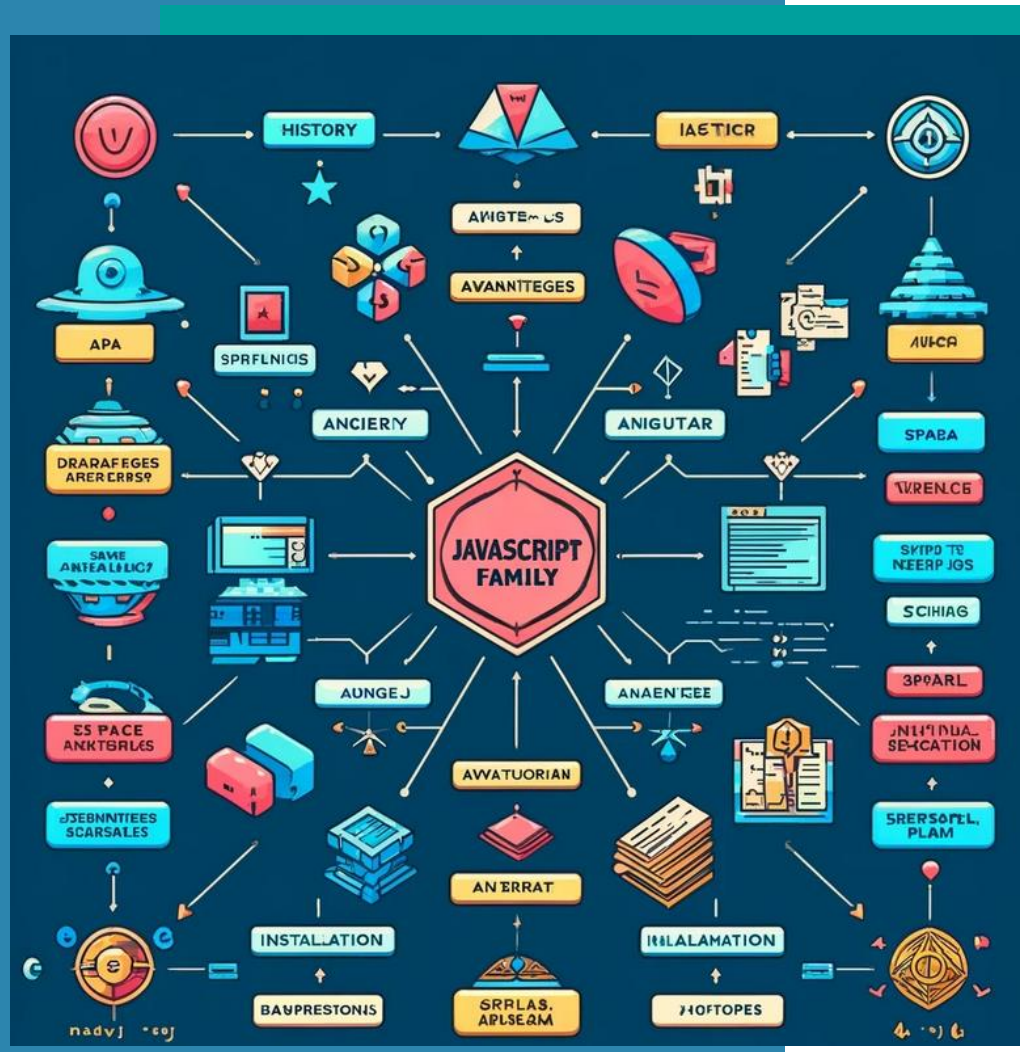
dr inż. Michał Malinowski
michal.malinowski@warszawa.merito.pl

https://www.linkedin.com/in/michał-malinowski-malkon/

Wprowadzenie do Node.js, Angular i Architektura SPA

Wykład 07

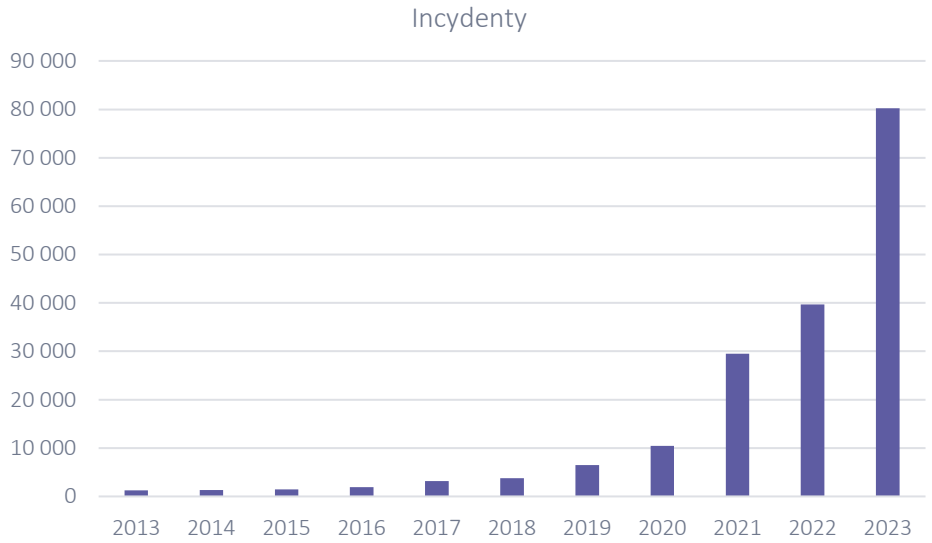
JĘZYKI PROGRAMOWANIA
STUDIUM PRZYPADKU



Zagadnienia (2h)

- Rodzina JavaScript
- Node.js
  - Historia
  - Zalety i wady
  - Architektura
  - Instalacja
  - Przykłady
- Angular
  - Historia
  - Zalety i wady
  - Architektura
  - Instalacja
  - Przykłady
- SPA (Single Page Application)
  - Zalety i wady
  - Wdrożenie
  - Zastosowanie

Cyberbezpieczeństwo

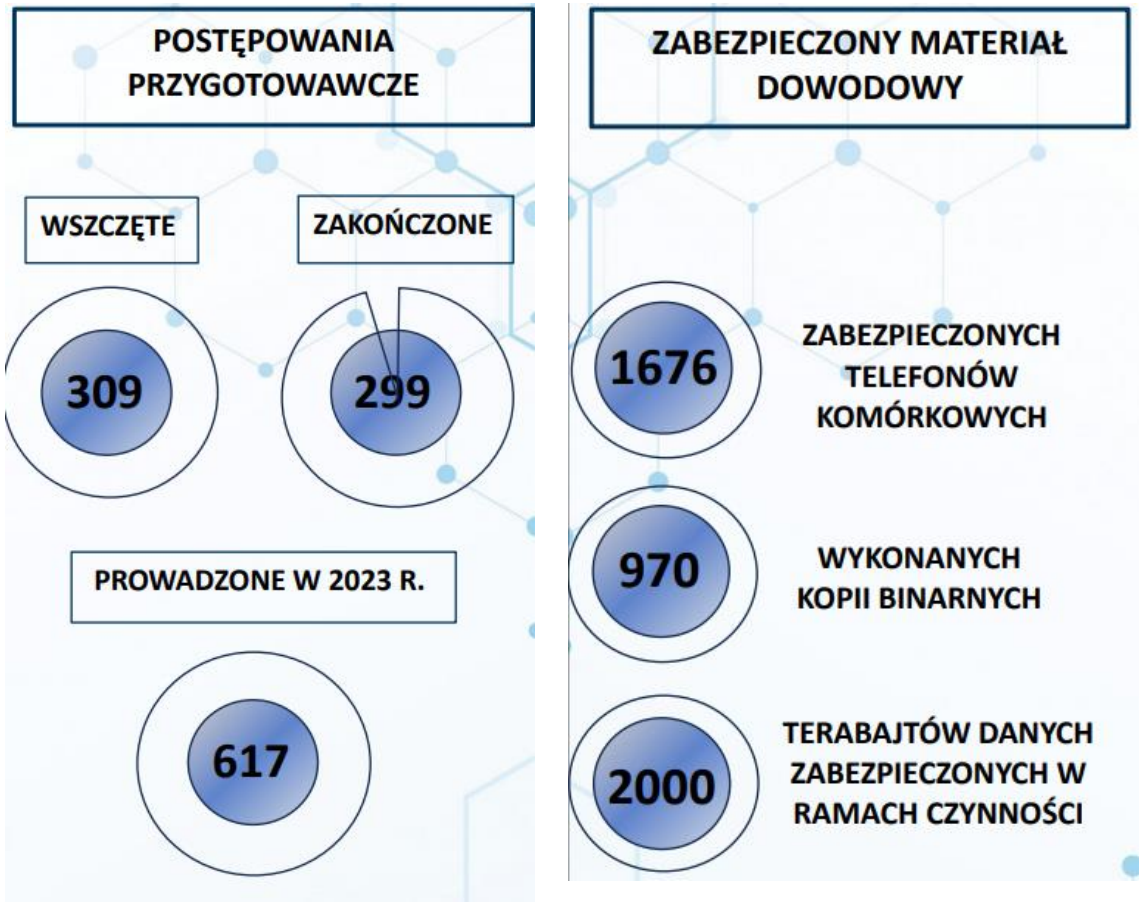


Typy incydentów	Liczba incydentów	Procent wszystkich
Oszustwa komputerowe	75 917	94,58%
Szkodliwe oprogramowanie	1 650	2,06%
Podatne usługi	964	1,20%
Obraźliwe i nielegalne treści	584	0,73%
Włamania	418	0,52%
Dostępność zasobów	385	0,48%
Próby włamań	205	0,26%
Atak na bezpieczeństwo informacji	59	0,07%
Inne	56	0,07%
Gromadzenie informacji	29	0,04%
Razem	80 267	100,00%
Osoby fizyczne	2 105	2,62%

Raport roczny z działalności CERT POLSKA (CSIRT NASK)
https://cert.pl/uploads/docs/Raport_CP_2023.pdf
(dostęp: 24 kwietnia 2024).

Liczba incydentów: 80 267 (2023)

Cyberbezpieczeństwo



Liczba postępowań przygotowawczych prowadzonych przez CBZC w 2023: 617

Co stanowi 0,77% incydentów z 2023 zidentyfikowanych przez CISIRT NASK

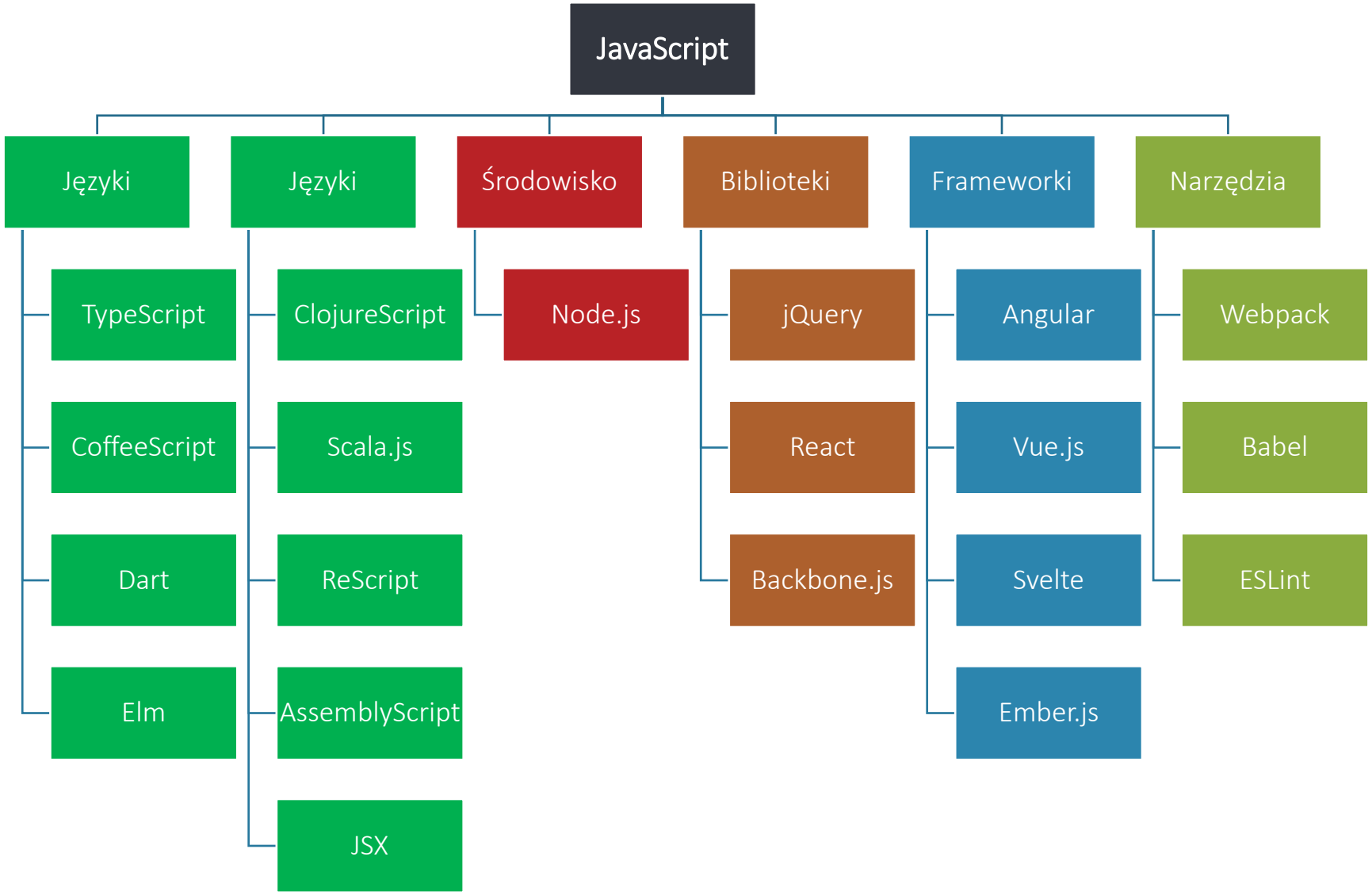
WYNIKI STATYSTYCZNE Centralnego Biura Zwalczania Cyberprzestępczości za 2023 rok

https://cbzc.policja.gov.pl/download/442/751/WYNIKISTATYSTYCZNECENTRALNEGOBIURAZWALCZANIACYBERPRZESTEPCZOSCIZA2023ROK.pdf (dostęp: 24 kwietnia 2024)

Zagrożenie cyberbezpieczeństwa	Odpowiadające akty prawne	Gdzie zgłosić
Phishing	Kodeks Karny - Art. 269b KK	Policja, CSIRT
Ataki ransomware	Kodeks Karny - Art. 269b KK, Ustawa o usługach płatniczych	Policja, ABW, KNF
Malware	Kodeks Karny - Art. 269b KK	Policja, CSIRT
Ataki DDoS	Kodeks Karny - Art. 269a KK	Policja, CSIRT
Wycieki danych	Ustawa o ochronie danych osobowych	UODO, Policja, GIODO
Insider Threats	Ustawa o ochronie danych osobowych	UODO, Policja
Exploity zeroday	Ustawa o krajowym systemie cyberbezpieczeństwa	Policja, CSIRT
Ataki Man-in-the-Middle (MITM)	Kodeks Karny - Art. 269b KK	Policja, CSIRT
SQL Injection	Kodeks Karny - Art. 269b KK	Policja
Cross-Site Scripting (XSS)	Kodeks Karny - Art. 269b KK	Policja
Ataki na infrastrukturę krytyczną	Rozporządzenie Rady Ministrów w sprawie usług kluczowych	CSIRT, ABW
Spoofing	Kodeks Karny - Art. 269b KK	Policja
Inżynieria społeczna	Kodeks Karny - Art. 269b KK	Policja
Cryptojacking	Kodeks Karny - Art. 269b KK	Policja, CSIRT
Bezpieczeństwo Internetu Rzeczy (IoT)	Ustawa Prawo telekomunikacyjne	UKE, Policja, CSIRT
Ataki sieciowe	Kodeks Karny - Art. 269a KK	Policja, ABW, NASK, CSIRT
Kradzież konta lub przedmiotu w grze online	Kodeks Karny - Art. 268 KK, Ustawa o świadczeniu usług drogą elektroniczną	Policja

CSIRT NASK (CERT POLSKA): pełniący rolę krajowego zespołu reagowania na incydenty komputerowe.
CSIRT GOV (ABW): odpowiedzialny za administrację rządową.
CSIRT MON (DKWOC): odpowiedzialny za sektor obronności i bezpieczeństwa państwa.

Rodzina JavaScript



Rodzina JavaScript

Nazwa	Typ	Krótki opis
JavaScript	Język	Podstawowy język skryptowy używany do tworzenia interaktywnych stron internetowych, zarówno po stronie klienta, jak i serwera.
TypeScript	Język	Nadzbiór JavaScript opracowany przez Microsoft, dodający statyczne typowanie oraz zaawansowane funkcje, takie jak interfejsy i klasy.
CoffeeScript	Język	Język programowania, który kompiluje się do JavaScript. Został zaprojektowany, aby uczynić kod bardziej zwięzłym i czytelnym.
JSX	Składnia	Rozszerzenie składni JavaScript, przypominające XML/HTML, używane głównie w React do opisywania struktury interfejsu użytkownika.
Node.js	Środowisko	Otwarte, wieloplatformowe środowisko uruchomieniowe dla JavaScript, umożliwiające uruchamianie kodu JavaScript po stronie serwera.
jQuery	Biblioteka	Lekka biblioteka JavaScript upraszczająca manipulowanie DOM, obsługę zdarzeń, animacje i komunikację AJAX.
Angular	Framework	Zaawansowany framework do tworzenia dynamicznych aplikacji webowych w TypeScript, opracowany przez Google, umożliwiający tworzenie aplikacji jednostronicowych (SPA).
React	Biblioteka	Biblioteka JavaScript opracowana przez Facebook, służąca do tworzenia interfejsów użytkownika za pomocą komponentów.
Vue.js	Framework	Progresywny framework JavaScript do budowy interfejsów użytkownika, który kładzie nacisk na modularność i łatwość integracji.
Svelte	Framework	Framework JavaScript kompilujący komponenty do wydajnego, czystego JavaScript podczas budowy, co eliminuje potrzebę ładowania frameworka w czasie wykonywania.
Ember.js	Framework	Kompleksowy framework JavaScript do tworzenia ambitnych aplikacji webowych, oferujący zestaw narzędzi i konwencji wspomagających szybki rozwój.
Backbone.js	Biblioteka	Lekka biblioteka JavaScript oferująca minimalną strukturę do aplikacji webowych, umożliwiająca tworzenie modeli z kluczami-wartości i zdarzeń niestandardowych.

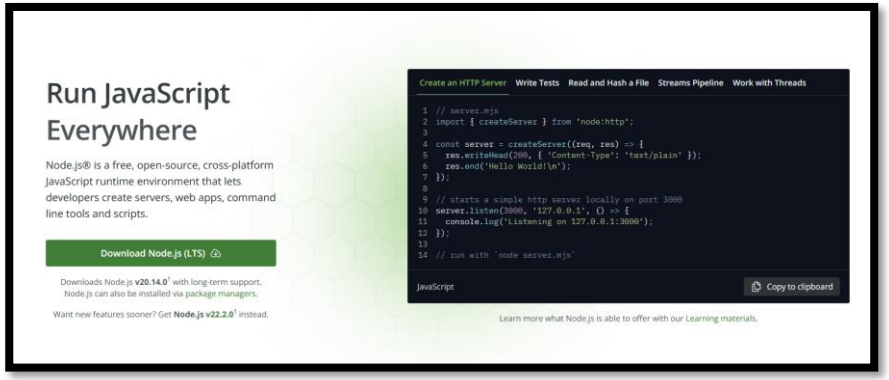
Node.js

Opis

otwarte, wieloplatformowe środowisko uruchomieniowe dla języka JavaScript, które umożliwia uruchamianie kodu JavaScript poza przeglądarką, głównie po stronie serwera.

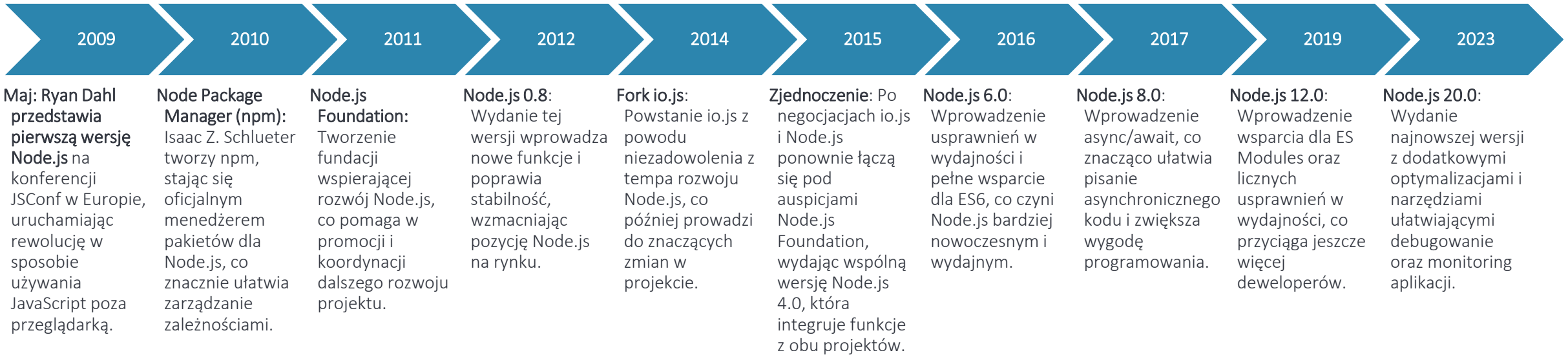
Wersja instalacyjna

https://nodejs.org/en



Historia

Historia



Zalety i wady

Zalety

Asynchroniczność i nieblokujący model I/O

Node.js korzysta z asynchronicznego modelu I/O, co pozwala na obsługę wielu równoczesnych połączeń bez blokowania wątku. To sprawia, że jest idealny do aplikacji wymagających wysokiej wydajności i skalowalności, takich jak serwery HTTP, czaty, aplikacje czasu rzeczywistego itp.

Wysoka wydajność dzięki V8

Node.js jest oparty na silniku V8 Google, który kompiluje JavaScript do natywnego kodu maszynowego, co zapewnia wysoką wydajność.

Jednowątkowość z pętlą zdarzeń

Choć Node.js działa w jednowątkowym środowisku, wykorzystuje pętlę zdarzeń do zarządzania wieloma równoczesnymi operacjami, co minimalizuje narzut związany z zarządzaniem wieloma wątkami.

Ekosystem npm

Node.js posiada wbudowany system zarządzania pakietami (npm), który jest jednym z największych repozytoriów open-source'owych pakietów. Dzięki npm, deweloperzy mają dostęp do tysięcy bibliotek, które mogą łatwo zintegrować w swoich projektach.

Uniwersalność JavaScript

Możliwość używania tego samego języka (JavaScript) po stronie klienta i serwera upraszcza proces tworzenia aplikacji i zmniejsza potrzebę nauki różnych technologii.

Szeroka adopcja i wsparcie społeczności

Duża społeczność użytkowników Node.js zapewnia wsparcie, liczne zasoby edukacyjne i szybkie rozwiązywanie problemów.

Zalety i wady

Wady

Jednowątkowość

Mimo że jednowątkowy model jest zaletą w wielu przypadkach, może być ograniczeniem dla zadań CPU. Operacje wymagające dużej mocy obliczeniowej mogą blokować pętlę zdarzeń, co wpływa na wydajność całej aplikacji.

Model programowania asynchronicznego

Chociaż asynchroniczność jest kluczową cechą Node.js, może wprowadzać złożoność do kodu. Zarządzanie callbackami może prowadzić do tzw. "callback hell", choć ten problem jest częściowo rozwiązany dzięki wprowadzeniu Promises i async/await.

Młodość ekosystemu

Mimo że ekosystem Node.js rozwija się dynamicznie, w niektórych obszarach brakuje dojrzałych i stabilnych bibliotek w porównaniu do bardziej ugruntowanych technologii, takich jak Java czy .NET.

SEO

Aplikacje oparte na Node.js (zwłaszcza SPA) mogą mieć problemy z optymalizacją pod kątem wyszukiwarek (SEO), ponieważ treść jest często generowana dynamicznie po stronie klienta.

Zarządzanie zależnościami

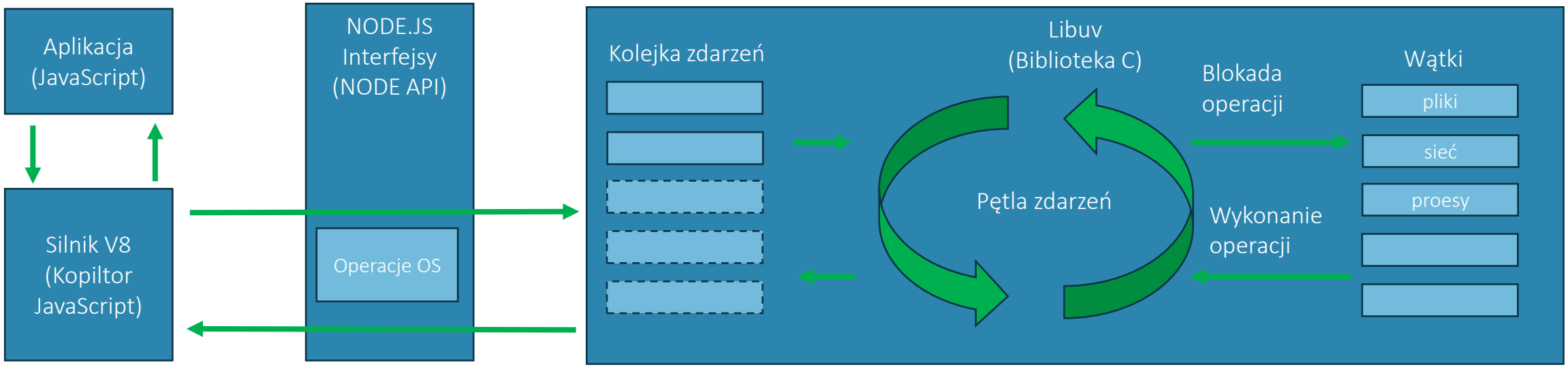
Chociaż npm ułatwia zarządzanie pakietami, jego architektura czasami prowadzi do problemów z zarządzaniem zależnościami, zwłaszcza w większych projektach (np. problemy z wersjonowaniem i zgodnością pakietów).

Bezpieczeństwo

Dynamiczna i szybka ewolucja ekosystemu może prowadzić do problemów z bezpieczeństwem, zwłaszcza jeśli deweloperzy nie zwracają uwagi na regularne aktualizacje i audyty zależności.

Architektura

Architektura



Architektura

Architektura

Aplikacja (JavaScript)

Opis: Kod napisany przez programistę, który działa na platformie Node.js. Jest to główna część, która zawiera logikę aplikacji.

Rola: Interpretuje żądania użytkowników, przetwarza dane, komunikuje się z bazami danych oraz zarządza logiką aplikacji.

Silnik V8 (Kompilator JavaScript)

Opis: Silnik JavaScript opracowany przez Google, który kompiluje kod JavaScript do natywnego kodu maszynowego.

Rola: Zapewnia szybkie i efektywne wykonywanie kodu JavaScript, co jest kluczowe dla wydajności aplikacji Node.js.

NODE.JS Interfejsy (NODE API)

Opis: Warstwa interfejsów, która umożliwia komunikację pomiędzy kodem JavaScript a niskopoziomowymi bibliotekami C, takimi jak libuv.

Rola: Umożliwia dostęp do funkcji systemowych, takich jak operacje na plikach, sieci itp., w sposób zgodny z API Node.js.

Operacje OS (System Operacyjny)

Opis: Bezpośrednie operacje wykonywane na systemie operacyjnym, takie jak operacje plikowe, sieciowe i procesowe.

Rola: Realizuje niskopoziomowe zadania zlecone przez Node.js, zarządzane przez bibliotekę libuv.

Kolejka zdarzeń

Opis: Struktura danych, w której przechowywane są zdarzenia do obsłużenia przez pętlę zdarzeń.

Rola: Organizuje zdarzenia w sposób umożliwiający ich efektywną obsługę i przetwarzanie przez pętlę zdarzeń.

Architektura

Architektura

Libuv (Biblioteka C)

Opis: Biblioteka C, która zapewnia mechanizmy wejścia/wyjścia asynchronicznego (I/O) na różnych systemach operacyjnych.

Rola: Zarządza pętlą zdarzeń, kolejką zdarzeń oraz obsługą niskopoziomowych operacji I/O, takich jak operacje na plikach, sieci, procesach itp.

Pętla zdarzeń

Opis: Mechanizm zarządzający obsługą zdarzeń i wywołaniem odpowiednich funkcji zwrotnych (callbacks).

Rola: Koordynuje wykonywanie zadań asynchronicznych, zapewniając, że zdarzenia są obsługiwane w odpowiednim czasie.

Blokada operacji

Opis: Mechanizm, który identyfikuje operacje, które mogą zablokować pętlę zdarzeń.

Rola: Przekierowuje blokujące operacje do wątków roboczych, aby nie zakłócały one płynnego działania pętli zdarzeń.

Wykonanie operacji

Opis: Proces wykonywania zadań, które nie blokują pętli zdarzeń.

Rola: Umożliwia szybkie i efektywne przetwarzanie operacji, które są gotowe do wykonania.

Wątki robocze (Worker Threads)

Opis: Dodatkowe wątki używane do wykonywania operacji blokujących i intensywnie obliczeniowych, takich jak operacje na plikach, sieci, procesach itp.

Rola: Przeprowadzają operacje, które mogłyby zablokować pętlę zdarzeń, zapewniając, że główna pętla zdarzeń pozostaje responsywna.

Instalacja

Instalacja

1. Przejdź na oficjalną stronę Node.js: nodejs.org
2. Kliknij przycisk "Download" dla zalecanej wersji LTS (Long Term Support).
3. Otwórz pobrany plik instalacyjny (np. node-v20.14.0-x64).
4. Otwórz cmd
    1. Wpisz node -v, aby sprawdzić zainstalowaną wersję Node.js.
    2. Wpisz npm -v, aby sprawdzić zainstalowaną wersję npm (Node Package Manager).

Instalacja

Pierwsze uruchomienie

1. Otwórz cmd i wpisz poniższe komendy

```
mkdir my-nodejs-project
cd my-nodejs-project
```

2. W katalogu my-nodejs-project utwórz plik app.js
3. W pliku umieść kod

```
// Importowanie modułu http
const http = require('http');

// Ustawienia serwera: adres IP i port
const hostname = '127.0.0.1';
const port = 3000;

// Tworzenie serwera
const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello, World!\n');
});

// Uruchomienie serwera
server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

4. Uruchom plik app.js za pomocą Node.js

```
cd /path/to/your/my-nodejs-project

node app.js
```

5. Otwórz przeglądarkę internetową i przejdź do adresu http://127.0.0.1:3000/.
6. Powinieneś zobaczyć stronę z napisem "Hello, World!".

Przykłady

Serwery WWW

Node.js jest często używany do tworzenia serwerów WWW, które mogą obsługiwać duże ilości jednoczesnych połączeń z minimalnym opóźnieniem.

Przykład: Tworzenie serwera HTTP za pomocą modułu http.

```
const http = require('http');
const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Przykłady

Aplikacje czasu rzeczywistego (Real-Time Applications)

Node.js jest idealnym wyborem dla aplikacji, które wymagają komunikacji w czasie rzeczywistym, takich jak czaty, gry online, aplikacje do współpracy (collaborative tools) itp.

Przykład: Tworzenie aplikacji czatu za pomocą socket.io.

```
const app = require('express')();
const http = require('http').createServer(app);
const io = require('socket.io')(http);

app.get('/', (req, res) => {
  res.sendFile(__dirname + '/index.html');
});

io.on('connection', (socket) => {
  console.log('a user connected');
  socket.on('disconnect', () => {
    console.log('user disconnected');
  });
});

http.listen(3000, () => {
  console.log('listening on *:3000');
});
```

Przykłady

API RESTful

Node.js jest szeroko stosowany do tworzenia serwisów RESTful API, które obsługują żądania HTTP i zwracają dane w formacie JSON.

Przykład: Tworzenie API za pomocą frameworka Express.

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/api', (req, res) => {
  res.json({ message: 'Hello World' });
});

app.listen(port, () => {
  console.log(`API listening at http://localhost:${port}`);
});
```

RESTful API
(Representational State Transfer Application Programming Interface)
to styl architektury, który umożliwia komunikację między systemami za pośrednictwem standardowych metod HTTP, takich jak GET i POST.
Jest szeroko stosowany w aplikacjach webowych do zarządzania zasobami, zapewniając prostotę, skalowalność i łatwość utrzymania.

Przykłady

Aplikacje mikroserwisowe

Node.js jest często używany w architekturze mikroserwisowej, gdzie aplikacja jest podzielona na małe, niezależne serwisy, które komunikują się ze sobą za pośrednictwem lekkich protokołów, takich jak HTTP/HTTPS.

Przykład: Implementacja mikroserwisów przy użyciu różnych frameworków i bibliotek, takich jak Express, Koa, Hapi.

Automatyzacja i skrypty narzędziowe

Node.js jest również używany do tworzenia skryptów automatyzujących różne zadania, takie jak przetwarzanie danych, migracje baz danych, automatyczne wdrażanie (deployment) itp.

Przykład: Używanie do automatyzacji zadań przy użyciu narzędzi takich jak Grunt, Gulp lub npm scripts.

Przykłady

Aplikacje desktopowe

Dzięki frameworkowi Electron, Node.js umożliwia tworzenie aplikacji desktopowych przy użyciu technologii webowych (HTML, CSS, JavaScript).

Przykład: worzenie aplikacji desktopowej za pomocą Electron.

```
const { app, BrowserWindow } = require('electron');

let mainWindow;

app.on('ready', () => {
  mainWindow = new BrowserWindow({ width: 800, height: 600 });
  mainWindow.loadFile('index.html');
});
```

Przykłady

IoT (Internet of Things)

Node.js jest również stosowany w aplikacjach IoT ze względu na swoją zdolność do obsługi wielu jednoczesnych połączeń i asynchronicznego przetwarzania danych.

Przykład: Tworzenie aplikacji IoT, która komunikuje się z urządzeniami za pomocą protokołów takich jak MQTT

```
const mqtt = require('mqtt');
const client = mqtt.connect('mqtt://broker.hivemq.com');

client.on('connect', () => {
  client.subscribe('myTopic', (err) => {
    if (!err) {
      client.publish('myTopic', 'Hello MQTT');
    }
  });
});

client.on('message', (topic, message) => {
  console.log(message.toString());
});
```

Angular

Opis

to platforma i framework do tworzenia dynamicznych aplikacji webowych w języku TypeScript, opracowana i utrzymywana przez Google. Angular umożliwia tworzenie aplikacji jednostronicowych (Single Page Applications - SPA), które są szybkie, responsywne i łatwe do utrzymania.

Strona projektu

https://angular.dev

Angular

TypeScript

to język programowania opracowany przez firmę Microsoft.

Jest to nadzbiór języka JavaScript, który dodaje do niego statyczne typowanie oraz dodatkowe funkcje, co sprawia, że kod jest bardziej zrozumiały, łatwiejszy do debugowania i utrzymania.

TypeScript kompiluje się do zwykłego JavaScript, dzięki czemu może być używany wszędzie tam, gdzie JavaScript jest obsługiwany – zarówno w przeglądarkach, jak i na serwerach (np. z Node.js).

Strona projektu

https://www.typescriptlang.org/

TypeScript jest nadzbiorem JavaScript, co oznacza, że każdy poprawny kod JavaScript jest również poprawnym kodem TypeScript.

Dzięki temu można stopniowo migrować istniejące projekty JavaScript do TypeScript.

Historia

Historia

Zalety i wady

Zalety

Modularność

Angular dzieli aplikacje na moduły, komponenty i serwisy, co ułatwia zarządzanie kodem, zwiększa jego ponowne użycie i poprawia organizację projektu.

Komponenty

Komponentowa architektura umożliwia łatwe tworzenie, zarządzanie i ponowne używanie komponentów, co sprzyja tworzeniu skalowalnych i łatwych do utrzymania aplikacji.

Dwukierunkowe wiązanie danych (Two-Way Data Binding)

Automatyczna synchronizacja między modelem a widokiem redukuje ilość kodu konieczną do obsługi aktualizacji interfejsu użytkownika, co przyspiesza rozwój aplikacji.

Dependency Injection (DI)

Wstrzykiwanie zależności ułatwia zarządzanie zależnościami i promuje dobre praktyki, takie jak modularność i testowalność kodu, co zwiększa elastyczność aplikacji.

CLI (Command Line Interface)

Angular CLI ułatwia tworzenie, budowanie i testowanie aplikacji, automatyzując wiele zadań programistycznych, co poprawia produktywność i spójność kodu.

TypeScript

Angular jest napisany w TypeScript, co dodaje statyczne typowanie do JavaScript, ułatwiając wykrywanie błędów na etapie kompilacji i zwiększając czytelność oraz utrzymywalność kodu.

Zalety i wady

Wady

Krzywa uczenia się

Angular ma stosunkowo stromą krzywą uczenia się ze względu na swoją złożoność i liczbę koncepcji, które trzeba opanować, takich jak TypeScript, DI, RxJS.

Słaba wydajność przy dużych aplikacjach

Może występować spadek wydajności w bardzo dużych aplikacjach, szczególnie jeśli nie są stosowane najlepsze praktyki optymalizacji.

Rozmiar pakietu

Aplikacje Angular mogą mieć większy rozmiar pakietu w porównaniu do niektórych innych frameworków, co może wpływać na czas ładowania aplikacji.

Skomplikowana konfiguracja

Chociaż Angular CLI automatyzuje wiele zadań, konfigurowanie zaawansowanych funkcji może być skomplikowane i wymagać dużej wiedzy.

Częste aktualizacje

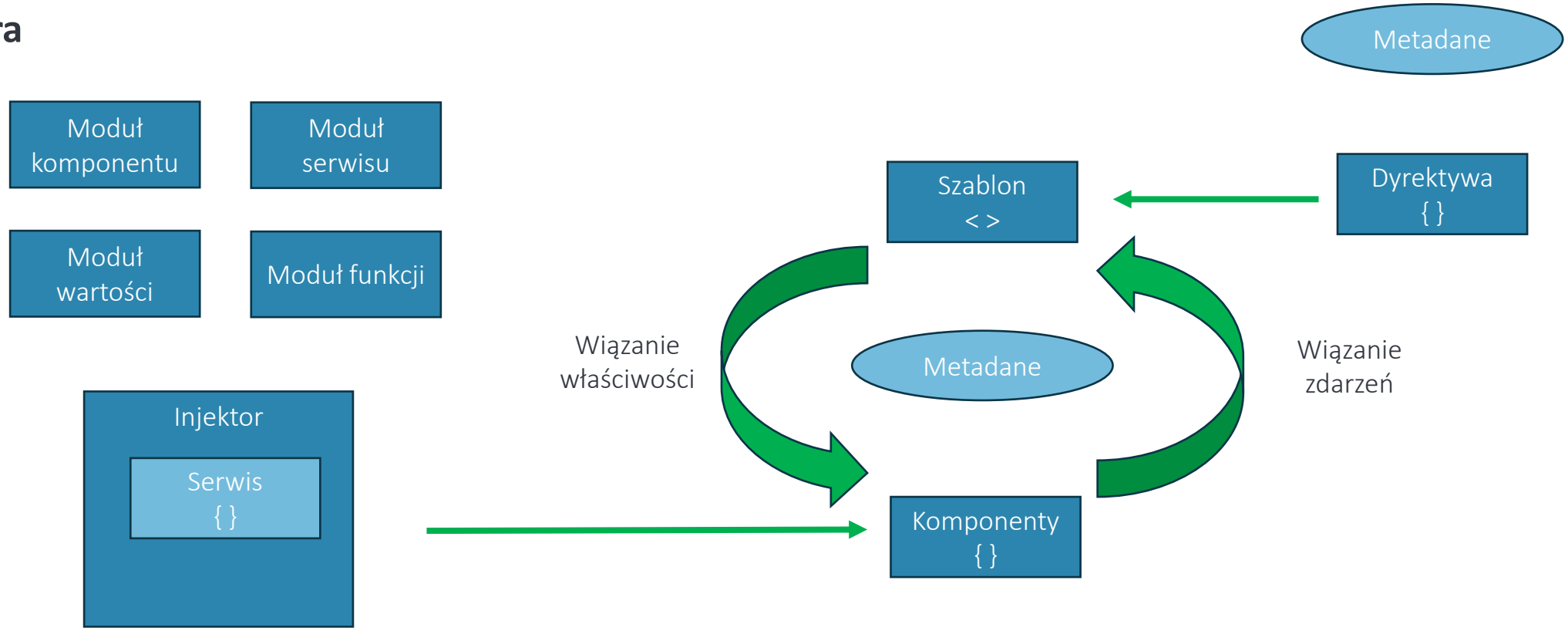
Angular jest często aktualizowany, co wymaga regularnego śledzenia zmian i potencjalnych modyfikacji kodu, aby utrzymać zgodność z nowymi wersjami.

Nadmiarowe cechy

Niektóre wbudowane funkcje i mechanizmy Angular mogą być zbędne dla prostych aplikacji, co może prowadzić do nadmiernego skomplikowania projektu.

Architektura

Architektura



Architektura

Architektura

Moduł komponentu (Module Component)

Opis: Komponenty to podstawowe jednostki interfejsu użytkownika w Angular. Każdy komponent składa się z klasy, szablonu HTML i stylów CSS.

Moduł serwisu (Module Service)

Opis: Serwisy w Angular są używane do przechowywania i zarządzania logiką biznesową, komunikacją z API oraz stanem aplikacji. Serwisy są wstrzykiwane do komponentów za pomocą mechanizmu dependency injection (DI).

Moduł wartości (Module Value)

Opis: Moduły wartości przechowują dane, które mogą być współdzielone między różnymi komponentami i serwisami w aplikacji.

Moduł funkcji (Module FN)

Opis: Moduły funkcji mogą zawierać funkcje, które są używane w różnych miejscach aplikacji. Mogą to być funkcje pomocnicze lub logika, która nie jest związana z konkretnym komponentem.

Injektor (Injector)

Opis: Injektor zarządza zależnościami w Angular, wstrzykując serwisy i inne zależności do komponentów i serwisów w aplikacji.

Architektura

Architektura

Serwis (Service)

Opis: Serwisy dostarczają funkcje i dane, które są współdzielone między różnymi komponentami w aplikacji. Mogą one zarządzać logiką biznesową, komunikacją z serwerem i stanem aplikacji.

Szablon (Template)

Opis: Szablony definiują widok komponentu za pomocą rozszerzonego języka HTML, który może zawierać dyrektywy Angular i wiązania danych.

Dyrektywa (Directive)

Opis: Dyrektywy to specjalne instrukcje w Angular, które mogą modyfikować DOM. Angular posiada wbudowane dyrektywy, takie jak *ngIf czy *ngFor, ale programiści mogą tworzyć własne.

Metadane (Metadata)

Opis: Metadane są używane do konfigurowania klas Angular, takich jak komponenty, moduły i serwisy. Dekoratory w TypeScript, takie jak @Component czy @NgModule, służą do definiowania metadanych.

Komponenty (Components)

Opis: Komponenty są podstawowymi blokami budującymi interfejs użytkownika. Każdy komponent składa się z klasy w TypeScript, szablonu HTML i opcjonalnych stylów CSS.

Instalacja

Instalacja

Instaluje się jako moduł node.js

1. Otwórz cmd
    1. Wpisz npm install -g @angular/cli
    2. Wpisz ng version
2. Utwórz nowy projekt
    1. Wpisz ng new my-angular-app
    2. ...

Przykłady

Dashboard administracyjny

Angular jest idealny do tworzenia interaktywnych paneli administracyjnych. Poniżej znajduje się prosty przykład komponentu wyświetlającego listę użytkowników.

Przykład: Komponent wyświetlający listę użytkowników.

```
export interface User { id: number; name: string; email: string; }

// user.service.ts
import { Injectable } from '@angular/core';
import { User } from './user.model';
@Injectable({ providedIn: 'root' })
export class UserService {
  private users: User[] = [
    { id: 1, name: 'John Doe', email: 'john@example.com' },
    { id: 2, name: 'Jane Doe', email: 'jane@example.com' }
  ];
  getUsers(): User[] { return this.users; }
}

// user-list.component.ts
import { Component, OnInit } from '@angular/core';
import { UserService } from './user.service';
import { User } from './user.model';
@Component({
  selector: 'app-user-list',
  template: `<ul><li *ngFor="let user of users">{{ user.name }} ({{ user.email }})</li>
})
export class UserListComponent implements OnInit {
  users: User[] = [];
  constructor(private userService: UserService) { }
  ngOnInit(): void { this.users = this.userService.getUsers(); }
}
```

Przykłady

Sklep internetowy

Angular może być używany do tworzenia sklepów internetowych. Poniżej przykład komponentu wyświetlającego listę produktów.

Przykład: Komponent wyświetlający listę produktów.

```
// product.model.ts
export interface Product { id: number; name: string; price: number; }

// product.service.ts
import { Injectable } from '@angular/core';
import { Product } from './product.model';
@Injectable({ providedIn: 'root' })
export class ProductService {
  private products: Product[] = [
    { id: 1, name: 'Laptop', price: 1000 },
    { id: 2, name: 'Phone', price: 500 }
  ];
  getProducts(): Product[] { return this.products; }
}

// product-list.component.ts
import { Component, OnInit } from '@angular/core';
import { ProductService } from './product.service';
import { Product } from './product.model';
@Component({
  selector: 'app-product-list',
  template: `<ul><li *ngFor="let product of products">{{ product.name }} - ${{ product.
})
export class ProductListComponent implements OnInit {
  products: Product[] = [];
  constructor(private productService: ProductService) { }
  ngOnInit(): void { this.products = this.productService.getProducts(); }
}
```

Przykłady

Aplikacja do czatu

Dzięki integracji z WebSocketami, Angular może być używany do tworzenia aplikacji czasu rzeczywistego, takich jak czaty.

Przykład: Komponent czatu.

WebSocket

to protokół komunikacyjny, który umożliwia dwukierunkową, pełnodupleksową komunikację między klientem a serwerem przez pojedyncze połączenie TCP.

```
// chat.service.ts
import { Injectable } from '@angular/core';
import { WebSocketSubject } from 'rxjs/webSocket';
@Injectable({ providedIn: 'root' })
export class ChatService {
  private socket$ = new WebSocketSubject('ws://localhost:8080');
  sendMessage(msg: string): void { this.socket$.next(msg); }
  getMessages() { return this.socket$.asObservable(); }
}

// chat.component.ts
import { Component, OnInit } from '@angular/core';
import { ChatService } from './chat.service';
@Component({
  selector: 'app-chat',
  template: `<div><ul><li *ngFor="let msg of messages">{{ msg }}</li></ul>
            <input [(ngModel)]="newMessage" placeholder="Type a message..." />
            <button (click)="sendMessage()">Send</button></div>`
})
export class ChatComponent implements OnInit {
  messages: string[] = [];
  newMessage: string = '';
  constructor(private chatService: ChatService) { }
  ngOnInit(): void { this.chatService.getMessages().subscribe(msg => this.messages.push(msg)); }
  sendMessage(): void { if (this.newMessage.trim()) { this.chatService.sendMessage(this.newMessage); this.newMessage = ''; } }
}
```

SPA (Single Page Application)

SPA (Single Page Application)

to rodzaj aplikacji webowej, która ładuje pojedynczy plik HTML i dynamicznie aktualizuje jego zawartość w miarę interakcji użytkownika, zamiast przeładowywać całą stronę.

Dzięki temu SPA oferują szybsze i bardziej responsywne doświadczenie użytkownika, ponieważ komunikacja z serwerem odbywa się asynchronicznie za pomocą AJAX lub WebSocketów, co pozwala na pobieranie tylko niezbędnych danych.

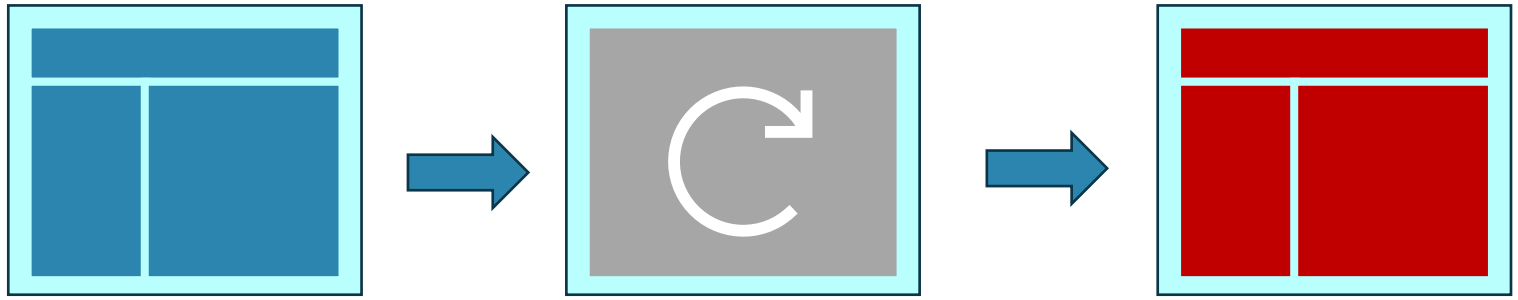
MPA (Multi-Page Application)

to tradycyjny rodzaj aplikacji webowej, w której każda zmiana lub interakcja użytkownika prowadzi do przeładowania całej strony lub załadowania nowej strony. Każda strona w MPA jest oddzielnym dokumentem HTML, który jest pobierany z serwera. Chociaż MPA może być mniej responsywna w porównaniu do SPA ze względu na częste przeładowania stron, jest ona łatwiejsza do zarządzania w kontekście SEO i może być bardziej odpowiednia dla aplikacji z dużą ilością treści.

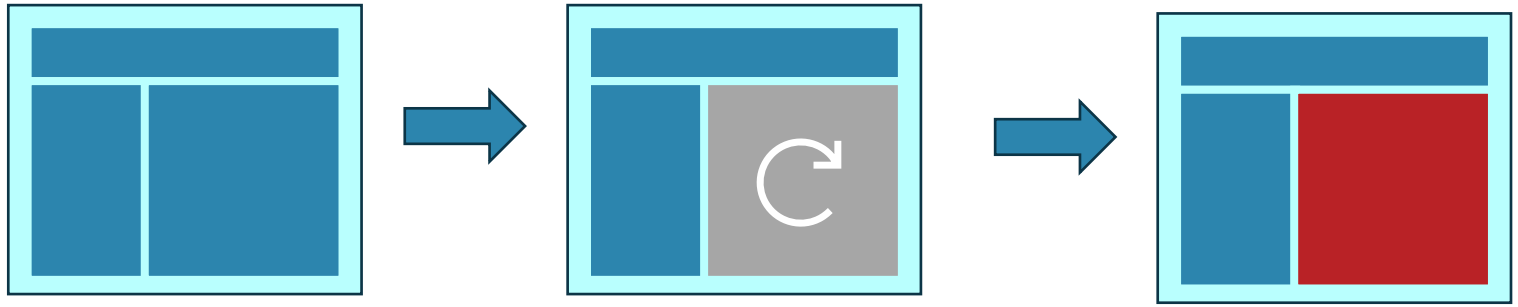
SPA (Single Page Application)

SPA vs MPA

MPA



SPA



Zalety i wady

Zalety

Szybkość i responsywność

SPA ładuje tylko jeden plik HTML i dynamicznie aktualizuje jego zawartość, co prowadzi do szybszego i bardziej płynnego działania aplikacji. Użytkownicy nie muszą czekać na pełne przeładowanie strony przy każdej interakcji.

Lepsze doświadczenie użytkownika

Przejścia między widokami są płynne, a nawigacja jest bardziej intuicyjna i podobna do natywnych aplikacji mobilnych, co zwiększa zaangażowanie użytkowników.

Mniejsza liczba żądań do serwera

Dzięki ładowaniu danych za pomocą AJAX lub WebSocketów, SPA może redukować liczbę żądań do serwera, co może prowadzić do lepszej wydajności i mniejszego obciążenia serwera.

Lepsza separacja logiki frontendowej i backendowej

SPA często korzystają z API do komunikacji z backendem, co sprzyja modularności i ułatwia rozwijanie oraz utrzymanie kodu.

Łatwiejsza rozwój i testowanie

SPA mogą być łatwiejsze do rozwijania i testowania, zwłaszcza przy użyciu nowoczesnych frameworków i narzędzi, takich jak Angular, React, czy Vue.js.

Zalety i wady

Wady

Problemy z SEO

Ponieważ SPA ładuje treść dynamicznie, indeksowanie przez wyszukiwarki może być trudniejsze. Choć istnieją techniki radzenia sobie z tym problemem (np. Server-Side Rendering, Prerendering), ich wdrożenie może być skomplikowane.

Dłuższy czas ładowania początkowego

Ładowanie całej aplikacji na początku może zająć więcej czasu, co może wpłynąć na pierwsze wrażenie użytkownika.

Wymaga bardziej zaawansowanego zarządzania stanem

Zarządzanie stanem aplikacji w SPA może być bardziej skomplikowane, zwłaszcza w przypadku większych aplikacji. Należy stosować narzędzia takie jak Redux, Vuex lub NgRx, aby efektywnie zarządzać stanem.

Bezpieczeństwo

Ponieważ SPA często polegają na API, konieczne jest szczególne zabezpieczenie komunikacji i danych. Ataki takie jak Cross-Site Scripting (XSS) i Cross-Site Request Forgery (CSRF) mogą stanowić większe ryzyko.

Obsługa przeglądarek bez JavaScript

SPA są zależne od JavaScript, więc użytkownicy z wyłączonym JavaScript mogą mieć problemy z korzystaniem z aplikacji. Dla takich użytkowników mogą być wymagane specjalne fallbacki.

Wdrożenie

Popularne frameworki i biblioteki do tworzenia SPA

Angular

Kompleksowy framework opracowany przez Google.

React

Biblioteka stworzona przez Facebook, często używana z dodatkowymi narzędziami jak Redux.

Vue.js

Progresywny framework, który jest łatwy do zrozumienia i integracji.

Zastosowanie

Obszary wykorzystania SPA

Dashboardy administracyjne

Dashboardy administracyjne wymagają dynamicznego i interaktywnego interfejsu, który może szybko aktualizować dane i oferować natychmiastową interakcję bez konieczności przeładowywania strony.

Systemy zarządzania treścią (CMS)

Systemy CMS mogą korzystać z SPA, aby oferować dynamiczne zarządzanie treścią, gdzie użytkownicy mogą edytować, publikować i zarządzać treścią w czasie rzeczywistym.

Aplikacje finansowe i analityczne

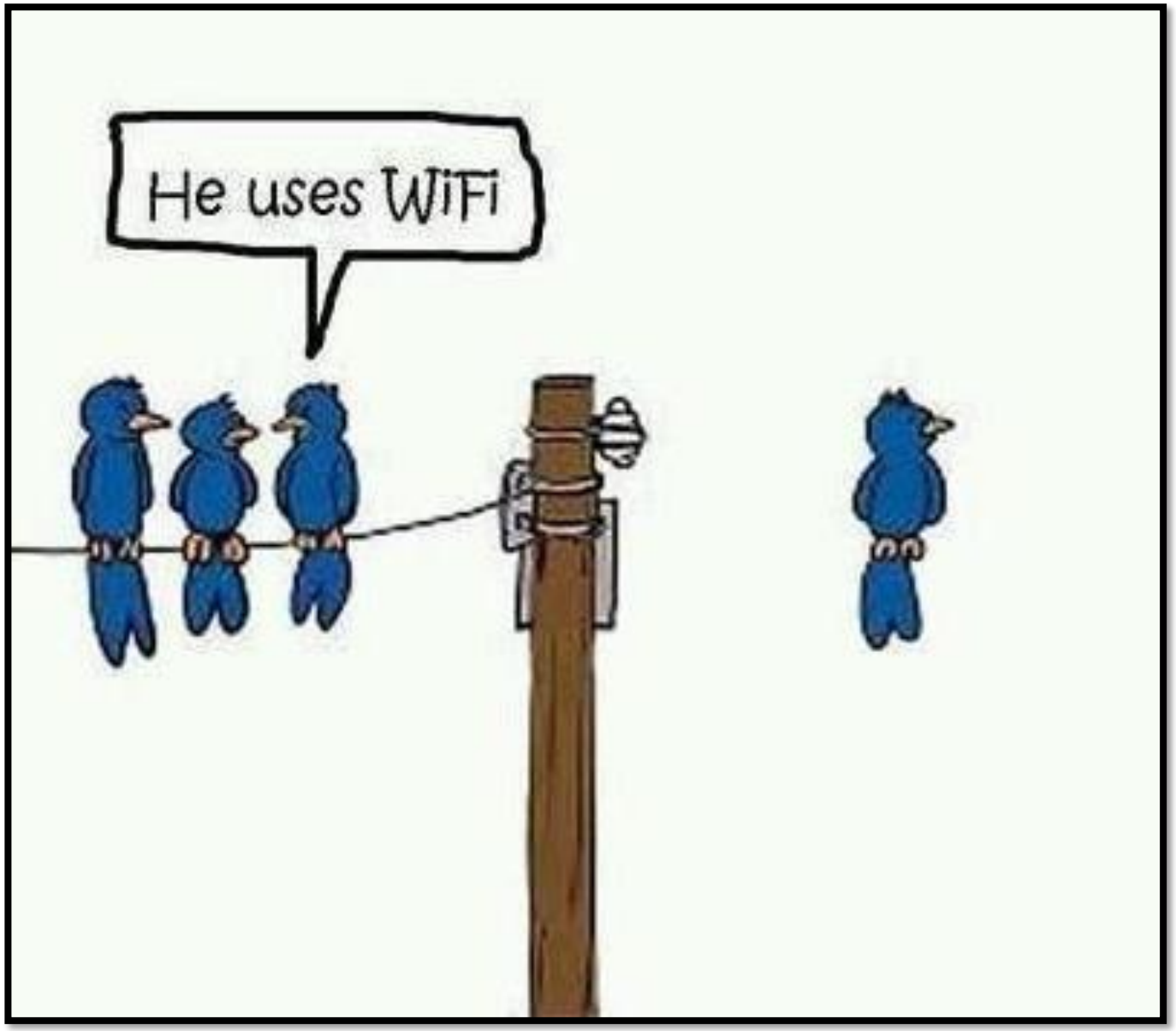
Aplikacje do analizy finansowej i danych mogą korzystać z SPA, aby oferować dynamiczne aktualizacje wykresów, tabel i raportów, co pozwala użytkownikom na interaktywne przeglądanie danych.

Aplikacje intranetowe

Aplikacje intranetowe w firmach mogą korzystać z SPA, aby zapewnić szybki dostęp do narzędzi i zasobów wewnętrznych, takich jak systemy zarządzania projektami, kalendarze, bazy wiedzy i narzędzia komunikacyjne, oferując płynne i efektywne środowisko pracy dla pracowników.

Gry przeglądarkowe

Gry przeglądarkowe korzystają z SPA, aby zapewnić płynną i interaktywną rozgrywkę bez konieczności częstego przeładowywania strony. Dzięki SPA gry mogą oferować dynamiczne aktualizacje stanu gry, interaktywne elementy i płynne animacje, co znacznie poprawia doświadczenie gracza.


He uses WiFi