy rodzaj elementów na stronie. Ale nie rób tego. Używaj znaczników semantycznych, opisujących zawartość i przeznaczenie elementu.

Do celów **wyłącznie** aplikowania styli CSS (np. podział układu strony na kolumny, osiągnięcie pożądanych

wizualnie efektów) możesz oczywiście korzystać z elementów `div` i `span`.

Właściwa semantyka ma również ogromne znaczenie w kwesti SEO (z ang. *Search Engine Optimization*), czyli w optymalizacji strony pod kątem indeksowania jej przez silniki wyszukiwarek (np. Google).

Dodatkowe linki

- HTML | Frontend | Web Development | Panel Nauki | Codisity
- Introduction to HTML - Learn web development | MDN
- Tutorials | MDN
- Learn HTML
- Semantics - MDN Web Docs Glossary: Definitions of Web-related terms | MDN
- hail2u/html-best-practices: For writing maintainable and scalable HTML documents

18. CSS

CSS (skrót z ang. *Cascading Style Sheets*) to język służący do opisywania wyglądu stron internetowych. CSS pozwala na określenie takich rzeczy jak kolory, rozmiary, położenie i inne aspekty wizualne elementów na stronie. Dzięki CSS, można oddzielić wygląd strony od jej treści, co pozwala na łatwiejsze zarządzanie i modyfikowanie wyglądu strony.

CSS składa się z reguł, które składają się z selektora (elementów HTML, które chcemy zmienić) i deklaracji (specyfikujące zmiany jakie chcemy wprowadzić). Przykład:

```
p {
  color: blue;
  font-size: 16px;
}
```

Reguła ta mówi, że wszystkie elementy HTML oznaczone jako `<p>` będą miały niebieski kolor i rozmiar czcionki 16px.

CSS umożliwia także kaskadowe (ang. *cascading*) dziedziczenie stylów. CSS pozwala na dziedziczenie stylów od rodzica do dziecka, co pozwala na łatwiejsze zarządzanie i unikanie powielania kodu.

Dodatkowe linki

- CSS | Frontend | Web Development | Panel Nauki | Codisity
- Learn to style HTML using CSS - Learn web development | MDN
- CSS Tutorials | MDN

- CSS Layout - Developer guides | MDN
- Using CSS animations - CSS: Cascading Style Sheets | MDN
- Learn Responsive Design

19. JavaScript

JavaScript to język skryptowy, który jest powszechnie używany do tworzenia interaktywnych i dynamicznych stron internetowych. JavaScript pozwala na dodawanie interaktywności do stron, np.:

- obsługę zdarzeń (kliknięcia przycisków i wprowadzanie danych)
- dynamiczne zmienianie zawartości strony (na podstawie danych z serwera lub lokalnych np. ciastek w przeglądarce)
- tworzenie efektów animacji (płynne przejścia pomiędzy podstronami)
- walidację formularzy (wyświetlanie błędów przy nieprawidłowo wypełnionych polach)

JavaScript jest językiem skryptowym, co oznacza, że jest on interpretowany w momencie wykonywania, a nie kompilowany. Oznacza to, że kod łatwo modyfikować i testować bez potrzeby ponownego kompilowania całej aplikacji.

JavaScript jest często używany w połączeniu z HTML i CSS, tworząc trójwarstwowy model:

- HTML jako warstwa danych
- CSS jako warstwa prezentacji
- JavaScript jako warstwa logiki

Pozwala to na tworzenie zaawansowanych, interaktywnych stron internetowych i aplikacji oraz ułatwia podział kodu ze względu na jego przeznaczenie użycia.

Różnica między językami Java i JavaScript

Jeśli zaczynasz dopiero przygodę z językami programowania, możesz wpaść w pułapkę związaną z nazwami podobnymi nazwami 2 popularnych języków: Java i JavaScript.

Mimo, że ich nazwy są podobne, to są to dwa, totalnie różne języki. Ich nazewnictwo może być mylące i warto zapamiętać, że **Java** ma się do **JavaScript** tak samo jak np. **Kot** do **Kotliny**.

EcmaScript

ECMAScript (zwany również ES) to standard języka skryptowego, na którym oparty jest JavaScript. ECMAScript został opracowany przez organizację Ecma International i pierwotnie opublikowany w 1997 roku.

ECMAScript jest specyfikacją języka skryptowego, która zawiera szczegółowe zasady składni i semantyki języka. JavaScript jest jednym z implementacji ECMAScript, ale istnieją też inne języki, takie jak ActionScript i JScript, które również są implementacjami ECMAScript.

Nowe wersje ECMAScript są publikowane co kilka lat i zawierają nowe funkcjonalności oraz poprawki do istniejącego języka. Na przykład, ECMAScript 2015 (również znany jako ES6) wprowadził nowe konstrukcje językowe, takie jak klasy i symbole, a ECMAScript 2016 (ES7) wprowadził np. operator *exponentiation* (obliczanie potęgi).

Numeracja wersji ECMAScript może być myląca. ES6 oznacza wersję 6 specyfikacji, która została wydana w roku

2015. ES7 to wersja 7 wydana w roku 2016, wersja 13 to wersja ECMAScript z roku 2022.

Node.js

JavaScript może być uruchamiany nie tylko w środowisku przeglądarki, ale także bezpośrednio w systemie operacyjnym. Popularnym środowiskiem uruchomieniowym JavaScript w systemie operacyjnym jest Node.js. Nie jest to jednak jedyne rozwiązanie. Istnieją również inne jak np. Deno lub Bun.

Dodatkowe linki

- JavaScript — Dynamic client-side scripting - Learn web development | MDN
- JavaScript Tutorials | MDN
- ECMA-262 - Ecma International
- Node.js — Run JavaScript Everywhere
- Deno — A modern runtime for JavaScript and TypeScript
- Bun — fast all-in-one JavaScript runtime

20. Dostępność

Dostępność (z ang. *accessibility*), oznacza projektowanie stron internetowych, aplikacji oraz urządzeń, aby były one dostępne dla jak największej liczby osób, w tym dla osób z różnymi niepełnosprawnościami. Celem projektowania zgodnie z zasadami dostępności jest umożliwienie wszystkim osobom swobodnego korzystania z treści i funkcji strony internetowej, bez względu na ich możliwości fizyczne, technologiczne czy sensoryczne.

Dostępność oznacza projektowanie tak, aby strony internetowe były zrozumiałe i umożliwiały korzystanie z nich przez osoby niedowidzące korzystających z czytników ekranu (np. udostępniając alternatywne opisy dla obrazów). Istotne jest również aby strony umożliwiały nawigację za pomocą klawiatury, a także aby były przyjazne dla użytkowników z różnymi formami dysleksji, czy innymi trudnościami z czytaniem, jak również dla osób z trudnościami w rozpoznawaniu barw.

Uwzględnienie dostępności przy projektowaniu stron internetowych jest ważne, ponieważ pozwala na umożliwienie dostępu do informacji i usług online dla jak największej liczby osób. W niektórych krajach istnieją przepisy prawne wymagające od stron internetowych spełnienia określonych wymagań dotyczących dostępności.

Słowo *accessibility* zapisuje się również w formie skrótu *a11y* (*a*, *11 liter*, *y*), wymawianego jako *alli* (*1* czytane są jako litery *L*).

Dodatkowe linki

- Accessibility | MDN
- Web Content Accessibility Guidelines (WCAG) 2.1
- Accessibility Principles | Web Accessibility Initiative (WAI) | W3C
- WebAIM: Web Accessibility In Mind
- Learn Accessibility
- Accessible to all

21. Biblioteki i frameworki

Biblioteki i frameworki to gotowe moduły kodu, które programiści mogą wykorzystać w swoich projektach, aby ułatwić i przyspieszyć proces tworzenia oprogramowania.

Biblioteki to kolekcje funkcji, procedur lub klas, które programiści mogą wykorzystać do rozwiązania określonych problemów lub zadań. Przykładem biblioteki w języku JavaScript może być React, który udostępnia funkcje do łatwego budowania interfejsów graficznych. Biblioteki nie narzucają sposobu ich wykorzystania. Mogą być dodawane do istniejących projektów w celu rozwiązania konkretnego problemu.

Frameworki to bardziej zaawansowane narzędzia, które zawierają w sobie zestaw bibliotek oraz określone modele i wzorce projektowe, czy narzędzia do tworzenia i testowania aplikacji. Frameworki narzucają programiście szkielet, na którym musi on budować swój projekt. Zapewniają dzięki temu jednak spójność i przewidywalność kodu niezależnie od projektu. Przykładem frameworka w języku JavaScript może być np. Angular. Mocno narzuca on określoną strukturę i sposób tworzenia projektu w przeciwieństwie do biblioteki takiej jak React.

Zarówno biblioteki jak i frameworki oferują wielokrotny użytek gotowego kodu, dzięki czemu programiści nie muszą od podstaw tworzyć wszystkich elementów aplikacji, co pozwala im skupić się na rozwiązywaniu problemów biznesowych i dodawaniu nowych funkcjonalności.

Popularnym menadżerem bibliotek, umożliwiającym wyszukiwanie i instalowanie ich w ekosystemie JavaScript, jest *npm* (skrót z ang. od *Node Package Manager*).

Dodatkowe linki

- React – A JavaScript library for building user interfaces
- Angular
- npm

22. Web Components

Web Components to standardowy zestaw technologii, które pozwala na tworzenie własnych, niestandardowych elementów HTML, które mogą być używane wewnątrz strony internetowej.

Główne elementy składające się na Web Components to:

- Custom Elements - pozwala na tworzenie własnych tagów HTML, które mogą być używane jak standardowe tagi HTML
- Shadow DOM - pozwala na izolowanie stylów i struktury elementów, co pozwala na uniknięcie konfliktów między różnymi elementami na stronie.
- HTML Templates - pozwala na zdefiniowanie szablonów kodu HTML, które mogą być wykorzystywane wielokrotnie w różnych miejscach na stronie.

Dzięki Web Components, programiści mogą tworzyć własne, niestandardowe elementy, które są odizolowane od pozostałej części strony i mogą być używane jak standardowe elementy HTML. Dzięki temu, Web Components pozwalają na lepszą modułowość i reusability (możliwość powtórnego użycia) kodu.

Dodatkowe linki

- Web Components | MDN

23. WebAssembly (WASM)

WebAssembly (WASM) to nowy format pliku i standardowy kod maszynowy, który jest przeznaczony do użytku w przeglądarkach internetowych. Jest on zaprojektowany tak, aby umożliwić uruchamianie kodu napisanego w różnych językach programowania (np. C/C++, C#, Rust) na maszynie klienta z wysoką wydajnością.

WebAssembly jest wykonywany natywnie przez przeglądarki, co oznacza, że jest on wykonywany bezpośrednio przez procesor, zamiast być interpretowany przez JavaScript, co daje znacznie lepszą wydajność. WebAssembly pozwala więc na szybkie ładowanie i wykonywanie kodu. Technologia ta jest kompatybilna z istniejącym ekosystemem webowym, co oznacza, że WASM można używać z HTML, CSS i JavaScript.

W ostatnim czasie coraz więcej projektów zaczyna wykorzystywać WebAssembly, ponieważ oferuje większą wydajność, a także daje możliwość użycia istniejącego kodu backendowego bezpośrednio w przeglądarkach.

Dodatkowe linki

- WebAssembly | MDN

24. SEO

SEO (Search Engine Optimization) to proces optymalizacji strony internetowej i jej treści, aby była ona lepiej widoczna w wynikach wyszukiwania. Celem SEO jest zwiększenie organicznego ruchu na stronie poprzez poprawienie pozycji strony w wynikach wyszukiwania.

SEO składa się z wielu różnych elementów, np.:

- Optymalizacja tytułów i opisów stron
- Używanie odpowiednich słów kluczowych
- Tworzenie wartościowej treści
- Linkowanie wewnętrzne i zewnętrzne
- Optymalizacja kodu HTML
- Tworzenie mapy strony

Optymalizacja tytułów i opisów stron polega na tworzeniu atrakcyjnych i zwięzłych opisów treści strony, które będą odpowiadały na pytania potencjalnych użytkowników.

Używanie odpowiednich słów kluczowych polega na wybieraniu tych słów, które są najważniejsze dla treści strony i które będą używane przez użytkowników w wyszukiwarkach.

Tworzenie wartościowej treści oznacza tworzenie treści, która jest interesująca i przydatna dla użytkowników, a także odpowiada na ich potrzeby i pytania.

Linkowanie wewnętrzne i zewnętrzne polega na tworzeniu odnośników do strony z innych stron, co pomaga

wyszukiwarkom zrozumieć związek między stronami i ich zawartością.

Optymalizacja kodu HTML polega na uporządkowaniu kodu, tak aby był on czytelny dla wyszukiwarek, zawierał odpowiednio semantyczne elementy oraz oferował wysoką dostępność (np. poprzez tekstowe opisy umieszczonych na stronie zdjęć).

Tworzenie mapy strony wymaga stworzenia pliku, który zawiera spis treści wszystkich podstron. Niektóre podstrony mogą być niedostępne dla automatycznego indeksowania (polegającego na wchodzeniu w linki przez roboty indeksujące). Podstrony, do których nie linkują inne strony, oraz nie znajdują się w mapie strony, nie będą zaindeksowane przez wyszukiwarki.

Dodatkowe linki

- SEO - MDN Web Docs Glossary: Definitions of Web-related terms | MDN
- SEO audits

25. Hosting frontendu

Hostowanie projektów frontendowych wiąże się z hostowaniem plików statycznych (tj. takich, które otwierane są bezpośrednio przez przeglądarkę).

Hosting stron statycznych polega więc na przechowywaniu plików HTML, CSS, JavaScript i innych związanych z treścią strony na serwerze, która jest dostępny przez internet. Strony statyczne nie wymagają skomplikowanej infrastruktury po stronie serwera, ponieważ nie korzystają z bazy danych ani skryptów dynamicznych. Z tego powodu hosting stron statycznych jest często tańszy i bardziej niezawodny niż hosting stron dynamicznych, który wymaga bardziej zaawansowanej konfiguracji serwera.

Strony statyczne

Strony statyczne cechuje stałość dostępnych treści. Mimo, że strony statyczne mogą być generowane z użyciem treści dostępnych w systemach dynamicznych (np. CMSach), to finalnie wszystkie pliki strony statycznej będą zawierały już całą dostępną treść.

Strony statyczne nie umożliwiają więc warunkowego dostępu do treści np. poprzez system logowania i autentykacji użytkownika.

Generatory statycznych stron

Do generowania statycznych stron frontendowych używa się gotowych narzędzi, takich jak np. Next.js, Gatsby, Jekyll,

Hugo, czy Docusaurus.

Dodatkowe linki

- Static web page - Wikipedia
- What is a static site generator? | Cloudflare
- Static Site Generators - Top Open Source SSGs | Jamstack

Backend

26. Interfejs lini komend

CLI (skrót z ang. *Command Line Interface*) jest interfejsem użytkownika, który pozwala na wykonywanie poleceń przez za pomocą linii komend, zamiast za pomocą interfejsu graficznego. CLI jest często używany do zarządzania systemem operacyjnym, aplikacjami i urządzeniami sieciowymi, a także do automatyzacji zadań.

W web developmencie poprzez CLI może być uruchamiane wiele poleceń związanych z działaniem strony/aplikacji.

Oprogramowania takie jak np. bazy danych, oferują zestaw narzędzi do zarządzania bazami za pomocą CLI.

Frameworki oraz CMSy często oferują zestaw narzędzi CLI do automatyzacji powtarzalnych zadań jak np. tworzenie nowych elementów.

Używanie systemu kontroli wersji Git odbywa się najczęściej również za pomocą lini komend, czyli CLI.

Zarządzanie serwerem (np. VPS) również odbywa się za pomocą CLI.

JavaScript'owe biblioteki (uruchamiane za pomocą poleceń `npm` lub `npx`), również oferują zestaw narzędzi obsługiwanych za pomocą CLI.

Każdy składnik systemu, który ma graficzny interfejs (GUI), ma również swój zestaw poleceń CLI. Graficzny interfejs systemu operacyjnego to tylko program, który umożliwia "wyklikanie" opcji w sposób wizualny. Systemami operacyjnymi można jednak administrować bez graficznego

interfejsu i wykonywać wszystkie zadania używając bezpośrednio wiersza poleceń.

Dodatkowe linki

- Command line crash course - Learn web development | MDN

27. Języki programowania

Backendowe języki programowania służą do tworzenia skryptów, które pracują “za kulisami” aplikacji internetowych. Napisany w nich kod przetwarza dane, komunikuje się z bazami danych oraz tworzy logikę aplikacji. Języki te są niezbędne do działania aplikacji, ale nie są bezpośrednio widoczne dla użytkownika końcowego (użytkownik aplikacji nie może podejrzeć takiego kodu).

Przykłady języków backendowych to: Python, Ruby, Java, C#, JavaScript (z użyciem środowiska uruchomieniowego), PHP, Go itd.

Wraz z pojawieniem się technologi WebAssembly, możliwe stało się wykorzystywanie języków backendowych do pisania kodu uruchamianego bezpośrednio przez przeglądarki.

W przeciwieństwie do języku znaczników (jak np. HTML), języki programowania umożliwiają wprowadzenie logiki do działania aplikacji. Oznacza to, że możliwe jest tworzenie instrukcji warunkowych i uruchamianie wybranych fragmentów kodu wyłącznie jeśli spełnione zostaną określone warunki np. użytkownik będzie miał prawo dostępu do danych treści (np. panelu admina).

Backendowe języki programowania uruchamiane są na serwerze aplikacji, czyli bezpośrednio w systemie operacyjnym. Oznacza to, że możliwe staje się wywoływanie również poleceń systemowych takich jak np. wysyłka oraz odbieranie wiadomości email, odczyt i zapis plików (na serwerze), zbieranie i przetwarzanie danych, konwersja

multimediów (np. transkodowanie wideo), strumieniowanie danych (np. odtwarzanie filmów z serwera na żądanie), generowanie odpowiedzi z wytrenowanych sieci neuronowych, wydzielanie zasobów serwera (przestrzeni dyskowej, mocy obliczeniowej) poprzez tworzenie maszyn wirtualnych lub kontenerów.

W skrócie, wszystko co można zrobić na komputerze, można z użyciem backendowych języków programowania, zaprogramować tak, aby działało jako usługa aplikacji internetowej.

JavaScript i Node.js

Osobiście polecam wykorzystywanie środowiska Node.js do uruchamiania języka JavaScript po stronie serwera. Nie jest to oczywiście rozwiązanie idealne dla wszelkiego typu projektów. Używając innych bardziej nisko poziomowych języków jak np. Rust mamy możliwość budowania bardziej wydajnych aplikacji. Mówimy tu jednak o potrzebie wykonywania naprawdę bardzo złożonych operacji obliczeniowych.

Niskopoziomowe języki dają większą kontrolę nad tym w jaki sposób kod zostanie wykonany przez komputer. Możemy np. ręcznie alokować wybrane miejsca zapisu w pamięci RAM, czy uruchamiać jednocześnie różny kod w wielu wątkach procesora.

Jeśli jednak naszym celem nie jest wykonywanie bardzo złożonych obliczeń, używanie JavaScriptu po stronie serwera daje jedną bardzo dużą przewagę nad innymi językami. Ekosystem języka JavaScript sprawia, że znając ten język możemy tworzyć warstwy aplikacji zarówno po stronie

serwera (backend) jak i przeglądarki (frontend). Tworzenie fullstackowych aplikacji staje się więc znacznie prostsze.

Dodatkowo z pomocą języka JavaScript i dostępnych narzędzi (jak React Native, czy Electron), możemy tworzyć aplikacje na urządzenia mobilne oraz desktopowe. JS daje nam więc tak szerokie możliwości, jakich aktualnie nie oferuje żaden inny język.

Dodatkowe linki

- Node.js - MDN Web Docs Glossary: Definitions of Web-related terms | MDN
- Node.js Developer Roadmap: Learn to become a modern node.js developer

28. API

API (ang. Application Programming Interface) to interfejs programowania aplikacji, który pozwala na komunikację między różnymi systemami lub aplikacjami. API umożliwia udostępnianie danych lub funkcjonalności przez jedną aplikację innym aplikacjom, co pozwala na tworzenie nowych aplikacji lub rozszerzanie już istniejących. Dzięki API, aplikacje mogą wymieniać dane i wykonywać funkcje bez potrzeby bezpośredniego dostępu do kodu źródłowego.

Popularnymi rozwiązaniami architektury tworzenia API aplikacji internetowych są REST, GraphQL oraz RPC.

RESTful API

REST (Representational State Transfer) jest architekturą sieciową, która opiera się na protokole HTTP i pozwala na tworzenie rozproszonych systemów. RESTful API nazywany jest interfejs aplikacji webowej, który przestrzega zasad REST i pozwala na komunikację między systemami za pomocą protokołu HTTP (np. za pomocą metod GET, POST, PUT, DELETE). API udostępnia zestaw reguł, które określają jak systemy powinny wymieniać się informacjami i jakie operacje mogą być wykonane na danych.

Tworząc API RESTful udostępnia się wiele puntów końcowych (z ang. *endpoint*), które umożliwiają na interakcje z aplikacją np. dodawanie, pobieranie, aktualizowanie i usuwanie użytkowników.

GraphQL

GraphQL jest językiem zapytań i jednocześnie protokołem sieciowym, który pozwala na zapytanie o konkretne dane z serwera. GraphQL pozwala na żądanie tylko tych danych, które są potrzebne dla aplikacji, zamiast tradycyjnego pobierania wszystkich danych z endpointu, co jest często nieefektywne.

W GraphQL klient tworzy zapytanie, które opisuje, jakie dane są potrzebne, i wysyła je do serwera. Serwer analizuje zapytanie, przetwarza je i zwraca tylko te dane, które są potrzebne klientowi. Dzięki temu klient ma pełną kontrolę nad danymi, jakie otrzymuje, a serwer nie musi przesyłać niepotrzebnych danych.

GraphQL umożliwia również dynamiczne tworzenie zapytań, co pozwala na elastyczność i skalowalność aplikacji. GraphQL jest często używany w aplikacjach mobilnych i webowych, gdzie szybkość i wydajność są kluczowe.

RPC

RPC (Remote Procedure Call) to mechanizm komunikacji między systemami, który pozwala na wywołanie procedury zdalnej jakby była to lokalna procedura.

W przypadku RPC, klient wywołuje procedurę zdalną, a serwer przetwarza ją i zwraca odpowiedź. W trakcie wywołania, klient i serwer wymieniają się odpowiednimi danymi, tak aby procedura mogła zostać wykonana na serwerze i odpowiednie dane zwrócone do klienta.

RPC jest podobny do wywoływania procedur lokalnych, ale pozwala na komunikację między systemami, które mogą być rozłożone geograficznie lub fizycznie. Dzięki temu aplikacje mogą korzystać z usług i danych z różnych systemów, co

pozwala na rozproszenie funkcjonalności i skalowanie aplikacji.

RPC jest często używany w aplikacjach rozproszonych, takich jak usługi sieciowe, aplikacje mobilne, i mikrousługi.

Popularnymi rozwiązaniami wykorzystującymi RPC jest gRPC (RPC stworzone Google) oraz tRPC (TypeScript RPC).

Dodatkowe linki

- What is RESTful API? - RESTful API Beginner’s Guide - AWS
- What is a REST API?
- GraphQL | A query language for your API
- Explore GraphQL: The API for modern apps.
- Apollo GraphQL | Supergraph: unify APIs, microservices, & databases in a composable graph
- Zdalne wywołanie procedury – Wikipedia, wolna encyklopedia
- gRPC - Wikipedia
- gRPC
- tRPC - Move Fast and Break Nothing. End-to-end typesafe APIs made easy. | tRPC

29. Bazy danych

Bazy danych to systemy zarządzania danymi, które pozwalają na przechowywanie, modyfikowanie, przeszukiwanie i udostępnianie danych w sposób efektywny i bezpieczny. Bazy danych składają się z trzech podstawowych elementów: danych, schematu bazy danych i interfejsu użytkownika.

Dane to informacje przechowywane w bazie danych, które mogą być przetwarzane i udostępnione przez aplikacje. Schemat bazy danych to opis tego, jak dane są organizowane i jakie związki istnieją między nimi. Interfejs użytkownika to sposób, w jaki aplikacje i użytkownicy mogą komunikować się z bazą danych.

Bazy danych można spotkać również pod skrótem *DB* (z ang. od *Database*) oraz *DBMS* jako systemy umożliwiające zarządzanie bazą danych (z ang. od *Database Management System*)

Bazy danych zazwyczaj dzieli się na dwa typy: relacyjne i nierelacyjne.

Relacyjne (SQL)

Znane też pod skrótem *RDBMS* (z ang. od *Relational Database Management System*).

Relacyjne bazy danych (RDB) to rodzaj bazy danych, w którym dane są przechowywane w tabelach, a relacje między tabelami są określane przez klucze obce.

Tablice zawierają rzędy i kolumny, z których każda kolumna odpowiada atrybutowi (np. imię, nazwisko, adres), a każdy wiersz odpowiada pojedynczej pozycji (np. konkretnemu klientowi lub produktowi). Klucze obce pozwalają na łączenie danych z różnych tabel, np. informacje o klientach mogą być połączone z informacjami o ich zamówieniach poprzez klucz obcy.

Relacyjne bazy danych są zarządzane za pomocą języka zapytań relacyjnych (SQL) i są często wykorzystywane w aplikacjach biznesowych, takich jak systemy CRM i ERP. Popularne systemy zarządzania bazami danymi relacyjnymi to np. Oracle, MySQL, Microsoft SQL Server, PostgreSQL, IBM Db2, Microsoft Access, SQLite, MariaDB.

Nierelacyjne (NoSQL)

Znane też pod skrótem *NRDBMS* (z ang. od *Non-Relational Database Management System*).

Nierelacyjne bazy danych (NoSQL) to rodzaj baz danych, które nie korzystają z tabel i relacji między nimi jak w przypadku baz relacyjnych.

Bazy NoSQL przechowują dane w różnych formatach, takich jak dokumenty, klucze-wartości, grafy. Są one często używane do przechowywania dużych ilości danych w rozproszonych systemach, takich jak chmury obliczeniowe. Mogą one również przetwarzać dane w czasie rzeczywistym, co czyni je odpowiednimi do aplikacji takich jak aplikacje mobilne lub aplikacje internetowe z dużym ruchem.

NoSQL bazy danych są często używane w aplikacjach, które wymagają skalowalności i wysokiej wydajności, takich jak aplikacje społecznościowe, systemy monitorowania danych,

aplikacje IoT. Przykładami popularnych systemów zarządzania bazami danymi NoSQL są np. MongoDB, Redis, Elasticsearch, Cassandra, Splunk, Amazon Dynamo DB,

ORMy

ORM (ang. Object-Relational Mapping) to technologia pozwalająca na mapowanie obiektów języka programowania na tabele w relacyjnej bazie danych oraz na mapowanie wierszy tabel na obiekty języka programowania. ORM pozwala programiście na pracę z bazą danych przy użyciu obiektów i metod języka programowania, zamiast korzystać z języka zapytań SQL.

ORM zapewnia nie tylko oszczędność czasu i zwiększenie produktywności programistów, ale również zwiększa przejrzystość i czytelność kodu oraz zwiększa bezpieczeństwo aplikacji poprzez eliminację błędów i zwiększenie odporności na ataki.

Istnieje wiele różnych bibliotek i narzędzi ORM dostępnych dla różnych języków programowania, takich jak Hibernate dla Javy, Entity Framework dla .NET, Doctrine dla PHP, SQLAlchemy dla Pythona, czy Prisma dla Node.js

Dodatkowe linki

- DB-Engines Ranking - popularity ranking of database management systems
- The Different Types of Databases - Overview with Examples
- Prisma | Next-generation ORM for Node.js & TypeScript

30. Szyfrowanie i haszowanie

Szyfrowanie

Szyfrowanie to proces zamiany czytelnego tekstu (plaintext) na zaszyfrowany tekst (ciphertext) za pomocą klucza, tak aby tylko osoby posiadające ten klucz mogły odczytać pierwotny tekst. Szyfrowanie może być symetryczne lub asymetryczne. Symetryczne oznacza, że ten sam klucz jest używany do zaszyfrowania i odszyfrowania tekstu, natomiast asymetryczne oznacza, że różne klucze są używane do zaszyfrowania i odszyfrowania tekstu.

W technologiach backendowych szyfrowanie pozwala na tworzenie rozwiązań zapewniających bezpieczeństwo i prywatność. Przykładem może być np. szyfrowanie ruchu sieciowego umożliwiającego odszyfrowanie przesłanych danych tylko przez docelowego odbiorcę.

Haszowanie

Haszowanie to proces polegający na generowaniu skrótu dla określonego ciągu danych (np. hasła, pliku itp.). Hasz jest skrótem, który jest zawsze tej samej długości, niezależnie od długości oryginalnego ciągu danych i jest unikalny dla każdego ciągu danych.

Haszowanie jest używane do zabezpieczania danych, ponieważ hasze są trudne do odwrócenia (co oznacza, że nie można odtworzyć oryginalnego ciągu danych na podstawie jego hasza). Haszowanie jest również często używane do porównywania dwóch ciągów danych, ponieważ

jeśli hasze dwóch ciągów danych są takie same, oznacza to, że ciągi te są takie same.

Niektóre z funkcji haszujących jak np. MD5 zostały złamane tj. odkryto kolizje, czyli sytuacje w których różne dane (np. hasła) mogą zostać przedstawione za pomocą takiego samego ciągu znaków hasza MD5.

Mimo, że odwrócenie działania funkcji haszującej nie jest możliwe (nie można odszyfrować hasła zapisanego w formie haszu), to jednak można wygenerować listę wielu haszy reprezentujących różne hasła. Finalnie pozostaje wtedy jedynie znalezienie identycznych haszy, aby poznać faktyczne hasło.

Funkcje haszujące powinny być więc na tyle skomplikowane, aby generowanie ich, zajmowało współczesnym komputerom istotną ilość czasu, utrudniając tym samym tworzenie słowników haszy z odpowiadającymi im danymi w formie czystego tekstu.

Dodatkowe linki

- Szyfr – Wikipedia, wolna encyklopedia
- Funkcja skrótu – Wikipedia, wolna encyklopedia
- List of hash functions - Wikipedia

31. Pamięć podręczna

W web developmencie, pamięć podręczna (z ang. *cache*) jest używana do przechowywania danych, które są często używane np. przez przeglądarkę internetową. Dzięki temu przeglądarka może szybciej dostać się do danych, które potrzebuje, co zwiększa szybkość ładowania stron internetowych i zmniejsza liczbę potrzebnych połączeń sieciowych.

Jednym z przykładów jest użycie pamięci podręcznej do przechowywania plików statycznych, takich jak obrazy, pliki CSS i JavaScript. Kiedy przeglądarka ładuje stronę internetową, pobiera te pliki z pamięci podręcznej, jeśli są już tam zapisane. Dzięki temu, gdy użytkownik przechodzi na inną stronę na tej samej witrynie, pliki te nie muszą być ponownie pobierane, co zwiększa szybkość ładowania strony.

Pamięć podręczna jest również często używana do przechowywania danych pochodzących z dynamicznych zapytań do bazy danych. Dzięki temu, gdy dane te są potrzebne ponownie, mogą być pobierane z pamięci podręcznej, co zwiększa wydajność i zmniejsza obciążenie na bazie danych.

Warto również wspomnieć, że pamięć podręczna jest tymczasowa i dane w niej przechowywane są wymazywane po zamknięciu przeglądarki lub po określonym czasie. Web developerzy mogą również ustawić politykę pamięci podręcznej dla swoich aplikacji, aby ustalić, które dane mają

być przechowywane w pamięci podręcznej i jak długo mają tam pozostać.

Dodatkowe linki

- Pamięć podręczna – Wikipedia, wolna encyklopedia
- Server-side caching and Client-side caching - Coding Ninjas CodeStudio

32 .Systemy zarządzania treścią

Znany również jako *CMS* (skrót z ang. od *Content Management System*).

CMS, czyli system zarządzania treścią, to oprogramowanie umożliwiające tworzenie, edycję i zarządzanie treścią na stronie internetowej bez konieczności posiadania specjalistycznej wiedzy z zakresu programowania. CMS pozwala na łatwą edycję treści przez osoby nie posiadające umiejętności programowania, umożliwiając ich publikację na stronie internetowej bez potrzeby udziału programisty.

Bezgłowy CMS

Z języka angielskiego znany jako *Headless CMS*.

Headless CMS to rodzaj systemu zarządzania treścią, który pozbawiony jest interfejsu do konsumowania treści. Oznacza to, że headless CMS wymaga stworzenia warstwy wizualnej dla końcowych użytkowników aplikacji, ale ma już wbudowany interfejs do edycji treści przez administratorów.

Taki CMS pozwala również na dostęp do danych za pomocą API (interfejsu programowania aplikacji). Dzięki temu, headless CMS jest bardzo elastyczny i pozwala na łatwą integrację z innymi systemami oraz na tworzenie aplikacji na różnych platformach, jako strony internetowe, aplikacje mobilne, czy internet rzeczy (IoT).

Chmurowy CMS

Z języka angielskiego znany jako *Cloud CMS*.

Chmurowy CMS, to rodzaj systemu zarządzania treścią, którego serwery i infrastruktura są zlokalizowane w centrum danych zarządzanym przez dostawcę takiego CMSa.

Jako użytkownik takiego CMSa dostajemy do dyspozycji panel do zarządzania treścią z poziomu admina, oraz dostęp do API w celu konsumpcji treści, integrując ją z naszą aplikacją. Nie interesują nas też kwestie techniczne tj. hosting CMSa, jego aktualizowanie, baza danych z której korzysta, itp. technikalia.

Cena jaką płacimy za taką wygodę jest z góry ustalona przez dostawcę usługi tj. wymagane jest wnoszenie zazwyczaj regularnych miesięcznych opłat za korzystanie z chmurowego CMSa.

Samodzielnie hostowany CMS

To taki CMS, który musimy samodzielnie zainstalować na wybranym przez nas hostingu. CMSy wymagają również dostępu do bazy danych, więc oprócz administrowaniem CMSem, musimy dodatkowo zarządzać bazą danych (tworząc np. regularne kopie zapasowe).

Rozwiązania tego typu są z reguły darmowe w kontekście używania CMSa (można znaleźć wiele otwarto źródłowych rozwiązań). Płatności jakie ponosimy w takim rozwiązaniu są związane z utrzymaniem CMSa na serwerze (opłaty hostingowe). Płacimy też swoim czasem, ponieważ skonfigurowanie serwera i zainstalowanie na nim CMSa wymaga dodatkowego czasu i wiedzy technicznej.

Dodatkowe linki

- System zarządzania treścią – Wikipedia, wolna encyklopedia
- CMS’y tradycyjne i bezgłowe

33. Silniki wyszukiwania

Silniki wyszukiwania przeszukują zawartość stron internetowych, analizując ich treść i metadane, takie jak tytuł i opis, a następnie indeksują je w swoich bazach danych. Kiedy użytkownik wprowadza zapytanie do wyszukiwarki, silnik wyszukiwania przeszukuje swoją bazę danych i zwraca odpowiednie wyniki.

Popularnym silnikiem wyszukiwania jest Google. Ale Google to silnik, który indeksuje wszystkie strony w internecie. Możemy oczywiście skorzystać z rozwiązań, które umożliwią nam osadzenie wyszukiwarki od Google w naszym serwisie, aby przeszukiwane były tylko nasze podstrony.

Nie jest to jednak rozwiązanie idealne na tym polu i na rynku dostępne są narzędzia które oferują większą kontrolę (indeksowanie strony odbywać się może w zarządzanej przez nas bazie danych) i umożliwiają tzw. pełnotekstowe wyszukiwanie w obrębie danego serwisu. Popularne silniki wyszukiwania do użytku w obrębie danej aplikacji to np. Elasticsearch, Algolia, Solr, czy Lucene.

Dodatkowe linki

- Elasticsearch: The Official Distributed Search & Analytics Engine | Elastic
- Site Search & Discovery powered by AI | Algolia
- Welcome to Apache Solr - Apache Solr
- Apache Lucene - Welcome to Apache Lucene

34. Brokery wiadomości

Broker wiadomości (ang. message broker) to oprogramowanie, które pośredniczy w komunikacji między różnymi systemami lub aplikacjami. Służy do przesyłania i przetwarzania wiadomości między różnymi komponentami architektury rozproszonej.

Broker wiadomości działa jako pośrednik między nadawcą i odbiorcą wiadomości. Nadawca wysyła wiadomość do brokera, a broker przesyła ją do odpowiedniego odbiorcy. Broker wiadomości może również przetwarzać wiadomości, np. zmieniając ich format lub filtrując je, zanim zostaną one dostarczone do odbiorcy.

Broker wiadomości jest często używany w architekturze mikrousług, która polega na podzieleniu aplikacji na małe, niezależne jednostki, które komunikują się między sobą przez brokera wiadomości. Dzięki temu, każda usługa może działać niezależnie, co pozwala na łatwiejsze tworzenie i utrzymanie aplikacji.

Broker wiadomości może być również używany do komunikacji między różnymi językami programowania, systemami operacyjnymi lub różnymi platformami.

Popularnymi systemami brokerów wiadomości są np. Kafka, RabbitMQ, czy Memphis.

Dodatkowe linki

- Message broker - Wikipedia
- Apache Kafka

- Messaging that just works — RabbitMQ
- Streaming For Modern Applications | Memphis.dev

35. Wzorce projektowe

Wzorce projektowe (ang. design patterns) to gotowe rozwiązania problemów, które pojawiają się często w projektowaniu oprogramowania. Są to sprawdzone sposoby rozwiązywania problemów, które zostały opisane i zweryfikowane przez wielu programistów.

Wzorce projektowe służą do rozwiązywania problemów dotyczących organizacji kodu, zarządzania dostępnością danych, tworzenia interfejsów użytkownika, itp.

Istnieje wiele różnych kategorii wzorców projektowych, takich jak wzorce kreacyjne, wzorce strukturalne, wzorce zachowania.

Wzorce kreacyjne służą do tworzenia nowych obiektów, takie jak fabryka (Factory), która służy do tworzenia nowych obiektów bez konieczności znajomości konkretnej klasy. Wzorce strukturalne służą do organizowania kodu i połączenia obiektów, takie jak adapter (Adapter), który pozwala na połączenie dwóch różnych interfejsów. Wzorce zachowania służą do opisania relacji między obiektami oraz ich zachowania, takie jak wzorzec obserwator (Observer), który pozwala na rejestrowanie zmian stanu obiektu i powiadomienie o nich innych obiektów.

Stosowanie wzorców projektowych pozwala na poprawienie jakości kodu, zwiększenie jego czytelności i łatwiejsze utrzymanie oprogramowania.

Dodatkowe linki

- Software design pattern - Wikipedia
- Katalog wzorców projektowych
- Design Patterns for Humans - roadmap.sh

36. Hosting backendu

Uruchamianie kodu backendowego wymaga zdalnego dostępu do komputera (serwera), na którym zostanie uruchomiony kod. W najprostszym wariancie hostingu współdzielonego, dostęp może być ograniczony wyłącznie do usługi transferu plików (FTP) oraz korzystania z preinstalowanego oprogramowania (bazy danych, poczty email).

W bardziej zaawansowanych wariantach otrzymujemy zdalny dostęp do serwera (za pomocą połączenia SSH), na którym możemy instalować dowolne oprogramowanie i wykonywać dowolne polecenia.

Hosting współdzielony

Hosting współdzielony oznacza, że wiele stron internetowych jest hostowanych w obrębie jednego serwera. Każda ze stron korzysta z tej samej ilości zasobów, takich jak procesor, pamięć i przestrzeń dyskowa. Jest to najtańszy sposób hostowania stron internetowych, ponieważ koszty są dzielone między wszystkimi użytkownikami tego samego serwera.

Współdzielone są nie tylko zasoby ale również zainstalowane oprogramowanie. Przykładowo baza danych jest zainstalowana jako jeden współdzielony system. Dostęp do bazy jest jedynie ograniczany prawami dostępu tj. utworzeniem użytkownika bazy danych z przypisaną do niego tabelą bazy.

Na takim hostingu nie mamy możliwości instalowania własnego oprogramowania. Przykładowo, na hostingu współdzielonym z zainstalowaną jedynie obsługą języka PHP, nie mamy możliwości zainstalowania środowiska Node.js.

Mimo, że marketingowo można spotkać bardzo atrakcyjne oferty, oferujące duże lub czasem nawet “nielimitowane” zasobu (np. transferu danych), to w praktyce mamy możliwość korzystania tylko z małej części deklarowanych parametrów.

Używana jest tutaj taktyka “oversellingu”, czyli sprzedawania większej ilości zasobów, niż firma hostingowa faktycznie posiada, zakładając, że nie zdarzy się sytuacja kiedy wszyscy klienci będą chcieli wykorzystać je w 100%.

W przypadku gdyby Twój serwis zużywał duże ilości zasobów, Twoja usługa hostingu współdzielonego zostanie po prostu wymówiona przez usługodawcę.

Hosting VPS

Hosting VPS (Virtual Private Server) oznacza, że każdy użytkownik korzysta z wirtualnego serwera, który jest oddzielony od innych użytkowników na tym samym fizycznym serwerze. Oznacza to, że każdy użytkownik otrzymuje własną (wirtualnie wydzieloną) przestrzeń dyskową, pamięć i moc obliczeniową procesora (*vCPU* czyli *virtual CPU*).

W praktyce działanie VPSa nie różni się prawie niczym od działania dedykowanego serwera. Do dyspozycji otrzymujemy możliwość zalogowania się do wybranego przez nas systemu operacyjnego, który zostaje

zainstalowany na wirtualnej maszynie, której zasoby również samodzielnie wybieramy (ilość RAMu, przestrzeń dyskowa, ilość vCPU).

Działanie przypomina to korzystanie z własnego komputera podpiętego do wysoce przepustowego łącza internetowego. Do VPSa logujemy się z poziomu linii poleceń, ale nic nie stoi na przeszkodzie w zainstalowaniu środowiska graficznego i logowaniu się zdalnie do takiego systemu, odbierając strumień przesyłu obrazu z takiego komputera za pomocą technologii typu VNC.

Otrzymujemy też stały, przypisany do wirtualnego serwera adres IP.

Hosting dedykowany

Oferuje wszystko to co oferuje VPS.

Od VPSa różni się tylko tym, że zamiast dzierżawić wydzielone zasoby komputera w formie wirtualnej maszyny, uzyskujemy dostęp do całego komputera, który jest w pełni do naszej prywatnej dyspozycji.

Daje to możliwość zainstalowania dowolnego systemu operacyjnego oraz uruchamiania oprogramowania do wirtualizacji (np. Proxmox), dzięki czemu umożliwia wydzielanie własnych VPSów w obrębie takiej dedykowanej maszyny.

Hosting chmurowy

Hosting chmurowy to to samo co hosting współdzielony lub VPS, tylko okraszony marketingowym sloganem *Cloud*. Nie wprowadza on z technicznego punktu widzenia żadnych

różnic. Chmura to po prostu usługa korzystania z dzierżawionych zasobów informatycznych.

Hosting funkcji

Hosting funkcji jest innowacyjnym typem hostingu, który polega na hostowaniu wyłącznie fragmentów kodu (funkcji). Wykonywanie działania takiego kodu zużywa zasoby, za które się płaci według ustalonej stawki.

Płacąc wyłącznie za czas wykonywania danego kodu można zoptymalizować koszty działania aplikacji. Nie jest to jednak reguła i dzierżawa maszyny w ujęciu miesięcznym (np. VPS) może okazać się bardziej opłacalna niż płacenie za czas wykonywania kodu.

Rozproszenie kodu aplikacji na niezależnie wykonywane funkcje może być wadą lub zaletą. Wszytko zależy od konkretnego przypadku i decyzję w tym zakresie trzeba rozpatrywać indywidualnie.

Popularnym rozwiązaniem w tym zakresie jest AWS Lambda.

Dodatkowe linki

- Virtual private server – Wikipedia, wolna encyklopedia
- Serwer dedykowany – Wikipedia, wolna encyklopedia
- What is the cloud? | Cloud definition | Cloudflare
- What is AWS Lambda? - AWS Lambda
- Virtual Network Computing – Wikipedia, wolna encyklopedia

Full Stack

37. Routing

Routing w web development oznacza mapowanie adresów URL na odpowiednie kontrolery lub funkcje w aplikacji. Pozwala to na przypisanie różnych akcji dla różnych adresów URL, umożliwiając przeglądanie różnych stron i funkcjonalności na stronie internetowej. Routing jest często używany w aplikacjach opartych na frameworku MVC (Model-View-Controller) i pozwala na separację logiki aplikacji od interfejsu użytkownika.

W tłumaczeniu na j. polski oznacza trasowanie, czyli wyznaczanie tras.

Dodatkowe linki

- Trasowanie (telekomunikacja) – Wikipedia, wolna encyklopedia
- Routing - Wikipedia

38. Paginacja

Paginacja (z ang. *pagination*, w j. polskim spotykana też jako *stronicowanie*) to technika, która pozwala na podzielenie dużego zbioru danych na mniejsze strony. Jest to często używane w przypadku wyświetlania danych w tabelach lub na stronach internetowych.

Działanie paginacji polega na tym, że dane są dzielone na mniejsze grupy, z których każda jest wyświetlana na osobnej stronie. Użytkownicy mogą przeglądać te strony, przechodząc między nimi za pomocą przycisków “następna” i “poprzednia”.

Przykładowo, jeśli mamy bazę danych zawierającą 1000 rekordów, paginacja pozwoli na wyświetlenie tylko określonej liczby rekordów na stronie, np. 50. W takim przypadku baza danych zostanie podzielona na 20 stron, z których każda będzie zawierała 50 rekordów.

Paginacja pozwala na łatwiejsze przeglądanie i filtrowanie danych przez użytkowników, a także na optymalizację wydajności działania aplikacji poprzez zmniejszenie liczby przesyłanych i przetwarzanych danych na raz.

Dodatkowe linki

- Pagination - Wikipedia

39. Autentykacja i autoryzacja

Autentykacja

Autentykacja to proces potwierdzania tożsamości użytkownika. W informatyce istnieje wiele różnych metod autentykacji, takich jak hasła, klucze publiczne, uwierzytelnianie biometryczne (np. odcisk palca, rozpoznawanie twarzy) i tokeny jednorazowe (np. kody SMS). Celem autentykacji jest upewnienie się, że osoba lub urządzenie, które próbuje uzyskać dostęp do systemu lub danych, jest tym, za kogo się podaje.

Autentykacja to spolszczona forma angielskiego słowa *authentication*. Mimo, że uważana za niepoprawną formę słowa *uwierzytelnianie*, autentykacja jest często występującym terminem w technicznych dokumentacjach pisanych w języku polskim.

Popularnymi mechanizmami autentykacji w web developmencie są:

- Hasła - najprostszy sposób autentykacji, polegający na wprowadzeniu przez użytkownika nazwy użytkownika i hasła
- Sesje - polegające na przydzieleniu użytkownikowi unikalnego identyfikatora sesji po udanej autentykacji przez hasło
- Ciastka - polegające na przydzieleniu użytkownikowi unikalnego identyfikatora ciasteczka po udanej autentykacji przez hasło

- OAuth/OpenID - polegające na uwierzytelnieniu użytkownika przez trzecią stronę, taką jak Google, Facebook, czy Twitter
- Tokeny JWT (JSON Web Token) - polegające na generowaniu przez system tokenu, który jest przesyłany w nagłówku żądania HTTP i wykorzystywany do uwierzytelnienia użytkownika

Autoryzacja

Autoryzacja to proces potwierdzający, że użytkownik posiada upoważnienie do dostępu do określonych zasobów lub funkcjonalności. Jest to krok po autentykacji, która potwierdza tożsamość użytkownika.

W informatyce, autoryzacja może odbywać się na różne sposoby, takie jak:

- Role-based access control (RBAC): użytkownicy są przypisywani do ról, a dostęp do zasobów jest kontrolowany na podstawie przynależności do danej roli
- Access control list (ACL): dostęp do zasobów jest kontrolowany na podstawie listy uprawnień
- Attribute-based access control (ABAC): dostęp do zasobów jest kontrolowany na podstawie atrybutów użytkownika (np. poziom dostępu, lokalizacja geograficzna)

Autoryzacja jest ważna, aby zapewnić odpowiedni poziom bezpieczeństwa i kontroli dostępu do zasobów i funkcjonalności.

Dodatkowe linki

- Tworzenie systemu autentykacji | Full Stack | Web Development | Panel Nauki | Codisity
- JSON Web Token – Wikipedia, wolna encyklopedia
- OAuth – Wikipedia, wolna encyklopedia
- Sesja (informatyka) – Wikipedia, wolna encyklopedia
- Basic access authentication - Wikipedia
- Access token - Wikipedia
- Identity

40. Przetwarzanie płatności

Przetwarzanie płatności w web developmencie polega na umożliwieniu użytkownikom dokonywania płatności przez stronę internetową za pomocą szybkich przelewów, BLIKa, kart płatniczych i innych jak np. PayPal, Google/Apple Pay.

Aby przetwarzać płatności, najpierw trzeba zintegrować stronę z systemem płatności, który będzie odpowiedzialny za przetwarzanie transakcji. Systemy płatności zazwyczaj oferują API, które pozwala na przesyłanie danych karty kredytowej z formularza na stronie do systemu płatności.

Kolejnym krokiem jest uwierzytelnienie transakcji, które polega na sprawdzeniu, czy dane karty kredytowej są prawidłowe oraz czy jest wystarczająca ilość środków na koncie.

Jeśli transakcja jest prawidłowa, system płatności potwierdza ją i przekazuje informacje o transakcji do sklepu internetowego, który może wykorzystać te dane, aby zakończyć transakcję i wysłać potwierdzenie do klienta.

Oczywiście, ważne jest aby proces przetwarzania płatności był zabezpieczony przed oszustwami i kradzieżami danych, zwłaszcza poprzez stosowanie protokołów SSL/TLS oraz metod autentykacji 3D Secure.

Dodatkowe linki

- Bramka płatnicza – Wikipedia, wolna encyklopedia
- Web Payments

41. WebSocket

WebSocket to protokół komunikacyjny umożliwiający dwukierunkową komunikację między klientem a serwerem przez pojedyncze połączenie sieciowe. Jest to alternatywa dla tradycyjnego protokołu HTTP, w którym komunikacja odbywa się poprzez wymianę pojedynczych żądań i odpowiedzi. WebSocket pozwala na dynamiczne i natychmiastowe przesyłanie danych, co jest szczególnie przydatne w przypadku aplikacji takich jak czaty, gry online czy aplikacje w czasie rzeczywistym.

Dodatkowe linki

- The WebSocket API (WebSockets) - Web APIs | MDN
- WebSocket – Wikipedia, wolna encyklopedia
- Socket.IO

42. Testy automatyczne

Testy automatyczne to proces polegający na wykorzystaniu specjalnie napisanego oprogramowania, zwanego testerką automatyczną, do automatycznego wykonywania testów oprogramowania. Testy te polegają na porównywaniu oczekiwanego zachowania oprogramowania z rzeczywistym zachowaniem, w celu wykrycia błędów lub niezgodności. Testy automatyczne pozwalają na szybkie i efektywne przetestowanie aplikacji, co pozwala na wczesne wykrycie błędów i zwiększa jakość oprogramowania.

Istnieją różne rodzaje testów automatycznych, takie jak testy jednostkowe, testy integracyjne, testy akceptacyjne czy testy e2e (end-to-end). Służą one do testowania różnych aspektów oprogramowania, od testowania pojedynczych funkcji po testowanie całego systemu.

Testy są tworzone bezpośrednio przez programistów, którzy piszą kod aplikacji, ale mogą być też tworzone przez testerów, którzy zajmują się tylko testowaniem.

Testy jednostkowe

Testy jednostkowe to rodzaj testów automatycznych, które polegają na sprawdzaniu pojedynczych elementów oprogramowania, takich jak funkcje, metody czy klasy. Celem testów jednostkowych jest zweryfikowanie, czy dany fragment kodu działa zgodnie z oczekiwaniami, czyli czy działa poprawnie i spełnia swoje zadanie.

Testy jednostkowe są przydatne, ponieważ pozwalają na wykrycie błędów na etapie ich powstawania, co jest dużo prostsze i tańsze niż ich naprawa na późniejszym etapie rozwoju oprogramowania. Są pisane w języku programowania, w którym jest napisany kod aplikacji i korzystają z bibliotek testowych, które umożliwiają automatyzację wykonywania testów.

Testy integracyjne

Testy integracyjne to rodzaj testów automatycznych, które polegają na sprawdzaniu jak poszczególne elementy oprogramowania współpracują ze sobą, takie jak różne moduły, funkcje czy komponenty. Celem testów integracyjnych jest zweryfikowanie, czy różne elementy oprogramowania są ze sobą poprawnie połączone i czy razem działają tak, jak powinny.

Testy integracyjne są przydatne, ponieważ pozwalają na wykrycie błędów wynikających z niepoprawnej współpracy między poszczególnymi elementami oprogramowania. Są one ważne, ponieważ pozwalają na znalezienie błędów, które mogłyby przeoczyć testy jednostkowe, czyli testy pojedynczych elementów aplikacji.

Tak samo jak testy jednostkowe, testy integracyjne są pisane w języku programowania, w którym jest napisany kod aplikacji, i korzystają z bibliotek automatyzujących testowanie. Testy integracyjne przeprowadzane są często po testach jednostkowych, po to żeby upewnić się że poszczególne elementy działają poprawnie w połączeniu ze sobą.

Testy e2e (end-to-end)

Testy end-to-end (E2E) to rodzaj testów automatycznych, które polegają na sprawdzaniu całego systemu od strony użytkownika, czyli od przesłania żądania przez przeglądarkę do otrzymania odpowiedzi. Celem testów E2E jest zweryfikowanie, czy cały system działa poprawnie i spełnia swoje zadanie od strony użytkownika.

Testy E2E mają na celu przetestowanie całego systemu jak w rzeczywistych warunkach, włącznie z interakcją z bazami danych, serwerami, systemami zewnętrznymi i innymi składnikami. Służą one do testowania, czy aplikacja działa zgodnie z oczekiwaniami pod kątem użyteczności, niezawodności i wydajności.

Testy E2E są najbardziej zasobożerne, wymagają bowiem uruchomienia przeglądarki (zazwyczaj w trybie bez interfejsu graficznego), na której wykonane zostaną testy. Mimo, że wykonują się automatycznie, czas trwania tych testów jest najdłuższy, porównując go do testów integracyjnych czy jednostkowych. Z reguły tworzy się więc tych testów ilościowo najmniej.

Dodatkowe linki

- Software testing - Wikipedia
- Testowanie oprogramowania – Wikipedia, wolna encyklopedia
- Test jednostkowy – Wikipedia, wolna encyklopedia
- Test akceptacyjny – Wikipedia, wolna encyklopedia
- Jest · 🃏 Delightful JavaScript Testing
- Testing Library | Testing Library
- Fast and reliable end-to-end testing for modern web apps | Playwright
- WebDriver | Selenium

DevOps

43. Administrowanie systemami

Administrowanie systemem polega na konfiguracji i utrzymywaniu sprawnie działającego systemu operacyjnego, zainstalowanego w nim oprogramowania, usług, oraz dbanie o bezpieczeństwo.

Do podstawowych zadań administratorów należą:

- Instalowanie i aktualizowanie systemu operacyjnego oraz oprogramowania
- Konfiguracja usług, takich jak serwer WWW, baza danych, serwer poczty elektronicznej
- Tworzenie i zarządzanie kontami użytkowników oraz uprawnieniami
- Monitorowanie wydajności i zużycia zasobów
- Tworzenie kopii zapasowych i planowanie odzyskiwania danych w razie awarii
- Dbanie o bezpieczeństwo serwera poprzez stosowanie zabezpieczeń, takich jak firewall, ochrona przed atakami DDoS, szyfrowanie danych.

Administrowanie wymaga znajomości systemów operacyjnych, protokołów sieciowych, bezpieczeństwa oraz umiejętności zarządzania serwerem.

W web developmencie najczęściej spotykanymi systemami operacyjnym, którym zarządzają administratorzy są dystrybucje Linuksa (Ubuntu, Debian, Fedora, CentOS, openSUSE) oraz wersje UNIXowego systemu *BSD (FreeBSD, OpenBSD)

Dodatkowe linki

- Linux – Wikipedia, wolna encyklopedia
- Berkeley Software Distribution – Wikipedia, wolna encyklopedia
- DevOps Automation Solutions | Chef
- Puppet – Powerful infrastructure automation and delivery
- Ansible is Simple IT Automation
- Terraform by HashiCorp
- Docker: Accelerated, Containerized Application Development

44. Serwery internetowe

Lepiej znane pod anglojęzyczną nazwą *web server*.

Web server to oprogramowanie, które udostępnia strony internetowe oraz inne zasoby sieciowe, najczęściej na żądanie przeglądarki internetowej, ale też i innych systemów korzystających z API udostępnionego na serwerze. Web serwer przetwarza żądania (najczęściej HTTP), a następnie przesyła odpowiedzi w postaci plików HTML, CSS, JavaScript, JSON, obrazów oraz innych typów plików.

Istnieje wiele popularnych web serwerów, takich jak Apache, Nginx, IIS, Tomcat, które różnią się między sobą funkcjonalnością oraz skalowalnością. Web serwery często są używane w połączeniu z innymi narzędziami, takimi jak bazy danych, aplikacje serwerowe oraz skrypty języków backendowych.

Dodatkowe linki

- Serwer WWW – Wikipedia, wolna encyklopedia
- Web server - Wikipedia

45. CI/CD

CI/CD to skrót od *Continuous Integration/Continuous Deployment*. Jest to metoda zarządzania procesami tworzenia oprogramowania, która polega na ciągłym integrowaniu, testowaniu i wdrażaniu kodu. W procesie CI, kod jest integrowany przez różnych członków zespołu, a następnie automatycznie testowany. W procesie CD, kod jest automatycznie wdrażany na produkcyjne środowisko po przejściu testów. Celem jest zwiększenie jakości kodu i szybkości dostarczania nowych funkcjonalności.

Dodatkowe linki

- GitHub Actions Documentation - GitHub Docs
- GitLab CI/CD | GitLab
- Jenkins
- Bamboo Continuous Integration and Deployment Build Server
- Home – Travis-CI

46. Wirtualizacja i konteneryzacja

Wirtualizacja polega na emulowaniu sprzętu lub oprogramowania, pozwalając na uruchamianie wielu różnych systemów operacyjnych lub aplikacji na jednym fizycznym komputerze. Wirtualizacja jest szczególnie przydatna do izolacji aplikacji i zwiększania wydajności infrastruktury.

Konteneryzacja jest rodzajem wirtualizacji, która polega na izolowaniu aplikacji w kontenerach, które zawierają wszystko, czego dana aplikacja potrzebuje do działania, włącznie z kodem, bibliotekami i ustawieniami. Kontenery są lekkie i przenośne, co umożliwia łatwe przenoszenie aplikacji między różnymi środowiskami. Dzięki konteneryzacji, aplikacje działają tak samo niezależnie od środowiska, w którym są uruchomione.

47. Infrastruktura

W DevOps, infrastruktura oznacza cały zestaw narzędzi i rozwiązań, które są potrzebne do uruchomienia i zarządzania aplikacjami w środowisku produkcyjnym. Obejmuje to serwery, bazy danych, sieci, magazyny danych, narzędzia do monitorowania i zarządzania, a także rozwiązania chmurowe i konteneryzację.

Infrastruktura w DevOps jest zarządzana przy użyciu automatyzacji i narzędzi do konfiguracji, takich jak Ansible, Puppet, czy terraform. Dzięki temu, infrastruktura jest elastyczna i łatwo skalowalna, a także łatwiej jest ją zarządzać i zapewnić ciągłość działania aplikacji.

Skalowanie

W przypadku gdy stworzona aplikacja zyskuje na zainteresowaniu i jest używana przez coraz to większą ilość użytkowników, pojawia się potrzeba skalowania, aby płynnie obsługiwać narastający ruch.

Skalowanie dzieli się na wertykalne i horyzontalne.

Wertykalne skalowanie oznacza, że zwiększamy zasoby maszyny, która obsługuje naszą aplikację (bądź wydzieloną jej część). Jest to z reguły proste rozwiązanie i wymaga jedynie zatwierdzenia zmiany parametrów u usługodawcy (co oczywiście wiąże się z wyższymi rachunkami za korzystanie z powiększanej usługi).

Wertykalnie można skalować maszyny bardzo wysoko. Jedną maszynę obsługującą aplikację możemy rozszerzać (w

zależności od usługodawcy) do rozmiarów rzędu kilkuset vCPU i kilkuset lub nawet kilku tysięcy GB pamięci RAM.

Koszty takiego skalowania mogą być jednak wysokie, jednocześnie nastawiając nas na ryzyko niedostępności usługi w przypadku awarii takiej maszyny.

Alternatywą jest skalowanie horyzontalne. Polega ono, nie na zwiększaniu zasobów pojedynczej maszyny, ale na dodawaniu kolejnych maszyn i równomiernym rozdystrybuowaniu obciążenia, spowodowanego ruchem użytkowników (tzw. *load balancing*).

Często nie jest to jednak zadanie łatwe i wymaga wdrażania dodatkowych rozwiązań architektonicznych zapewniających np. replikację bazy danych.

Dodatkowe linki

- DevOps – Wikipedia, wolna encyklopedia
- Zarządzanie infrastrukturą informatyczną – Wikipedia, wolna encyklopedia
- Infrastruktura jako usługa – Wikipedia, wolna encyklopedia