

# Backend Stos MEAN vs Firebase

Node.js - Serverside Javascript

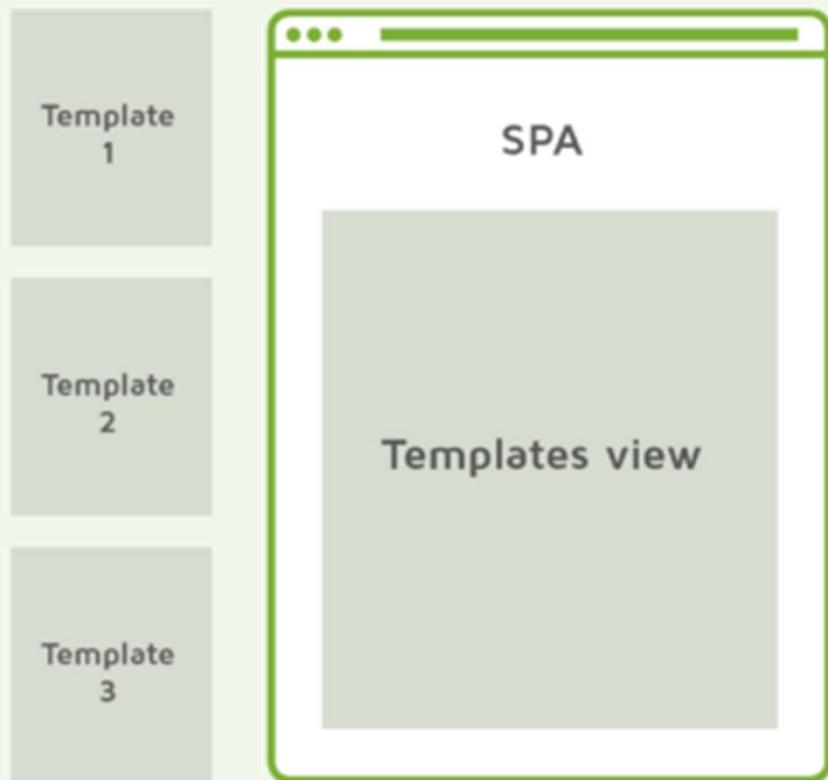
dr inż. Grzegorz Rogus



# Routing

## Nawigacja po aplikacji

Single Page Application

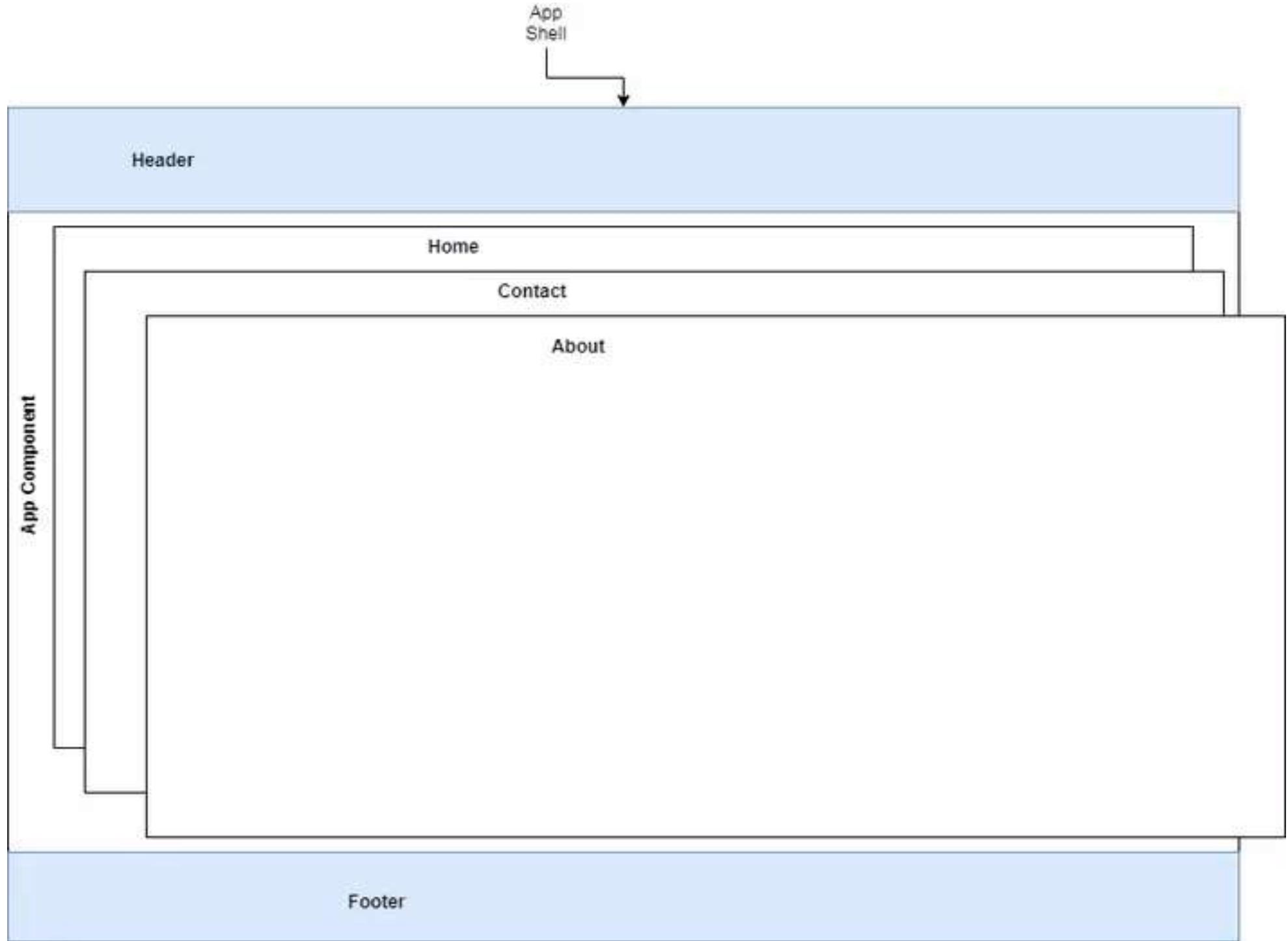


No page refresh on request

Traditional Web Application



Whole page refresh on request





# Routing

Routing pozwala zawrzeć pewne aspekty stanu aplikacji w adresie URL.

Dla aplikacji front-end jest to opcjonalne - możemy zbudować pełną aplikację bez zmiany adresu URL. Dodanie routingu pozwala jednak użytkownikowi przejść od razu do pewnych funkcji aplikacji.

Dzięki temu aplikacja jest łatwiej przenośna i dostępna dla zakładek oraz umożliwi użytkownikom dzielenie się linkami z innymi.

Routing ułatwia:

- Utrzymanie stanu aplikacji
- Wdrażanie aplikacji modułowych
- Stosowanie ról w aplikacji (niektóre role mają dostęp do określonych adresów URL)



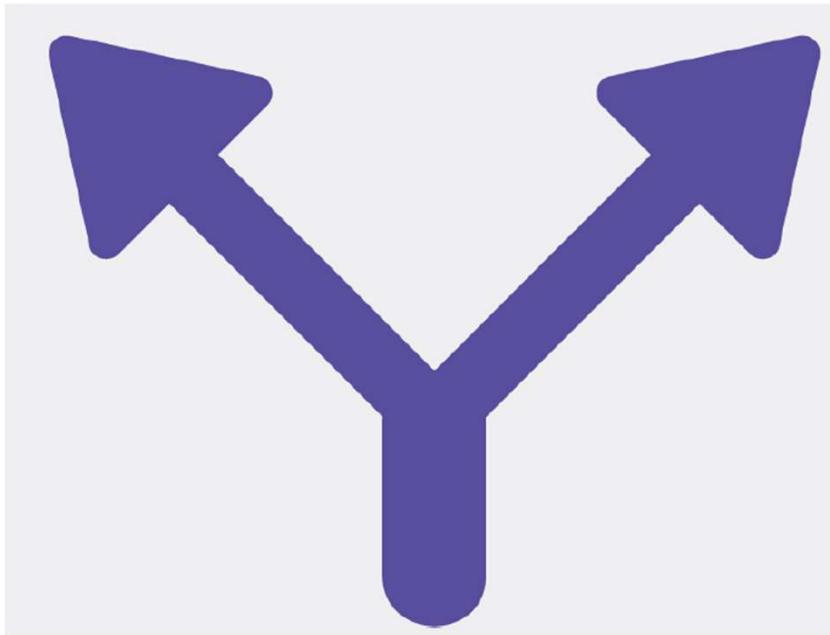
# Routing

- Zadaniem routera w ramach frameworka Angular jest nawigacja między widokami
- Router Angulara bazuje na modelu nawigacji przeglądarki
  - URL wprowadzony w pasku adresu prowadzi do wskazanego widoku
  - Kliknięcie linku w aktualnym widoku powoduje przejście do innego
  - Adres URL może zawierać parametry dla widoku
  - Przyciski Back i Forward w przeglądarce nawigują po historii
- Obszar na stronie, w którym wyświetlane mają być różne komponenty zależnie od stanu routera, wskazuje się znacznikiem  
`<router-outlet></router-outlet>`



# Routing

Składowe routingu



1. `<base href="/">`
2. `import RouterModule`
3. Konfiguracja ścieżek
4. `<router-outlet>`

# Konfiguracja routingu

CLI -> Would you like to add Angular routing? (y/N)

src/app/app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

const routes: Routes = [];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

src/app/app.component.html

```
<router-outlet></router-outlet>
```

# Definiowanie tablicy routes

```
const routes: Routes = [  
  { path: 'component-one', component: ComponentOne },  
  { path: 'component-two', component: ComponentTwo }  
];
```

## Definiowanie połączeń między trasami

```
<a routerLink="/component-one">Component One</a>
```

## Nawigacja programowo

```
this.router.navigate(['/component-one']);
```

# Deklaracja parametrów trasy

```
export const routes: Routes = [  
  { path: '', redirectTo: 'product-list', pathMatch: 'full' },  
  { path: 'product-list', component: ProductList },  
  { path: 'product-details/:id', component: ProductDetails }  
];
```

localhost:4200/szczegóły produktu/5

## Powiązanie tras z parametrami

```
<a *ngFor="let product of products"  
  [routerLink]=["'/product-details', product.id]">  
  {{ product.name }}  
</a>
```

# Kontrolowanie dostępu do lub z Route

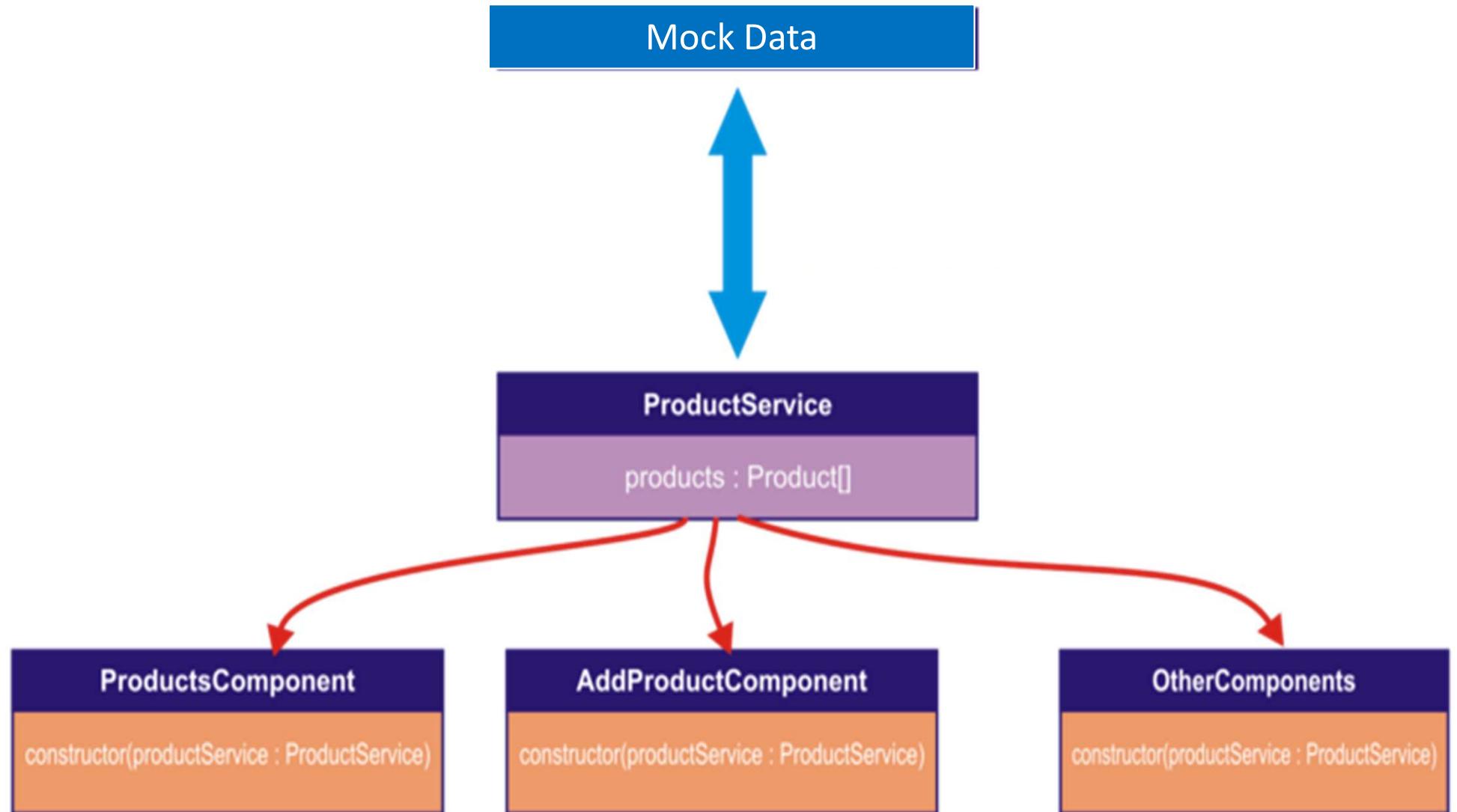
Niektóre trasy mają być dostępne tylko po zalogowaniu się użytkownika lub zaakceptowaniu Warunków.

```
const routes: Routes = [
  { path: 'home', component: HomePage },
  {
    path: 'accounts',
    component: AccountPage,
    canActivate: [LoginRouteGuard],
    canDeactivate: [SaveFormsGuard]
  }
];
```

```
import { CanActivate } from '@angular/router';
import { Injectable } from '@angular/core';
import { LoginService } from './login-service';

@Injectable()
export class LoginRouteGuard implements CanActivate {
  constructor(private loginService: LoginService) {}

  canActivate() {
    return this.loginService.isLoggedIn();
  }
}
```



# Uzycie servisu

- ng g service product

```
import { MockData } from './mock-data/mock-product-data';
import { Injectable } from '@angular/core';
import { Product } from '../models/product';

@Injectable()
export class ProductService {
  products: Product[] = [];

  constructor() {
    this.products = MockData.Products;
  }

  getProducts(): Product[] {
    return this.products;
  }

  removeProduct(product: Product) {
    let index = this.products.indexOf(product);
    if (index !== -1) {
      this.products.splice(index, 1);
    }
  }

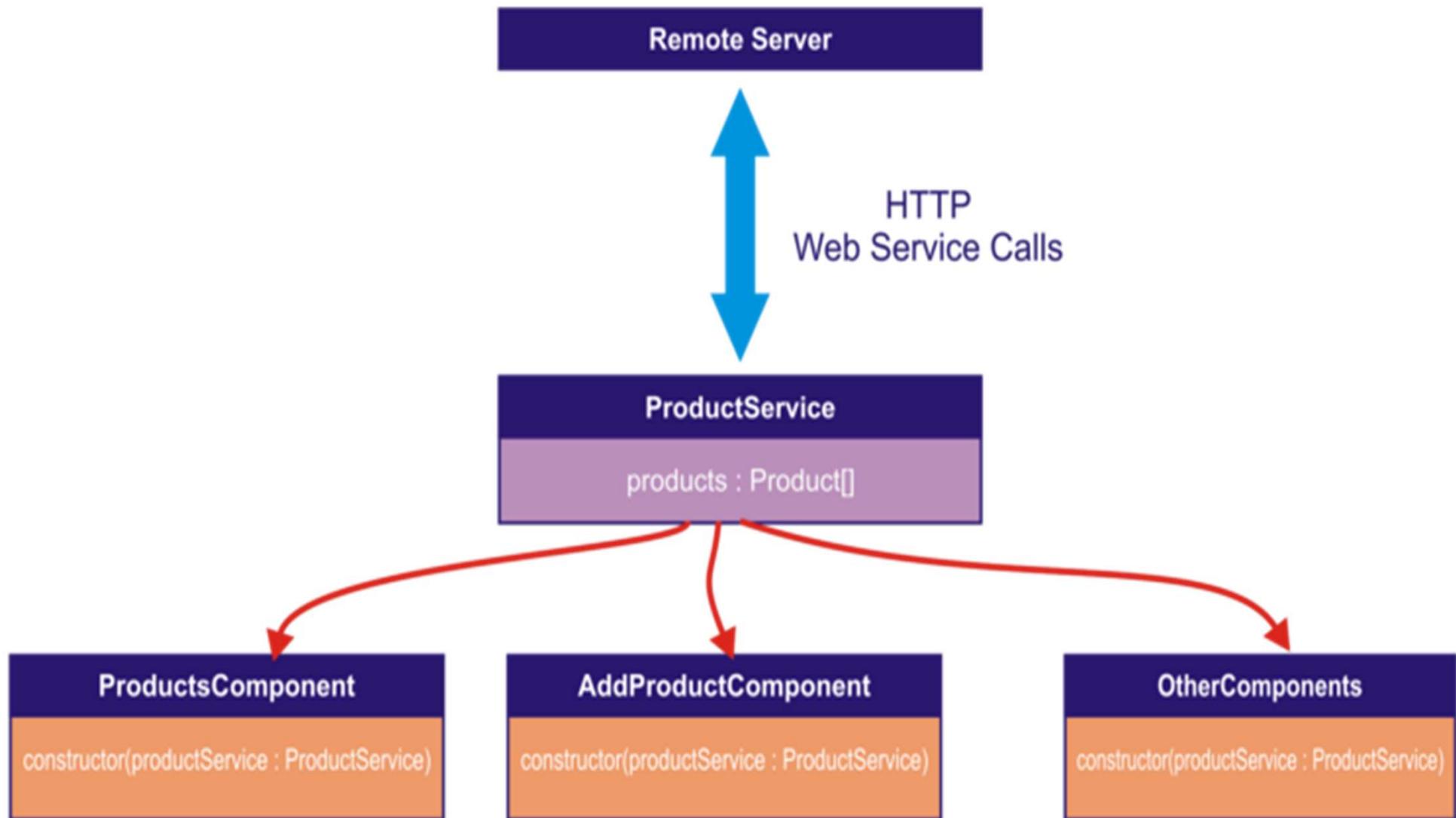
  getProduct(id: number): Product {
    return this.products.find( p => p.id === id));
  }

  addProduct(product: Product) {
    this.products.push(product);
  }
}
```

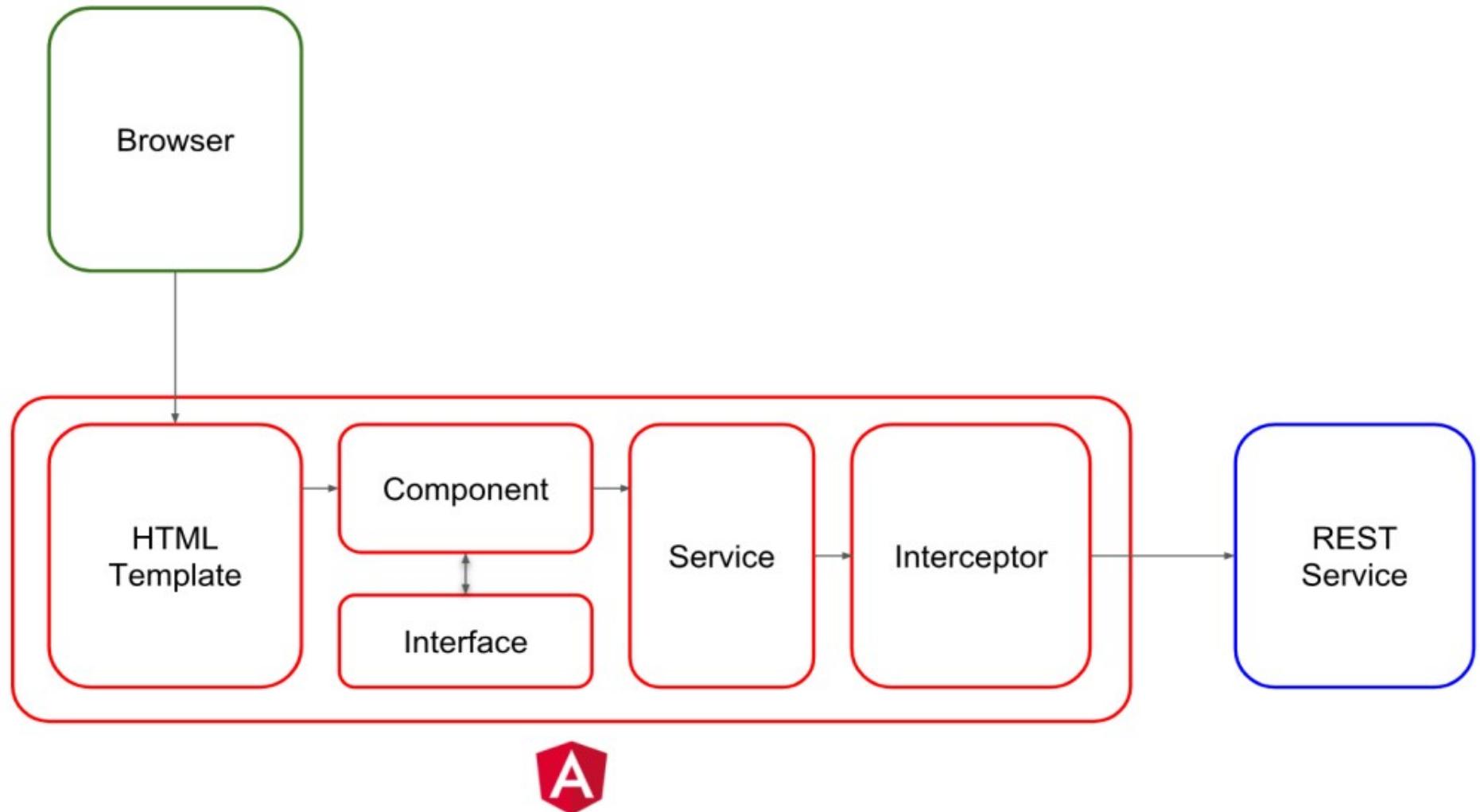
# Wywołanie usługi – przeniesienie danych do usługi

```
export class ProductsComponent implements OnInit {  
  products: Product[] = [];  
  
  constructor(public productService: ProductService) {  
    //  this.products = productService.getProducts();  
  }  
  
  ngOnInit() {  
    this.products = productService.getProducts();  
  }  
  
  deleteProduct(product: Product) {  
    this.productService.removeProduct(product);  
    this.products = this.productService.getProducts();  
  }  
}
```

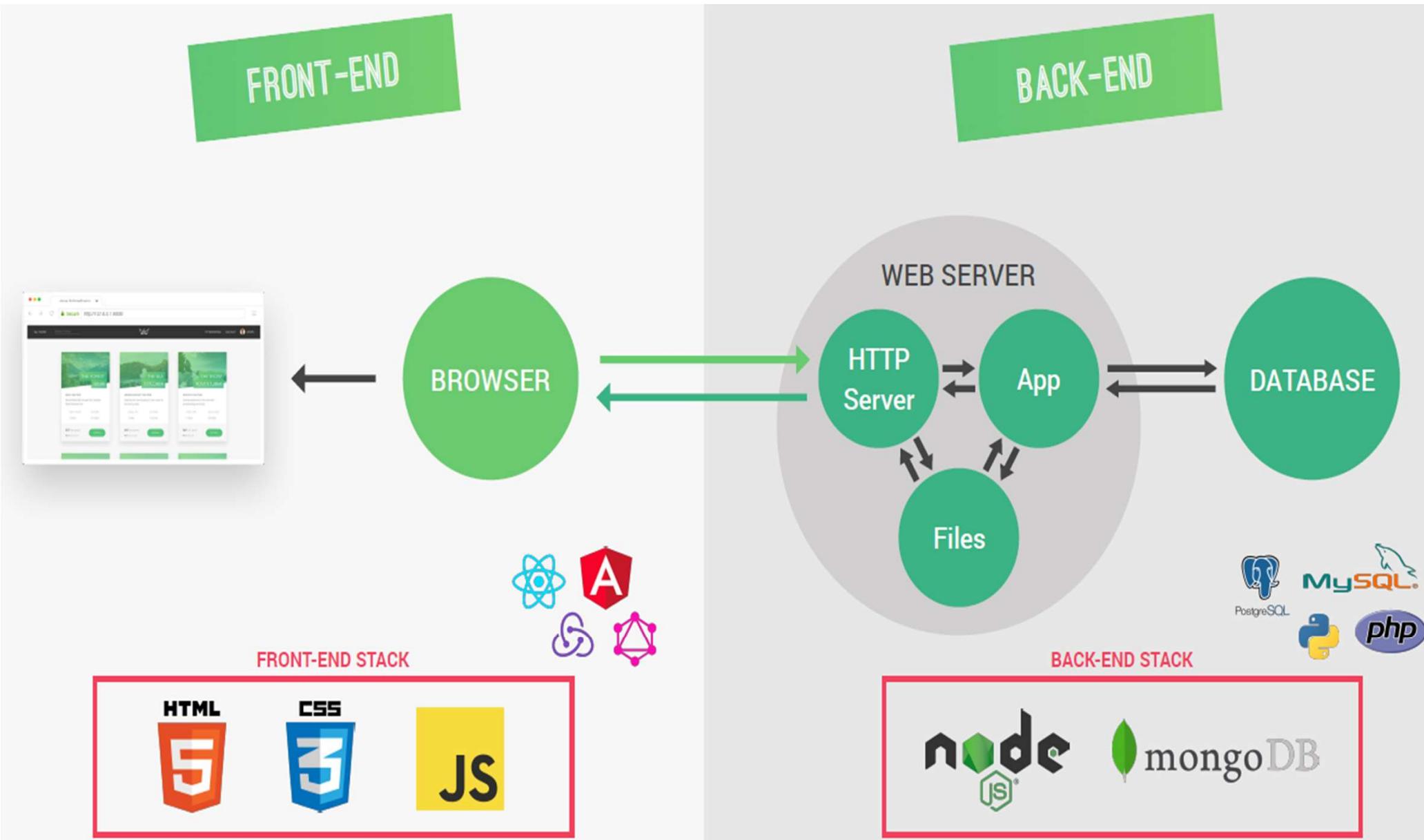




# Architektura rest API w Angularze

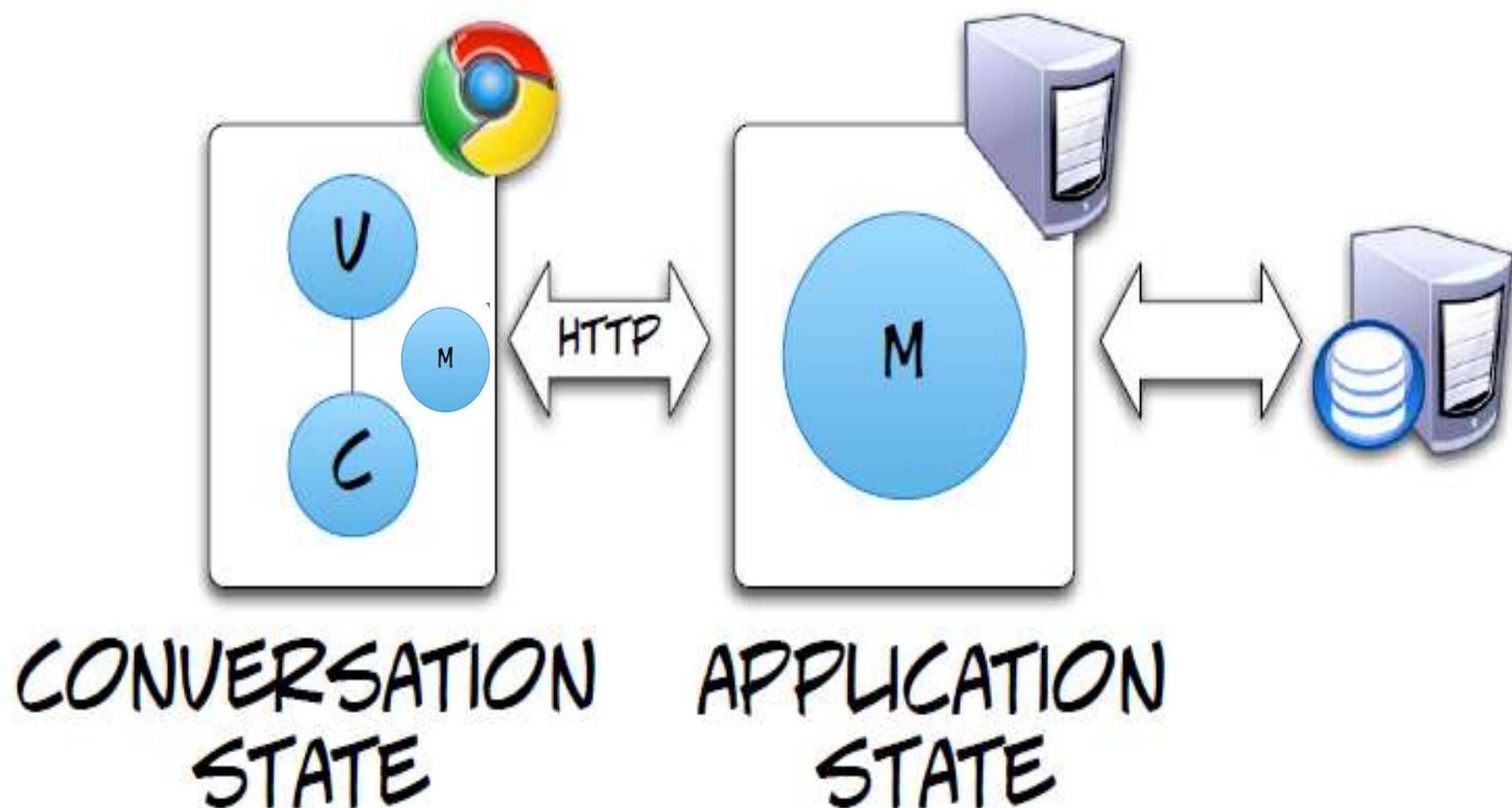


# Nasz stos technologiczny

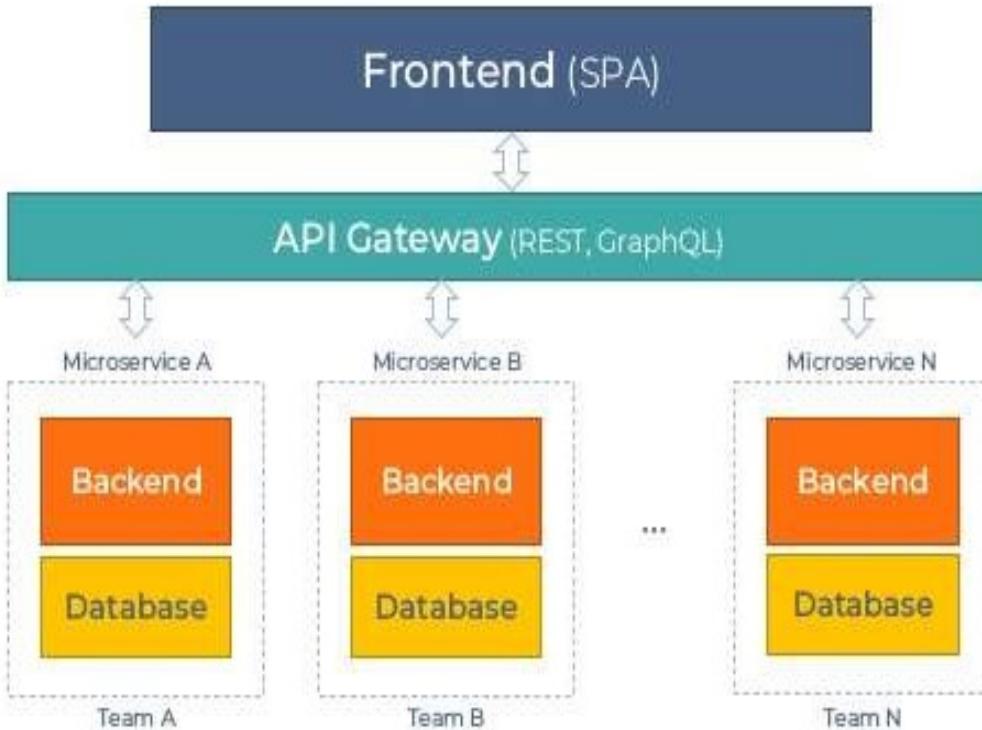


# Aplikacje Webowe dzisiaj

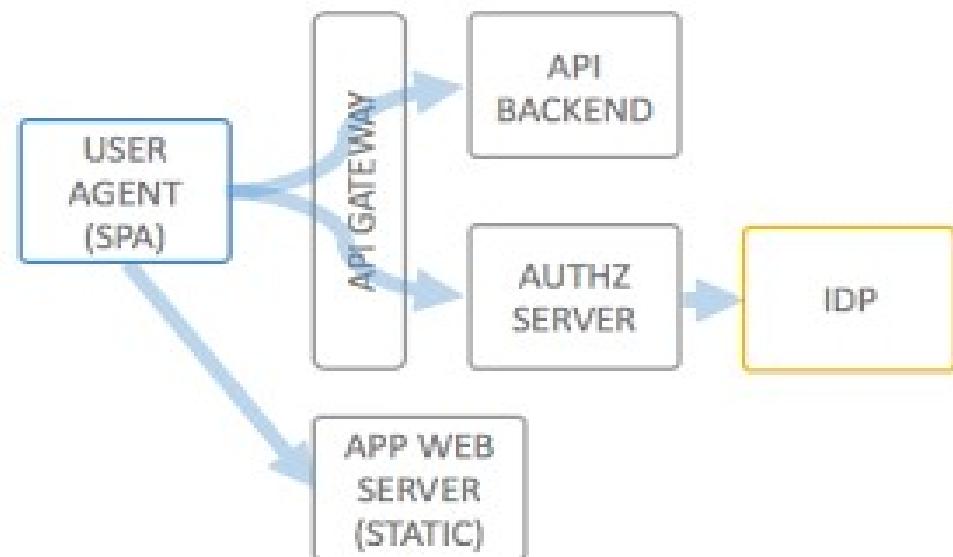
- łatwe do skalowania
- łatwa reużywalność



# Rola Backend w SPA



- autentykacja użytkownika
- dostarczanie danych
- persystencja danych w bazie danych
- współdzielona logika biznesowa



# Backend – sposoby realizacja

## Technologie



Ruby



PHP



Python



C#



## BAZY



MongoDB



MySQL



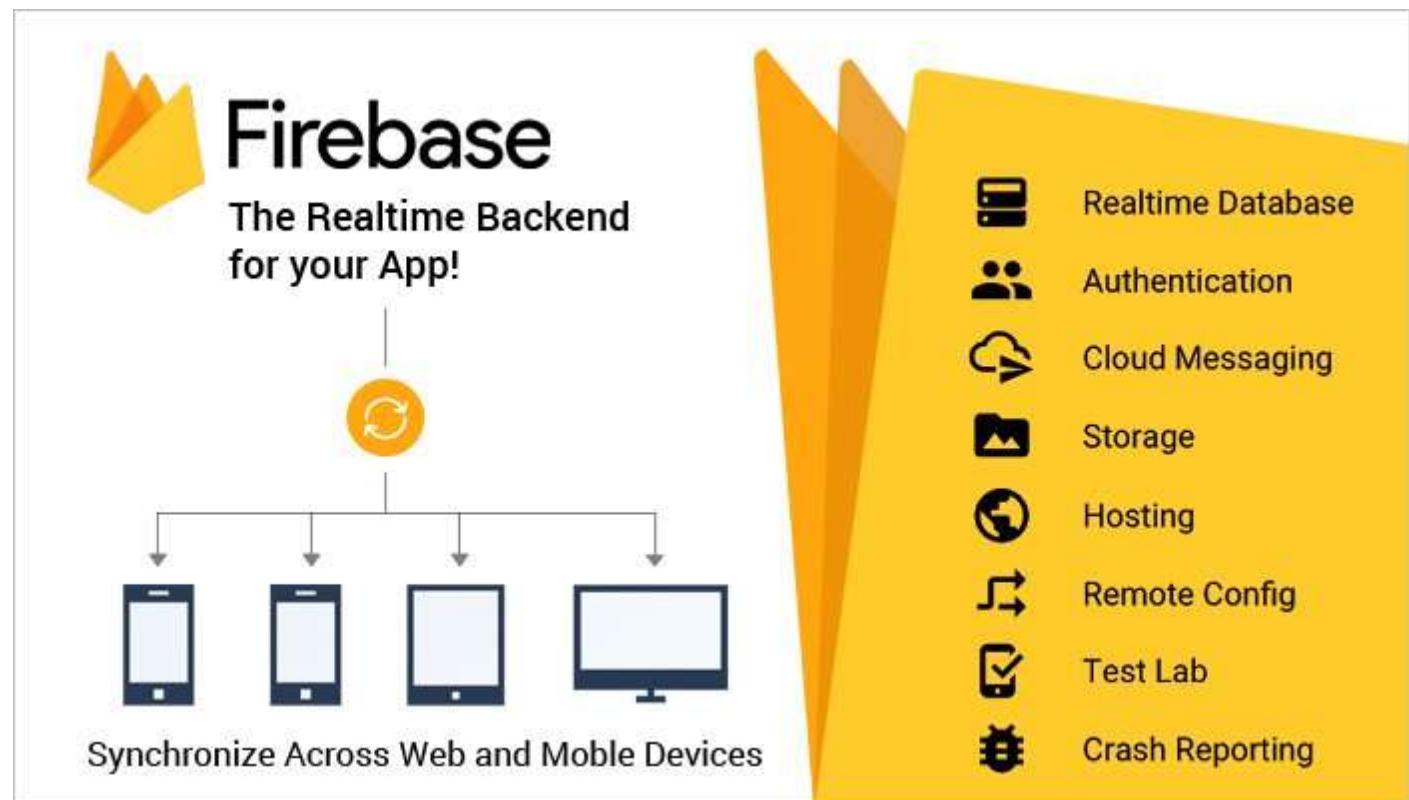
Oracle



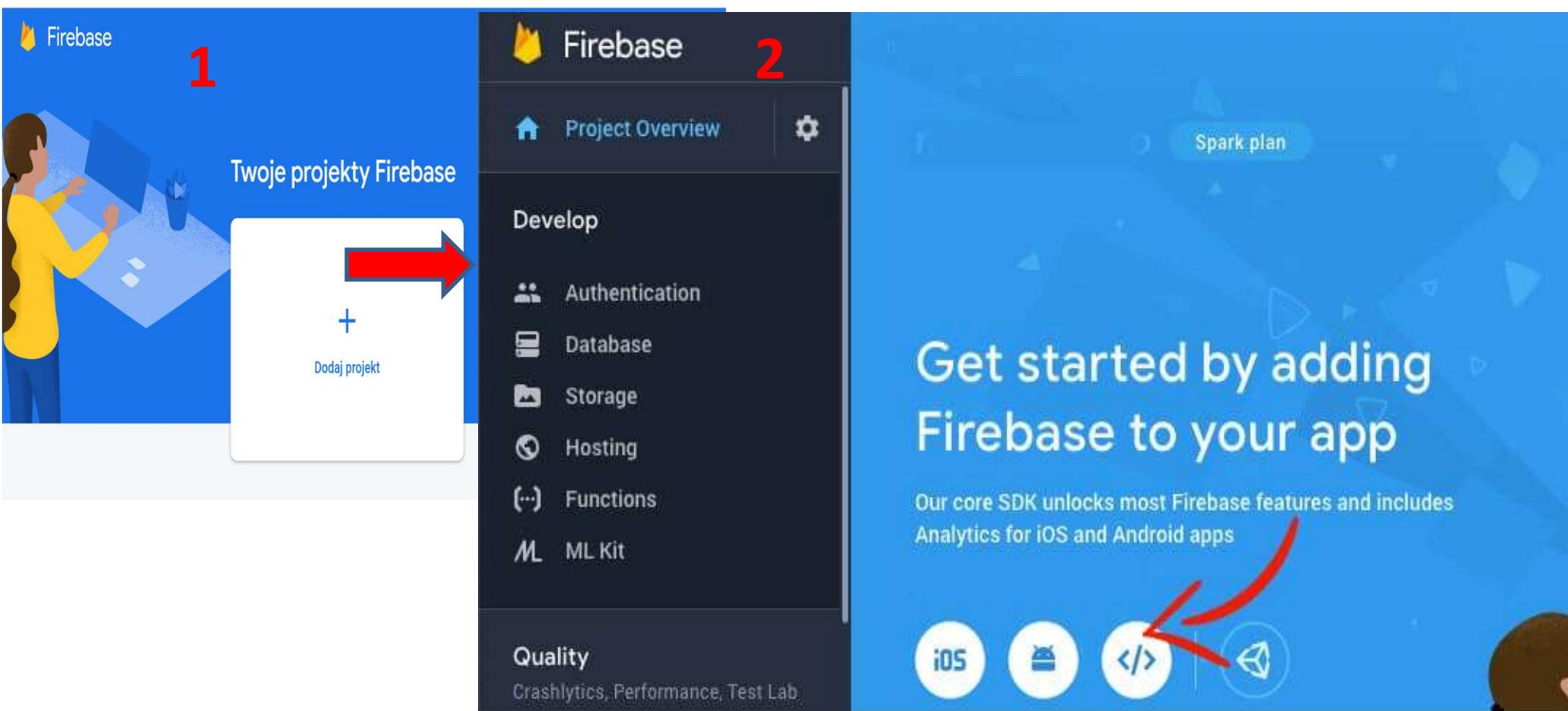
Postgresql

# Nasz wybór

- Samodzielnie stworzony serwer w oparciu o stos MEAN
- Firebase (Backend-as-a-Service — BaaS) — Google CloudPlatform.



# Firebase – konfiguracja konta



## Dodaj Firebase do swojej aplikacji internetowej

### 1 Zarejestruj aplikację

Pseudonim aplikacji 

ListaKomentarzyGR

Aplikacje internetowe

Sk

 ListaKomentarzyGR

H

Pseudonim aplikacji

ListaKomentarzyGR 

Identyfikator aplikacji 

1:745908412191:web:090f06f8653fb028f0da2a

[Połącz z witryną w Hostingu Firebase](#)

### 2 Doda

**Firebase SDK snippet**

CDN   Konfiguracja 

Zanim zaczniesz używać usług Firebase, skopiuj i wklej poniższe skrypty na końcu tagu <body>

```
const firebaseConfig = {
  apiKey: "AIzaSyBj0mFv3il1fNt7xlCBuvbeHSGxrjNKqZI",
  authDomain: "listakomentarzy-e981e.firebaseio.com",
  databaseURL: "https://listakomentarzy-e981e.firebaseio.com",
  projectId: "listakomentarzy-e981e",
  storageBucket: "listakomentarzy-e981e.appspot.com",
  messagingSenderId: "745908412191",
  appId: "1:745908412191:web:090f06f8653fb028f0da2a"
};
```

The screenshot shows the Firebase Project Overview page. On the left, there's a sidebar with 'Programowanie' and several service links: Authentication, Database (circled in red), Storage, Hosting, Functions, and ML Kit. The main area shows 'Project Overview' with tabs for 'Develop' and 'Database'. Under 'Database', there are 'Data' and 'Rules' tabs. A large callout box labeled 'Cloud Firestore' contains information about it being the next generation of the Realtime Database. Another callout box labeled 'Realtime Database' provides details about its real-time synchronization. Red arrows and numbers indicate the steps: 1 points to the 'Cloud Firestore' callout, 2 points to the 'Realtime Database' callout, and another arrow points from the 'Database' link in the sidebar to the 'Database' tab in the main area.

## Cloud Firestore

**Database**

Realtime Database  
Store and sync data in realtime across all connected clients

Cloud Firestore  
The next generation of the Realtime Database with more powerful queries and automatic scaling

lub wybierz Realtime Database



### Realtime Database

Oryginalna baza danych Firebase.  
Obsługuje synchronizację danych w czasie rzeczywistym, tak samo jak Cloud Firestore.

[Zobacz dokumentację](#)

[Więcej informacji](#)

[Utwórz bazę danych](#)

listakomentarzy-e981e

# Format danych w Real Database

items

1 + X

```
hide: false
imie: "Grzegorz"
komentarz: "ABC Grzegorz"
```

2

```
hide: false
imie: "Iza"
komentarz: "CDE Iza"
```

5

```
hide: false
imie: "Kuba"
komentarz: "test Jakuba"
```

-LuPoSm\_7D1h67NzUbBe

```
hide: true
imie: "Alicja"
komentarz: "test Alicji"
```

+ -LuPogisyawvTGBkQpj3

+ -LuPpuvmk142JmhXKk4G



## Konfigurowanie Cloud Storage

1 Ustaw reguły zabezpieczeń w  
Cloud Storage

2 Ustaw lokalizację Cloud Storage

Domyślnie reguły zezwalają na wszystkie operacje odczytu i zapisu przez uwierzytelnionych użytkowników.

Po zdefiniowaniu struktury danych **musisz stworzyć reguły, które je zabezpieczą.**[Więcej informacji](#)

## Database



Cloud Firestore ▾

Dane

Reguły

Indeksy

Wykorzystanie

Home > komentarze > ZWtNtqopMZw2...

listakomentarzy-e981e

komentarze

⋮

ZWtNtqopMZw27YMI63r2

+ Utwórz kolekcję

+ Dodaj dokument

+ Utwórz kolekcję

komentarze

ZWtNtqopMZw27YMI63r2

>

tGAI70Jx02jm3w8scmUp

+ Dodaj pole

hide: false

imie: "Grzegorz"

komentarz: "Test GR"



Project Overview



Programowanie

Authentication

Database

Storage

Hosting

Functions

ML Kit

ListaKomentarzy ▾

# Authentication

Użytkownicy

Metoda logowania

Szablony

Wykorzystanie

## Dostawcy logowania

Dostawca	Stan
E-mail/hasło	Wyłączono
Telefon	Wyłączono
Google	Wyłączono
Gry Play	Wyłączono
Game Center <small>Beta</small>	Wyłączono
Facebook	Wyłączono



Wyszukaj według adresu e-mail, numeru telefonu lub identyfikatora UID użytkownika

**Dodaj użytkownika**



Identyfikator

Dostawcy

Utworzono

Zalogowano

Identyfikator UID użytkownika ↑

Dodaj użytkownika, używając adresu e-mail / hasła

E-mail

rogus@agh.edu.pl

Hasło

Test123

# Angular – konfiguracja dla Firebase

1. `npm install firebase @angular/fire --save`

2.

```
export const environment = {  
  production: true,  
  firebase: {  
    apiKey: "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",  
    authDomain: "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",  
    databaseURL: "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",  
    projectId: "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",  
    storageBucket: "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",  
    messagingSenderId: "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"  
  }  
};
```

**environment.ts**



```
<script src="https://www.gstatic.com/firebasejs/4.9.0.firebaseio.js"></script>  
<script>  
  // Initialize Firebase  
  // TODO: Replace with your project's customized code snippet  
  var config = {  
    apiKey: "<API_KEY>",  
    authDomain: "<PROJECT_ID>.firebaseapp.com",  
    databaseURL: "https://<DATABASE_NAME>.firebaseio.com",  
    storageBucket: "<BUCKET>.appspot.com",  
    messagingSenderId: "<SENDER_ID>",  
  };  
  firebase.initializeApp(config);  
</script>
```

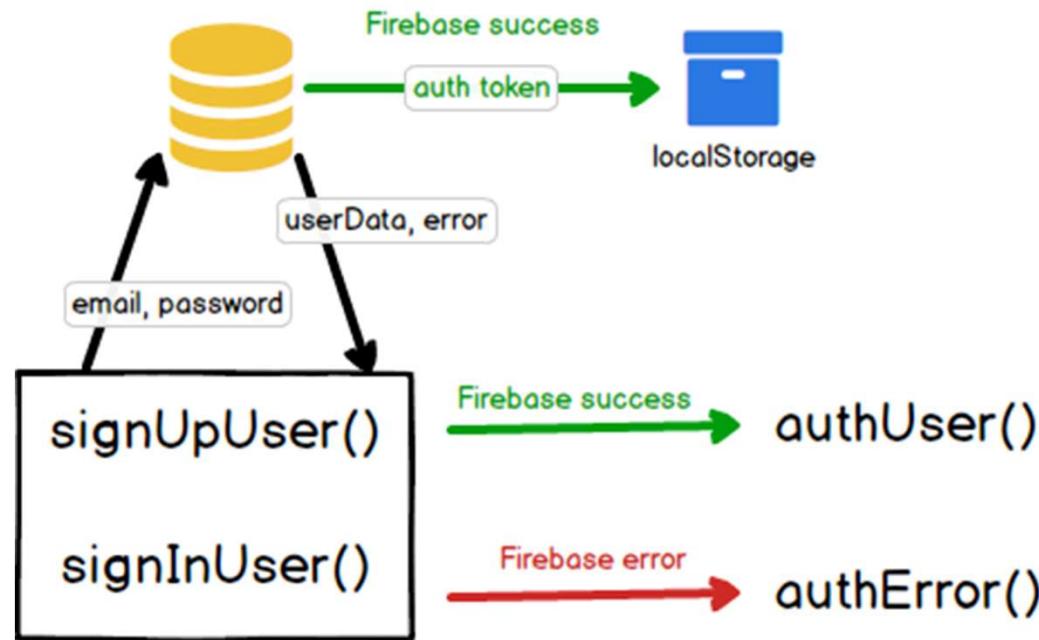
3.

```
import { AngularFireModule } from "@angular/fire";  
import { AngularFirestoreModule } from "@angular/fire/auth";  
import { AngularFirestoreModule } from '@angular/fire/firestore';  
import { AngularFireDatabaseModule } from '@angular/fire/database';  
import { environment } from './environments/environment';
```

**app.module.ts**

```
imports: [  
  AngularFireModule.initializeApp(environment.firebaseio),  
  AngularFireAuthModule, // do obsługi autentykacji  
  AngularFirestoreModule, // do obsługi baz danych  
  AngularFireDatabaseModule // do obsługi baz danych  
],
```

# Angular – autentykacja z Firebase



```
import { AngularFireAuth } from "@angular/fire/auth";
.....
constructor(public afAuth: AngularFireAuth)
```

```
SignIn() {
  return this.afAuth.auth.signInWithEmailAndPassword(email, password)
    .then((result) => {
      localStorage.setItem('user', JSON.stringify(result));
    })
    .catch((error) => {
      console.log(error.message);
    });
}
```

```
SignUpUser (email, password) {
  return this.afAuth.auth.createUserWithEmailAndPassword(email, password)
    .then((result) => {
      /* Np. wyslij mail w celu weryfikacji */
    })
    .catch((error) => {
      window.alert(error.message);
    });
}
```

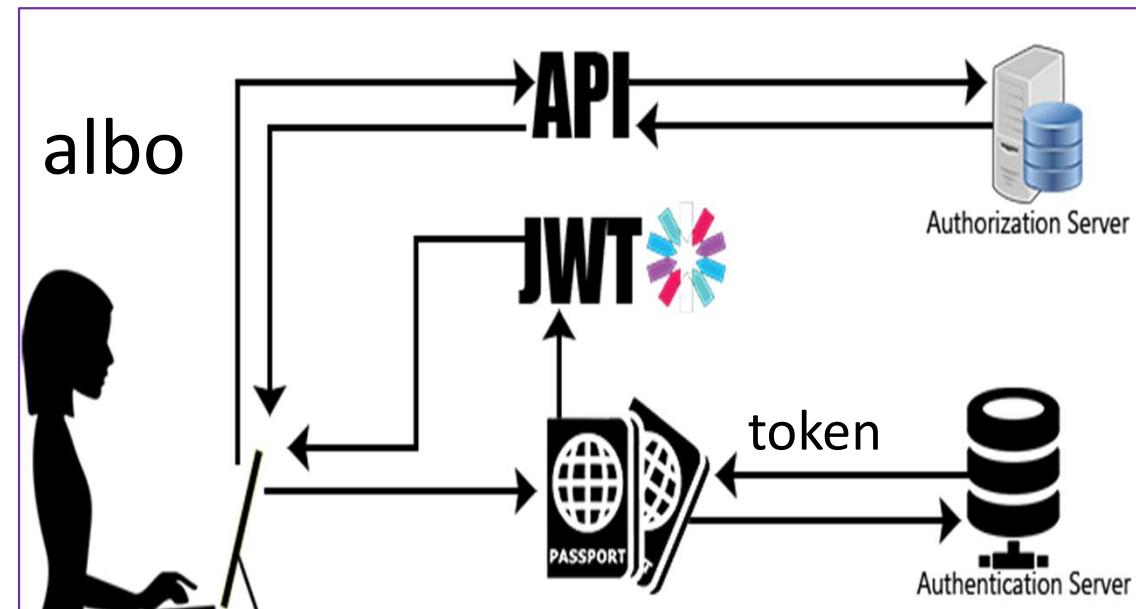
# Angular - Autoryzacja

```
CanRead(user: User): boolean {  
  const allowed = [,admin', ,edytor', reader'];  
  return this.checkAuthorization(user,allowed);  
}
```

```
CanEdit(user: User): boolean {  
  const allowed = [,admin', ,edytor'];  
  return this.checkAuthorization(user,allowed);  
}
```

```
CanDelete(user: User): boolean {  
  const allowed = [,admin'];  
  return this.checkAuthorization(user,allowed);  
}
```

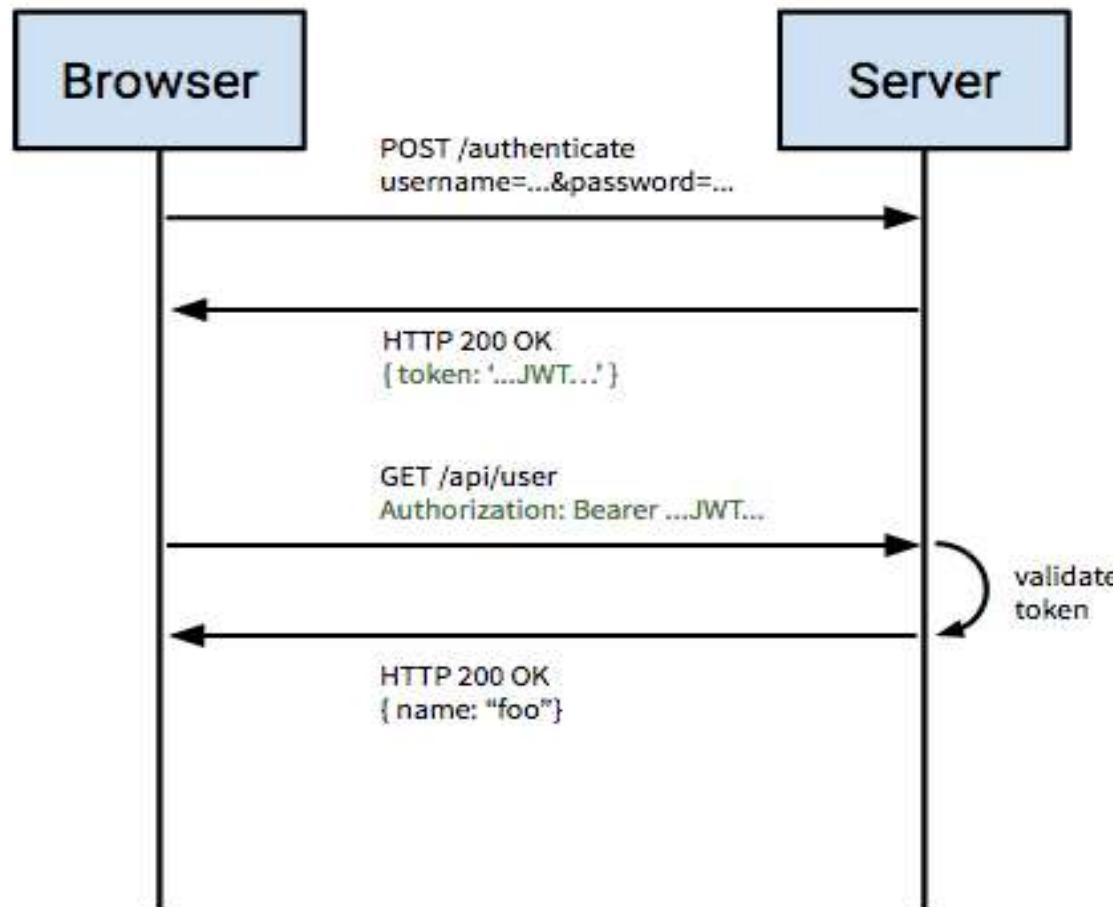
```
private checkAuthorization (user: User, allowedRoles: string[]): boolean {  
  if (!user) return false;  
  for (const role of allowedRoles) {  
    if (user.roles[role]) { return true; }  
  }  
}
```



```
Data: User = { uid: user.id  
               email: user.email;  
               roles: { admin: true;  
                         reader: true }  
 }
```

# Rest API - autoryzacja

## Modern Token-Based Auth



# Firebase – realtime database

```
constructor(private db: AngularFireDatabase) {}

getComments(): Observable<any> {
    return this.db.list(PATH_KOMENTARZE).valueChanges();  }

getComment (id: number): Observable<any> {
    return this.db.object(PATH_KOMENTARZE + id).valueChanges();  }

addComment(item: Comment) {
    this.db.list(PATH_KOMENTARZE).set(String(item.id), item);  }

updateComment(item: Comment) {
    this.db.object(PATH_KOMENTARZE + item.id).update(item);  }

deleteAllComments(item: Comment) {
    this.db.object(PATH_KOMENTARZE + item.id).remove();  }
```

# Firebase – FireStore database

```
deleteComment(item_id){  
    return  
this.db.collection('comments').doc(item_id).delete(); }
```

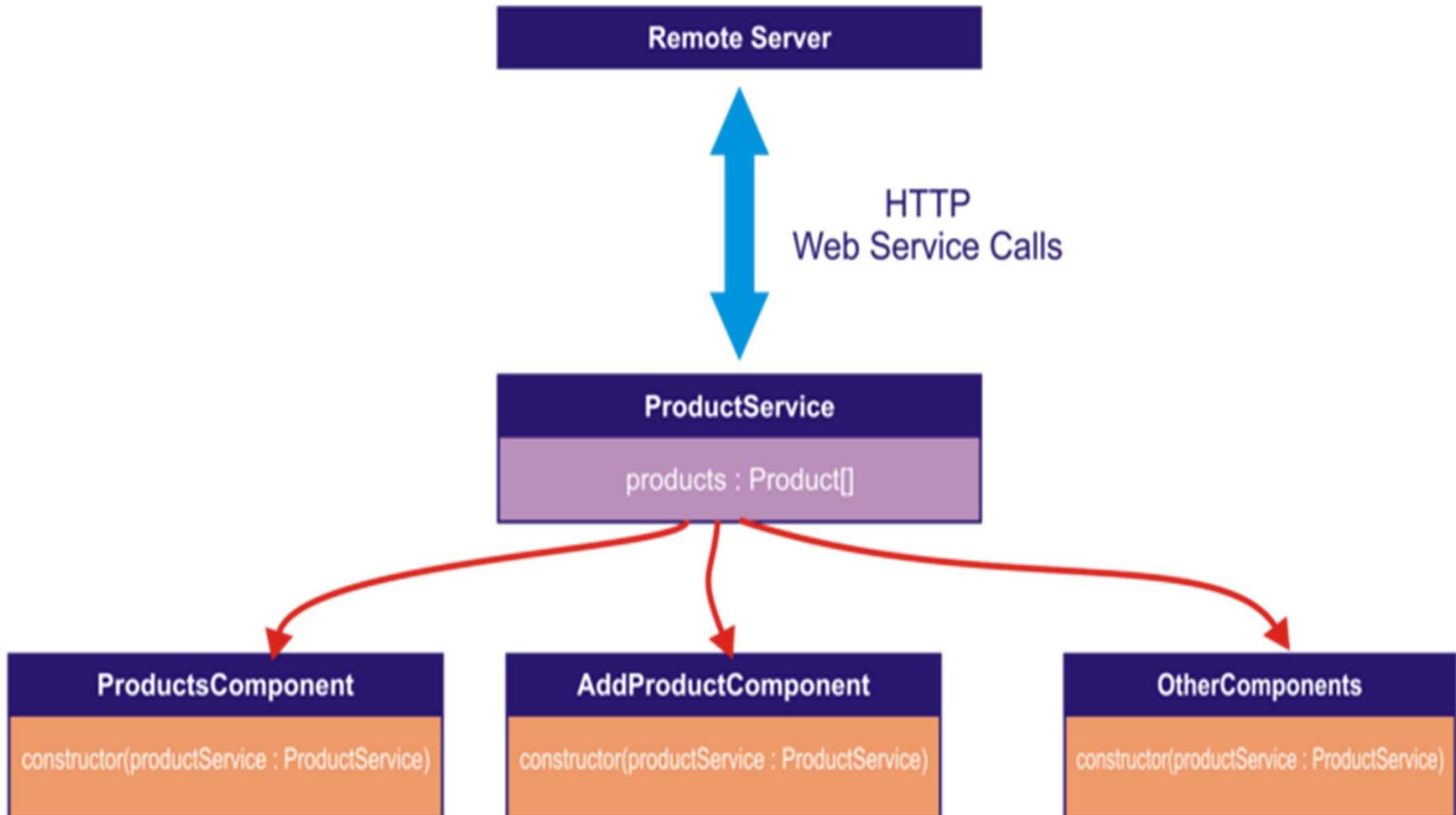
```
updateComment() {  
    this.db.collection('comments').doc('my-custom-id').set({'imie': this.imie,  
        'komentarz': this.komentarz, 'hide': this.hide}); }
```

```
addComment(){  
    this.db.collection('comments').add({'imie': this.imie, 'komentarz': this.komentarz});
```

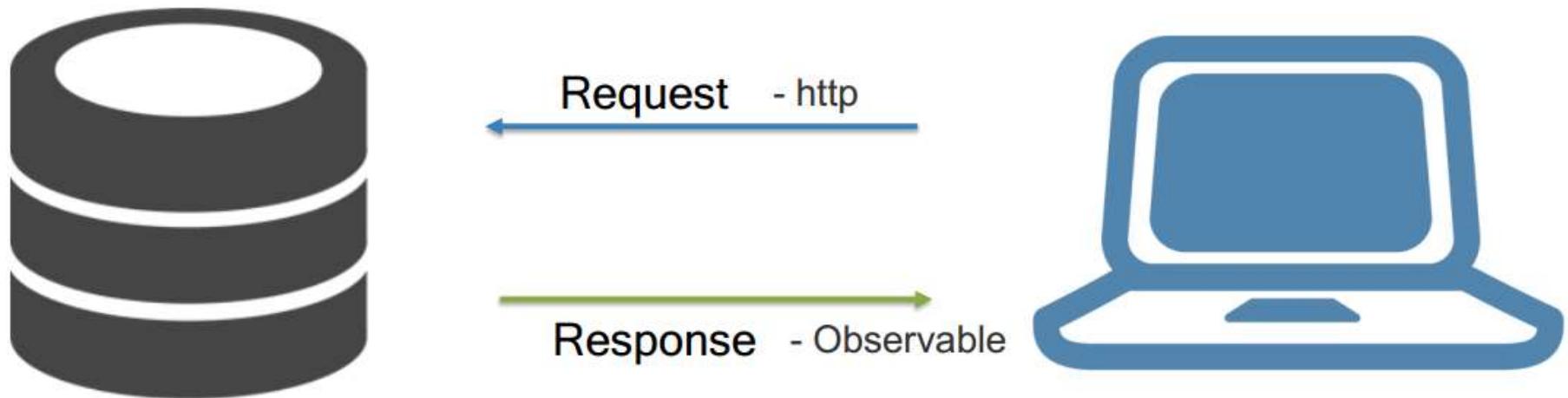
```
getComment (commentId) {  
    this.wynik = this.db.doc('comments/'+commentId);  
    this.comment = this.wynik.valueChanges(); }
```

```
getComments(){  
    this.afs.collection('comments').snapshotChanges().subscribe( snapshots => {  
        resolve(snapshots) })  
}
```

# Angular – usługi asynchroniczne



# HTTP

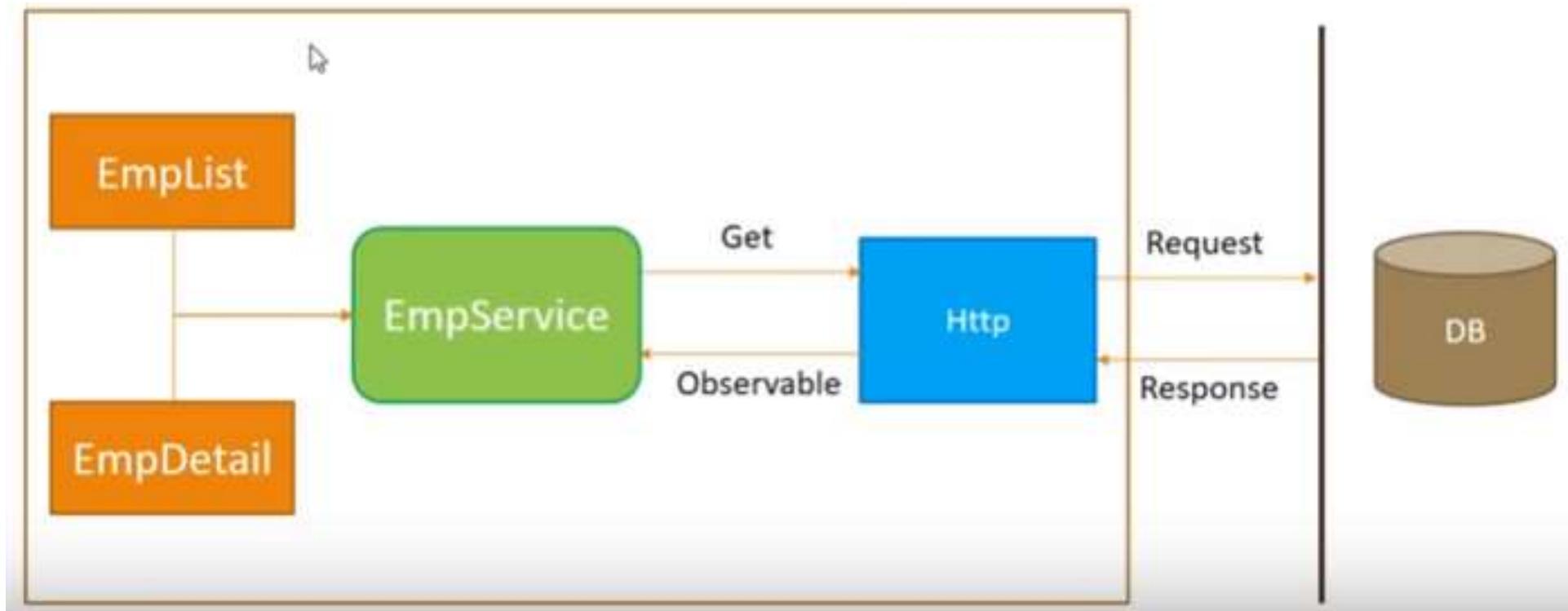


- Dodajemy do projektu usługę HTTP (HttpClient):

Angular korzysta z usługi HttpClient do komunikacji ze zdalnym serwerem poprzez protokół HTTP

- Aby udostępnić usługę HTTP w całej aplikacji należy:

- otworzyć główny moduł AppModule
- zimportować HttpClientModule z modułu @angular/common/http
- dodać go do tablicy @NgModule.imports



# Programowanie Reaktywne

- Asynchroniczne strumienie danych
- Wszystko jest strumieniem danych

zdarzenia generują dane

Dane z serwera

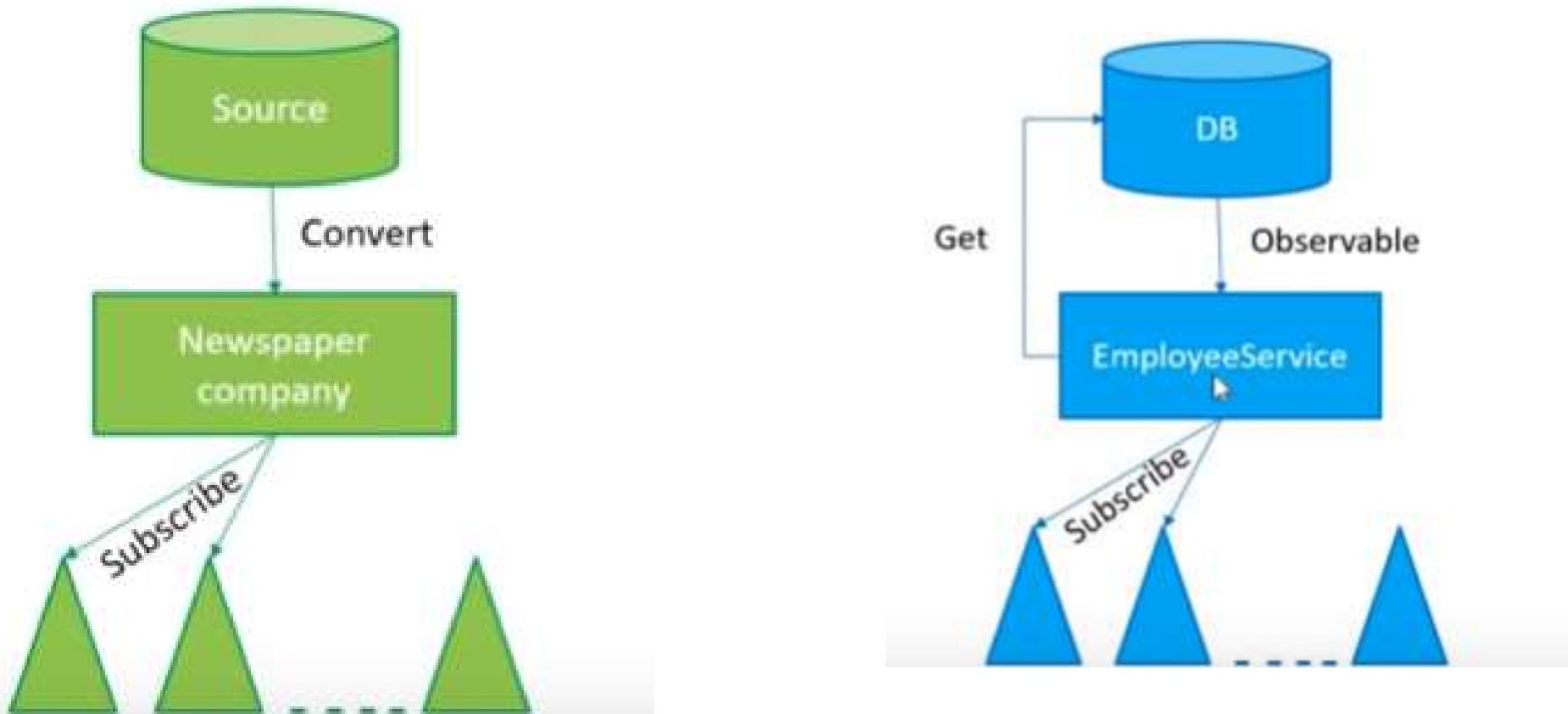
Dane z formularzy

# PROGRAMOWANIE REAKTYWNE

- **Programowanie reaktywne jest to programowanie asynchroniczne oparte na obserwacji strumienia danych.**
- Generyczna implementacja wzorca projektowego Obserwator
- Mamy strumienie danych (Observable), które za pomocą wbudowanych operatorów przekształcamy w strumień o interesujących nas własnościach (łączenie, map, flatMap, fold, filter, opóźnienie, wątek ...)
- Ktoś kiedyś zasubskrybuje się na nasz strumień i będzie dostawać powiadomienia o pojawieniu się nowych wartości

# Observables

Reprezentuje ideę przyszłego zbioru wartości lub wydarzeń (strumień danych/wydarzeń)



Obiekty typu Observable reprezentują strumień wartości, które zostaną wyemitowane w późniejszym czasie

# Najważniejsze pojęcia w nowym podejściu

- **Observable** - reprezentuje sekwencję wartości w czasie, która może być obserwowana
- **Observer** - obiekt który otrzymuje dane z Observable
- **Subscription** - wynik subskrybowania, służy głównie do zamknięcia/zatrzymywania subskrypcji
- **Operators** - zbiór metod dla Observable, najczęściej zwracające nowe Observable (np. *map*, *filter*)

# Podstawy Observable

```
var range = Rx.Observable.range(1, 3); // 1, 2, 3
```

```
var range = range.subscribe(  
    function(value) {},  
    function(error) {},  
    function() {}  
);
```

The word "optional" is positioned to the right of the three empty function bodies. Three red arrows point from this word towards each of the three functions, indicating that they are optional parameters.

Podłączenie się do strumienia jest banalnie proste.  
Observable posiada metodę **subscribe**, do której  
parametry możemy przekazać na dwa sposoby:

# Podstawy Observable

```
var obs = ...;
```

```
// query?
```

gdy sukcesem odbierzemy  
wartość ze strumienia

```
var sub = obs.Subscribe(  
    onNext : v => DoSomething(v),  
    onError : e => HandleError(e),  
    onCompleted : () => HandleDone);
```

gdy w strumieniu wystąpi error

gdy observer otrzyma ostatnią wartość ze  
strumienia (wypstryka się z danych)

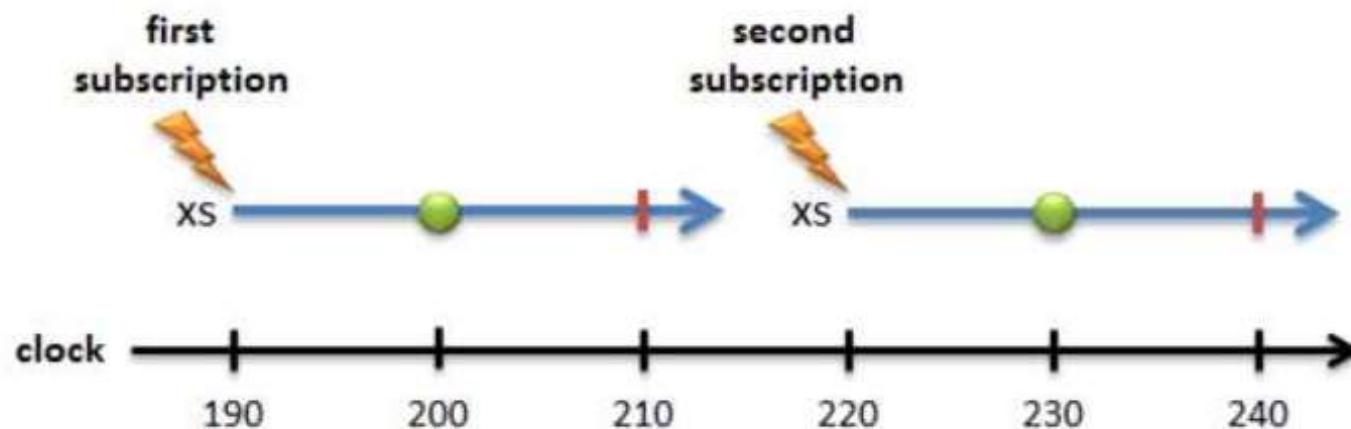
# Źródła danych

	Pojedyncza wartość	Wiele wartości
Synchronicznie	<pre>const value = 42;  Console.log(value);</pre>	<pre>var values = [1,2,3,4];  values.forEach( value =&gt; {     console.log(value); });</pre>
Asynchronicznie	<pre>const asyncValue = Promise.resolve(42);  asyncValue.then(value =&gt; {     console.log(value); });</pre>	<pre>let val\$= Observable.of(1,2,3,4);  val\$.subscribe(val\$ =&gt; {     console.log(val\$); });</pre>

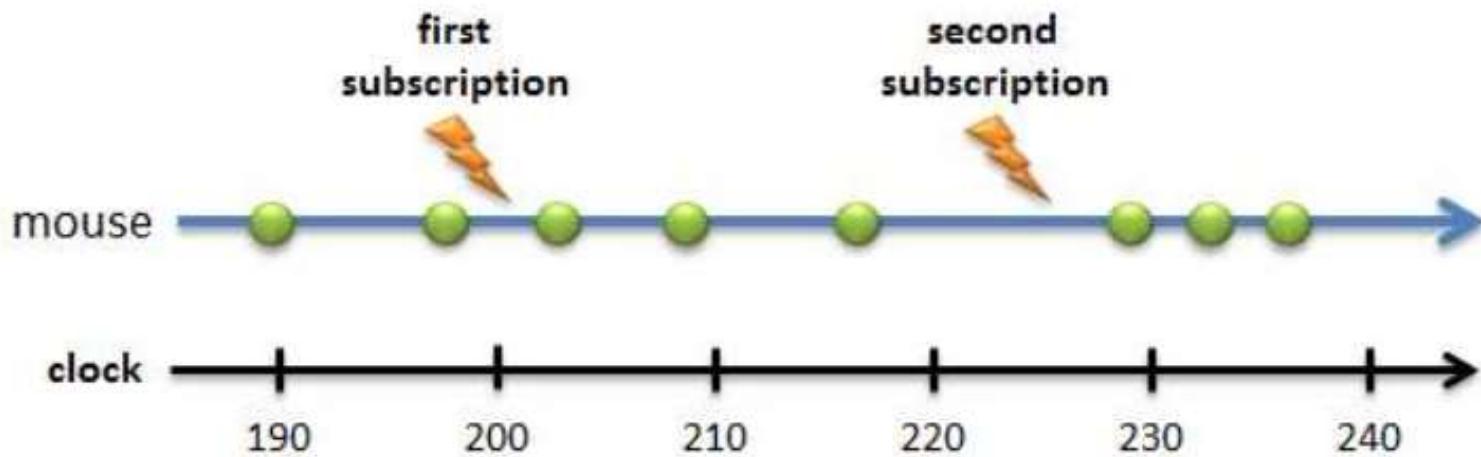
# Zastosowania

- Reagowanie na akcje użytkownika:  
kliknięcia, ruch myszki, klawiatura itp.
- Otrzymywanie i reagowanie na dane, również w czasie rzeczywistym, np. po web socketach
- Zdarzenia wykonywane po czasie np. setTimeout,  
setInterval
- Wszystko co asynchronicznie zwraca od 0 do  
nieskończonej ilości wartości

# Cold Observables



# Hot Observables



# Usługi asynchronous

- Usługi asynchronous zwracające jako wynik obiekty typu Observable lub Promise.
- Należy więc przekształci zwracana wartość w Observable lub promise

```
import { Observable } from 'rxjs/Observable';
```

```
import { of } from 'rxjs/observable/of';
```

```
getProducts(): Observable<Product[]> {
```

```
    return of(this.products);
```

```
}
```

of(this.products) emituje pojedynczą wartość pochodząca z tablicy produktów.

# Usługi asynchroniczne

- Zmiany w metodach usługi ProductService skutkują błędem w ProductsComponent. -> powód  
ProductService.getProducts() zwraca teraz Observable<Product[]>

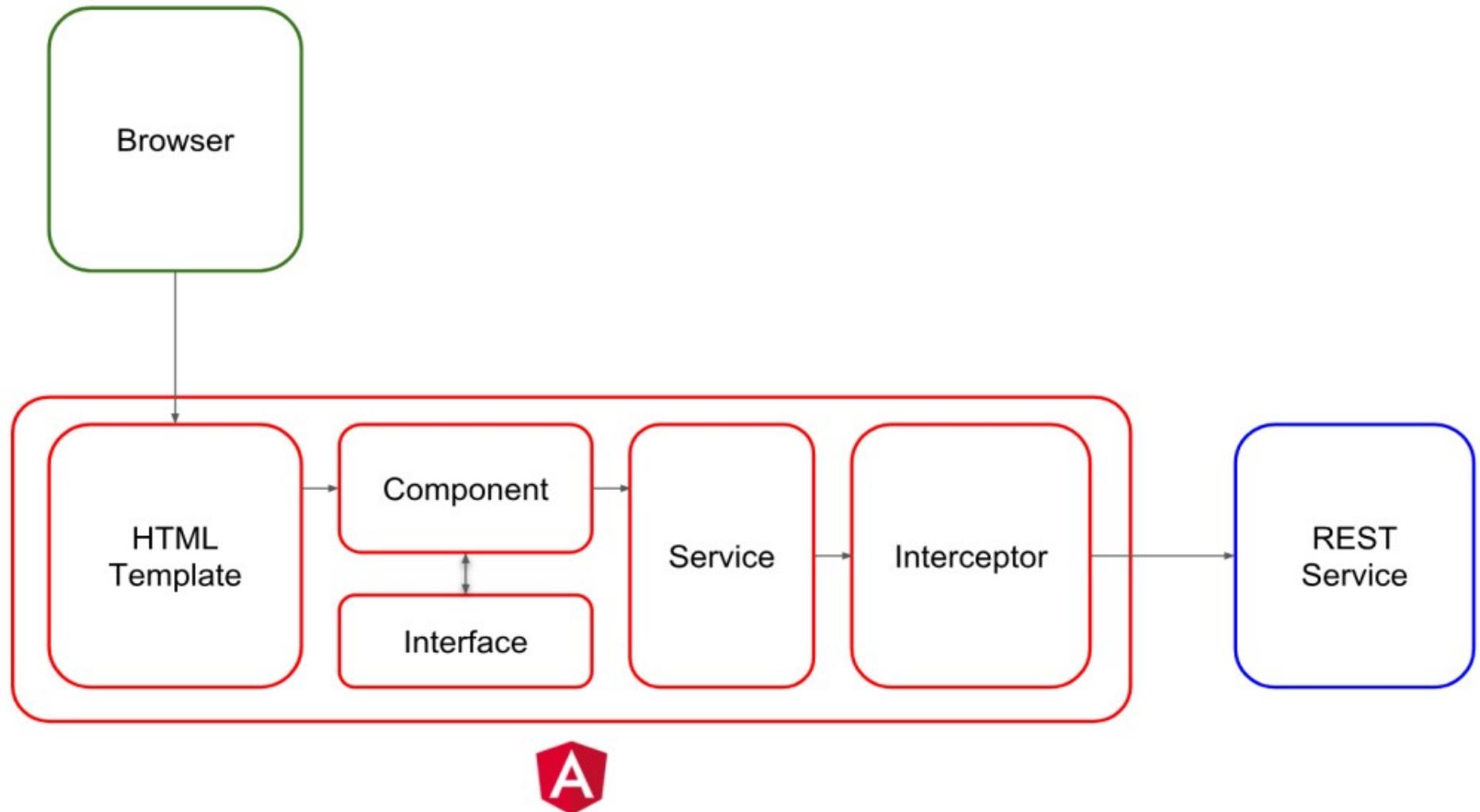
```
ngOnInit() {  
    this.products = this.productService.getProducts();  
}
```

Stara wersja

```
ngOnInit() {  
    this.productService.getProducts().subscribe(  
        products => this.products = products  
    );  
}
```

nowa wersja

# Architektura rest API w Angularze



# Usługi asynchroniczne i wzorzec obserwator

- Najważniejsza różnica to wykorzystanie Observable.subscribe().
- Wcześniej przypisywaliśmy całą tablicę komentarzy synchronicznie.
- Nie zadziała to przy rzeczywistej usłudze, która zwraca dane z opóźnieniem.
- Nowa wersja czeka, dopóki obserwowana usługa nie wyemituje tablicy z komentarzami.
- Obserwujący usługę komponent czeka na dane z komentarzami.
- Metoda subscribe() przekazuje otrzymaną tablicę do wywołania zwrotnego (tutaj wyrażenie lambda, funkcja strzałkowa), które przypisuje zawartość lity komentarzy do lokalnego pola komponentu.

# Dane zwracane z HttpClient

- Wszystkie metody HttpClient zwracają jakiś obserwowany obiekt (Observable).
- Protokół HTTP dla pojedynczego zapytania zwraca pojedynczą odpowiedź.
- Obiekt obserwowany może zwrócić wiele wartości w czasie.
- Obiekt obserwowany z HttpClient zawsze zwraca pojedynczą wartość i kończy działanie.
- Metoda HttpClient.get u nas zwraca Observable <Comments[]> (obserwowanie obiektu z tablicą komentarzy).
- Zwraca po prostu pojedynczą tablicę komentarzy.

# HttpClient – współpraca z serwerem danych

## Konfiguracja

### 1. Import modułu httpClient

```
import { HttpClient Module} from  
'@angular/common/http';
```

```
@NgModule({  
  declarations: [  
    AppComponent  
  ],  
  imports: [  
    BrowserModule,  
    HttpClientModule  
  ],  
  providers: [],  
  bootstrap: [AppComponent]  
})  
export class AppModule {  
}
```

AppModule

### 2. Import HttpClient w serwisie

```
import { HttpClient, HttpHeaders } from '@angular/common/http';  
  
export class ProductService{  
  ...  
  constructor(private httpClient: HttpClient) {  
    ...  
  }  
  ...  
}
```

ProductService.service.ts

# HttpClient – współpraca z serwerem danych odbiór danych – get()

## 3. Odczyt danych

```
export class ProductService{  
  productsUrl = 'api/products';  
  
  constructor(private httpClient: HttpClient) {  
  }  
  
  getProducts(): Observable<Product[]> {  
    return this.httpClient.get<Product[]>(this.productsUrl);  
  }  
  ...  
  ...  
}
```

ProductService.service.ts

HttpClient.get zwraca response jako untyped JSON object.  
Zastosowanie specyfikatora typu np. <Product[]>, daje obiekt zrzutowany.

# HttpClient – współpraca z serwerem danych

```
constructor (private http: HttpClient) {}

getProducts(): Observable<Product[]> {
    return this.http.get<Product[]>(PATH);    }

getProduct(id: number): Observable<any> {
    return this.http.get(PATH+id);    }

addProduct(produkt: Produkt) {
    this.http.post<Produkt>(PATH, produkt).subscribe(
        res => { console.log(res) },
        (err: HttpErrorResponse) => { console.log(err) }      );
}

updateProduct(produkt: Produkt) {
    this.http.put(PATH + produkt.id, produkt).subscribe();    }

deleteAllProduct(produkt: Produkt) {
    this.http.delete(PATH + produkt.id).subscribe();    }
```

# HttpClient – współpraca z serwerem danych

```
getCustomers(): Observable<Customer[]> {  
    return this.http.get('/api/customers');  
}
```

1 wersja

```
getCustomer(id): Observable<Customer> {  
    return this.http.get(`/api/customers/${id}`);  
}
```

```
getCustomers(): Observable<Customer[]> {  
    return this.http.get <Customer[]> ('/api/customers');  
}
```

2 wersja

```
getCustomer(id): Observable<Customer> {  
    return this.http.get <Customer> (`/api/customers/${id}`);  
}
```

```
postCustomer(customer): Promise {  
    return this.http.post('/api/customers', customer)  
        .toPromise()  
        .then((data) => data);  
}
```

3 wersja

# Czym jest NodeJS?



- Data utworzenia: **2009 r.**
- Niskopoziomowa implementacja silnika **V8 JavaScript'u** po stronie serwera.
- Napisany w języku **C/C++** (8000 linii) oraz **JavaScript** (2000 linii), obsługuje programy napisane w **JavaScript**.

*« A platform built on Chrome's  
JavaScript runtime for easily building  
fast, scalable network applications. »*  
<http://nodejs.org/>

Platformę utworzoną na podstawie środowiska uruchomieniowego JavaScript przeglądarki internetowej Chrome, przeznaczoną do łatwego tworzenia szybkich, skalowalnych aplikacji.



<http://nodejs.org/>

Node.js, w skrócie Node, to środowisko uruchomieniowe języka JavaScript open source po stronie serwera.

Za pomocą środowiska Node.js można uruchamiać aplikacje i kod JavaScript w wielu miejscach poza przeglądarką, na przykład na serwerze.

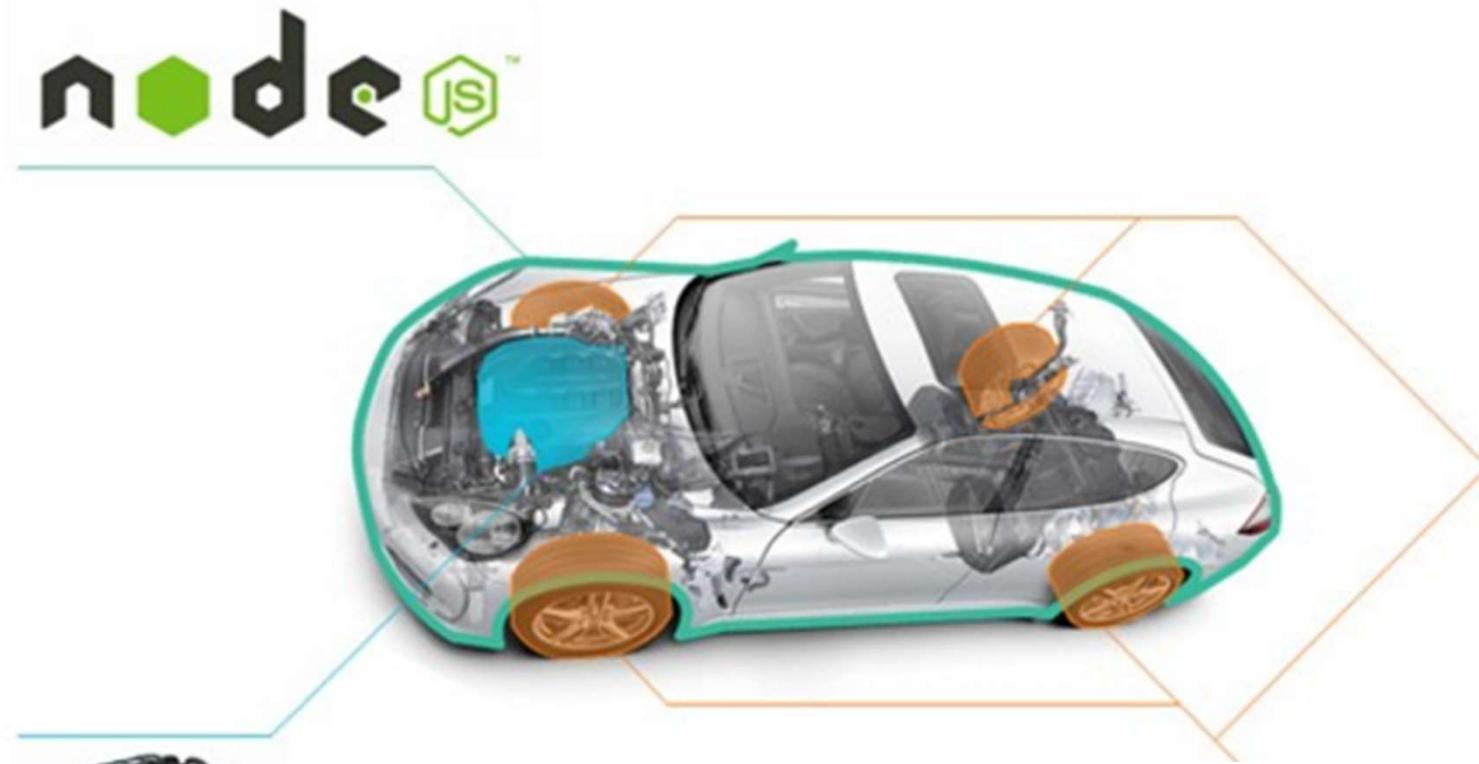
**Node.js - nowoczesne środowisko uruchomieniowe języka JavaScript, działające na serwerze (poza przeglądarką).**

**Jest dystrybuowane na zasadzie otwartego oprogramowania (OpenSource).**

**Jego podstawową funkcją (nie jedyną) jest możliwość tworzenia serwerów aplikacji internetowych opartych protokoły HTTP.**

**Sterowany zdarzeniami wykorzystując system wejścia/wyjścia (I/O) który jest asynchroniczny.**

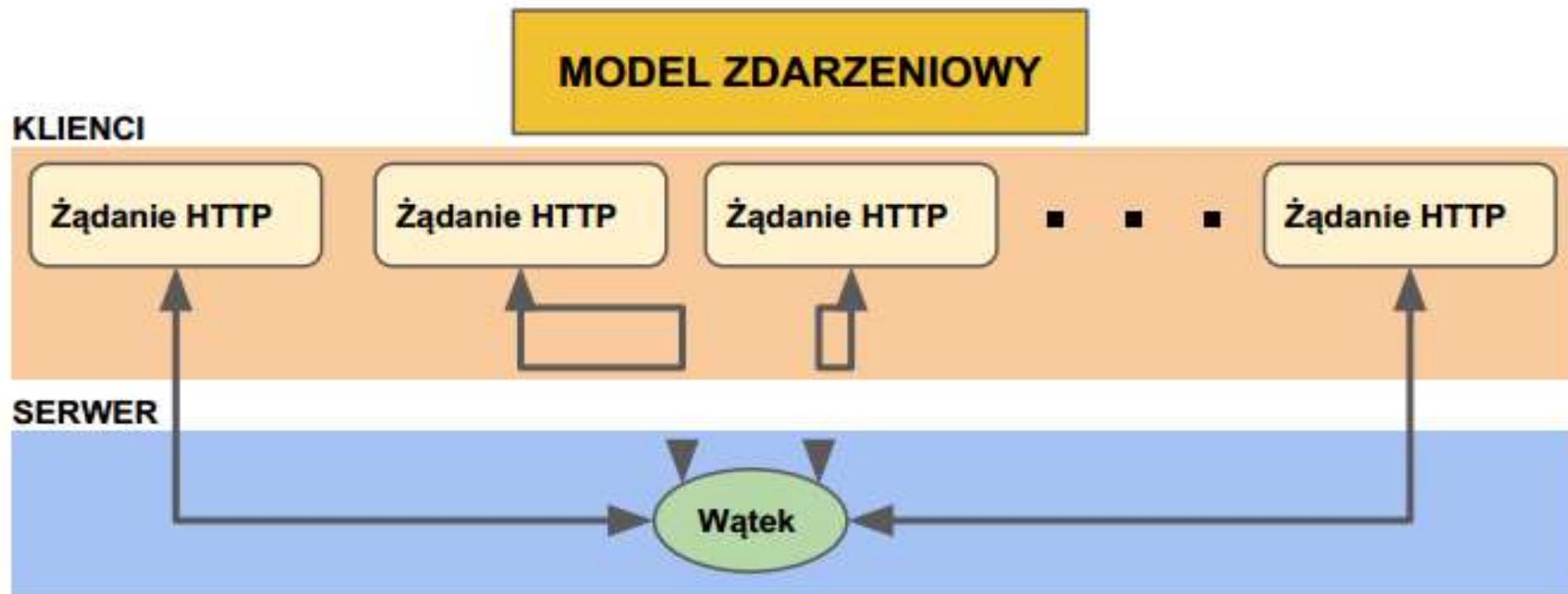
## Czym jest Node.js



**Node.js** używa opartego na zdarzeniach, nieblokującego modelu wejścia-wyjścia, co zapewnia lekkość i efektywność.

Stanowi doskonałe rozwiązanie dla działających w czasie rzeczywistym aplikacji intensywnie korzystających z danych oraz aplikacji rozproszonych w różnych urządzeniach

# NodeJS

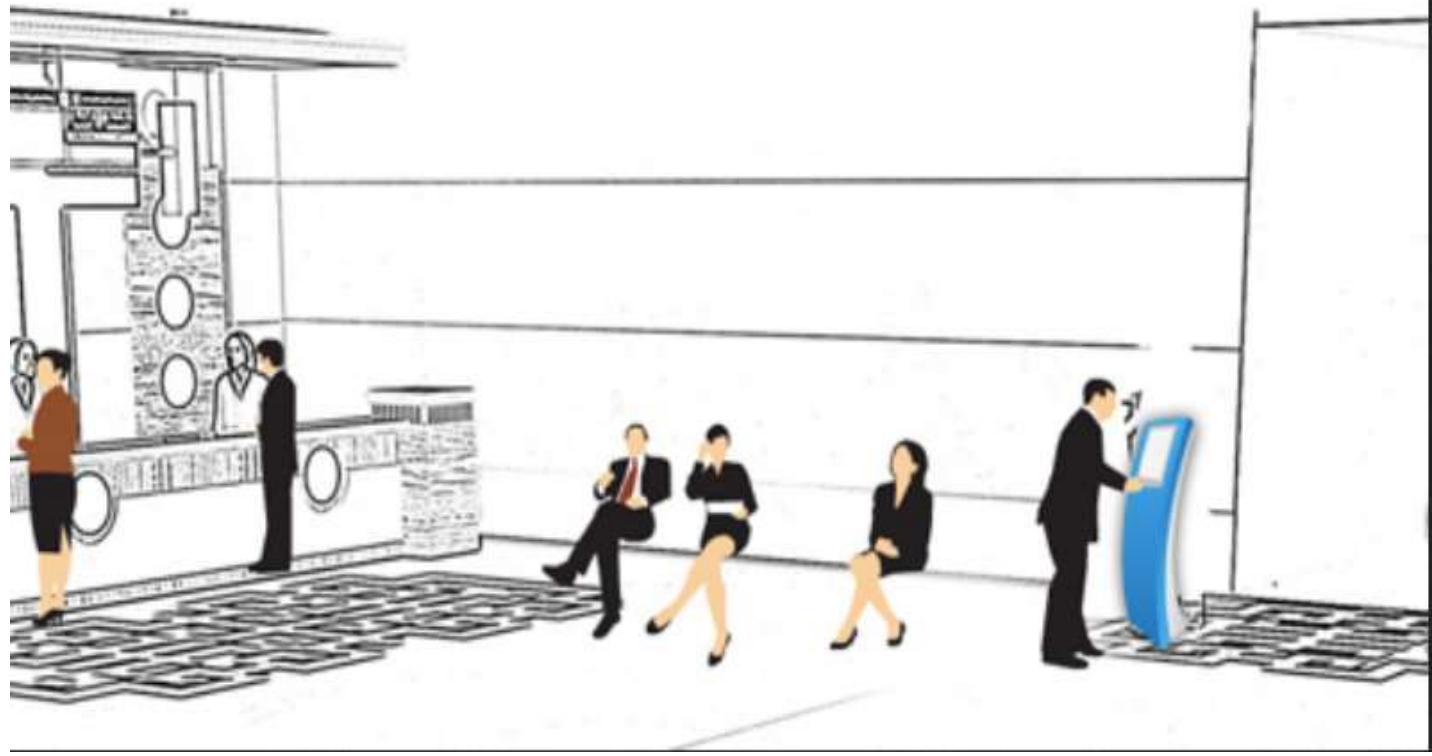


**W modelu zdarzeniowym Node.js wykorzystuje tylko jeden wątek do obsługi wielu zadań, oraz “pętlę zdarzeń” co powoduje że aplikacja taka jest bardzo wydajna i skalowalna. W praktyce przy żądaniach które nie wymagają złożonych operacji obliczeniowych można obsłużyć nawet do 1 miliona żądań jednocześnie.**



I/O blokujące

I/O nieblokujące



# NodeJS



# Kiedy stosować NodeJS?

- zalecany do tworzenia aplikacji:
  - z dużą liczbą operacji wejścia/wyjścia,
  - strumieniowania danych, np. video,
  - Single Page Applications (SPA),
  - udostępniających API w formacie JSON,
  - z intensywną wymianą danych w czasie rzeczywistym na wielu urządzeniach, np. portale społecznościowe,
- nie zalecany przy aplikacjach intensywnie wykorzystujących procesor (CPU),
- npm - system pakietów Node.js - największy zbiór bibliotek open source na świecie (260000) → łatwa produkcja aplikacji internetowych,
- Node.js = środowisko uruchomieniowe + biblioteki JavaScript

Aplikacje typu DIRT (ang  
Data Intensive Real Time)

# Do czego stosować NodeJS

- Serwery internetowe HTTP
- Mikrousługi lub bezserwerowe zaplecza interfejsu API
- Sterowniki umożliwiające dostęp do bazy danych i wykonywanie zapytań
- Interakcyjne interfejsy wiersza polecenia
- Aplikacje klasyczne
- Biblioteki serwera i klienta IoT w czasie rzeczywistym
- Wtyczki dla aplikacji klasycznych
- Skrypty powłoki do manipulowania plikami lub uzyskiwania dostępu do sieci
- Biblioteki i modele uczenia maszynowego

# Pierwszy projekt w NodeJS

Większość aplikacji Node.js składa się z 3 części:

- import wymaganych modułów - używa się dyrektywy **require**,
- utworzenie serwera - serwer będzie oczekiwany na żadania klientów i zwracał odpowiedzi,
- odczytywanie żądań i zwracanie odpowiedzi - podstawowe działanie serwera.

```
var http = require("http");

var server = http.createServer(function (request, response) {
    // Wysyłanie nagłówków protokołu HTTP
    // Status HTTP: 200 : OK, Content Type: text/plain
    response.writeHead(200, {'Content-Type': 'text/plain'});

    // Wysyłanie ciała odpowiedzi, niezależnie od rodzaju żądania
    response.end('Pierwszy projekt w Node.js\n');
});

server.listen(5000);
console.log('Server działa na http://127.0.0.1:5000/');
```

# Tworzenie serwera w NodeJS

```
var http = require('http');
var fs = require('fs');
var url = require('url');

http.createServer( function (request, response) {
    // Parsuje żądanie zawierające nazwę pliku
    var pathname = url.parse(request.url).pathname;
    // Wyświetlanie nazwy pliku, którego dotyczyło żądanie
    console.log("Request for " + pathname + " received.");

    fs.readFile(pathname.substr(1), function (err, data) {
        if (err) {
            console.log(err);
            response.writeHead(404, {'Content-Type': 'text/html'});
        } else {
            response.writeHead(200, {'Content-Type': 'text/html'});
            response.write(data.toString()); // zwracanie treści wybranego pliku
        }
        response.end(); // wysyłanie odpowiedzi
    });
}).listen(5000);

console.log('Serwer uruchomiony na http://127.0.0.1:5000/');
```

# Tworzenie serwera z użyciem NodeJS

index.html

```
<html>
<head>
<title>Pierwszy projekt</title>
</head>
<body>
Pierwszy projekt w Node.js
</body>
</html>
```

```
$ node server.js
```

Serwer uruchomiony na <http://127.0.0.1:5000/>

Otwieramy w przeglądarce: <http://localhost:5000/index.html>

## REST Client



Browser /  
HTTP Tools/plug-ins



Any windows /  
.NET app



Any Java app



PHP, JSP, Servlet, CF,  
ASP.NET, ASP.NET MVC,  
Any Javascript SPA,  
Angular JS,  
etc.



iOS,  
Android,  
Windows Phone,  
PhoneGap/Cordova,  
Sencha,  
etc.

GET, POST, PUT, DELETE etc.



## REST Service



- \* core
  - db.js
  - httpMsgs.js
  - server.js
- \* controllers
  - employee.js
  - app.js



# NodeJS

**Wróćmy do struktury podstawowego serwera WWW:**

```
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}/`);
});
```

Czy da się przedstawić tą strukturę w bardziej uporządkowany i usystematyzowany sposób ?

TAK...

# Express.js



**Express.js jest “pakietem” które pozwala na tworzenie aplikacji internetowych o architekturze MVC (serwisowej).**

**Ułatwia obsługę żądań HTTP i wprowadza szerokie możliwości tworzenia aplikacji internetowych / sieciowych.**

```
> npm install express
```

# Express - instalacja

```
$ npm install express --save
```

Inne ważne moduły, które warto od razu zainstalować:

- **body-parser** - warstwa pośrednia obsługująca JSON, Raw, Text i dane formularza przekazane w URL,
- **cookie-parser** - przetwarza nagłówki ciasteczek (cookie header) i dodaje obiekt do `req.cookies`, w którym klucze to nazwy przesłanych ciasteczek,
- **multer** - warstwa pośrednia w Node.js do obsługi multipart/form-data (kodowanie danych z formularza).

# Express.js

```
//- plik app.js

const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Hello World!')
});

app.listen(3000, () => {
  console.log('Serwer dziala na porcie 3000!')
});
```



Prosta aplikacja napisana z użyciem Express.js nastuchująca żądań na porcie 3000 i odsyłająca odpowiedź w postaci tekstu.

Przedstawiony kod w zasadzie niczym nie różni się od “prostego” serwera HTTP....  
ma jednak kilka zalet ....

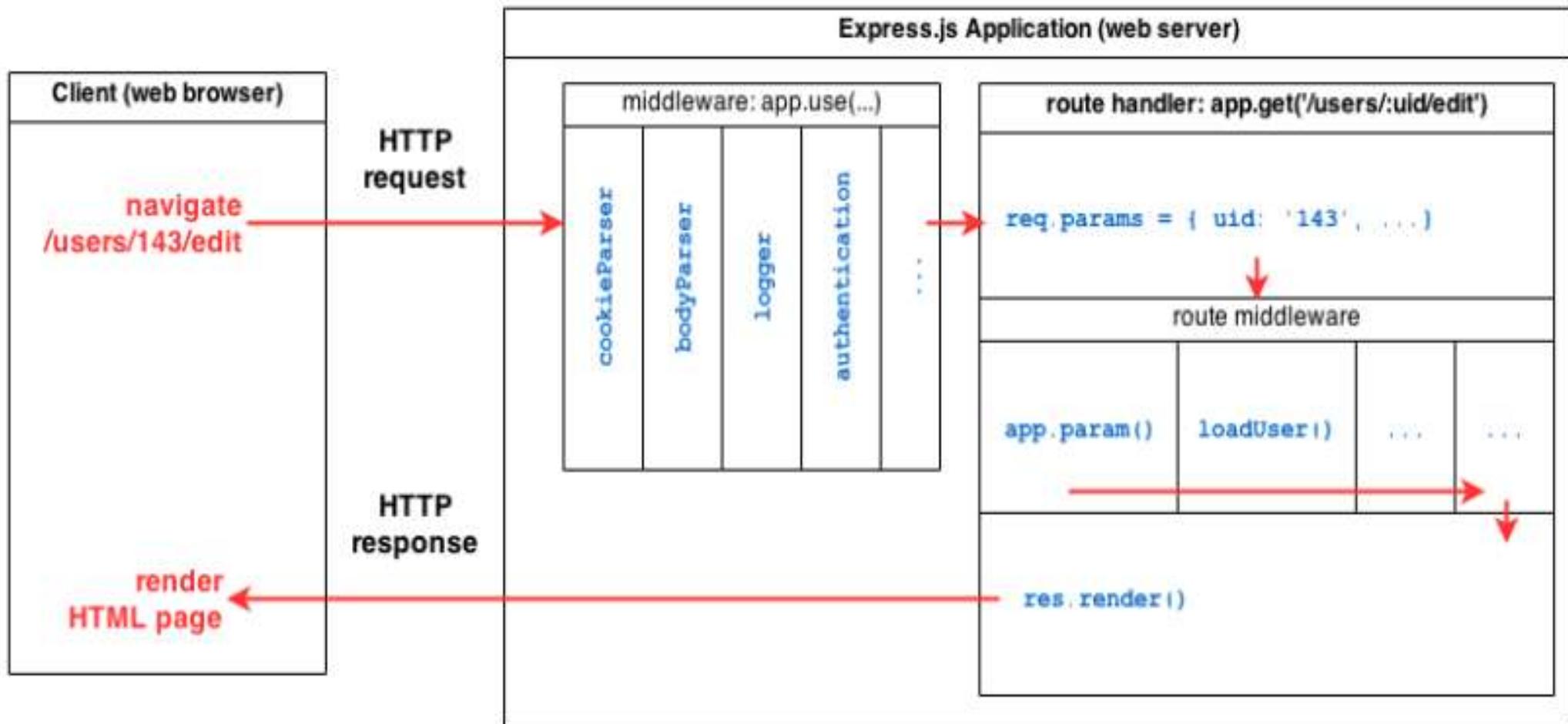
# Express.js - inna wersja

```
var express = require('express');

var app = express();
app.get('/', function(req, res) {
    res.send('Pozdrowienia od GR!');
});
app.listen(3000, function() {
    console.log('Aplikacja nasluchuje na porcie 3000!');
});
```

# ExpressJS i warstwy pośrednie

Warstwy pośredniczące dodajemy do ExpressJS używając `app.use` dla dowolnej metody albo `app.VERB` (np. `app.get`, `app.delete`, `app.post`, `app.update`, ...)



# Express - statyczne pliki

Express posiada wbudowaną warstwę pośrednią `express.static`, która pozwala na udostępnianie statycznych plików (obrazy, CSS, HTML, JavaScript, ...)

Wystarczy przekazać nazwę katalogu, z którego pliki będą dostępne poprzez serwer:  
`app.use(express.static('public'));`

```
var express = require('express');
var app = express();

app.use(express.static('public'));

app.get('/', function (req, res) {
  res.send('Hello World');
})

var server = app.listen(5000, function () {
  var host = server.address().address
  var port = server.address().port
  console.log("Przykładowa aplikacja nasłuchuje na http://%s:%s", host, port)
})
```

Wywołanie: `http://localhost:5000/npmLogo.png`  
wyświetli odpowiedni plik znajdujący się w katalogu `public`.

# Express.js

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});

module.exports = router;
```

Plik: routes/index.js

req - obiekt żądania przychodzącego od klienta

res - obiekt odpowiedzi (do klienta)

Pojedyncza trasa obsługująca określone żądanie przychodzące od klienta.

Odpowiedź na żądanie klienta jest formułowana za pomocą mechanizmu funkcji wywołania zwrotnego "callback".

Najważniejszą rolę w kodzie powyżej spełnia obiekt "router", który jest odpowiedzialny za "trasowanie" czyli obsługę wszystkich wywołań adresów URL.

Każda trasa może posiadać jedną lub więcej funkcji obsługi trasy. W momencie rejestracji żądania router określa która funkcja obsługi trasy zostanie wykonana.

Ogólna postać funkcji obsługi trasy wygląda następująco:

```
router.METODA( TRASA, OBSŁUGA TRASY )
```

Metoda HTTP np.  
GET, POST, PUT,  
DELETE

Trasa: adres URL  
żądania klienta

Najczęściej wywołanie zwrotne  
realizujące określoną funkcjonalność.

```
app.get('/', (req, res) => res.send('GET'));
app.post('/', (req, res) => res.send('POST'));
app.put('/', (req, res) => res.send('PUT'));
app.delete('/', (req, res) => res.send('DELETE'));
```

# Express.js – prosty routing

Przykłady funkcji obsługi żądań:

```
router.get('/', function (req, res) {
  res.send('Hello World!');
});

router.post('/', function (req, res) {
  res.send('Żądanie POST');
});

router.put('/user', function (req, res) {
  res.send('Żądanie PUT dla URL /user');
});

router.delete('/user', function (req, res) {
  res.send('Żądanie DELETE dla URL /user');
});
```

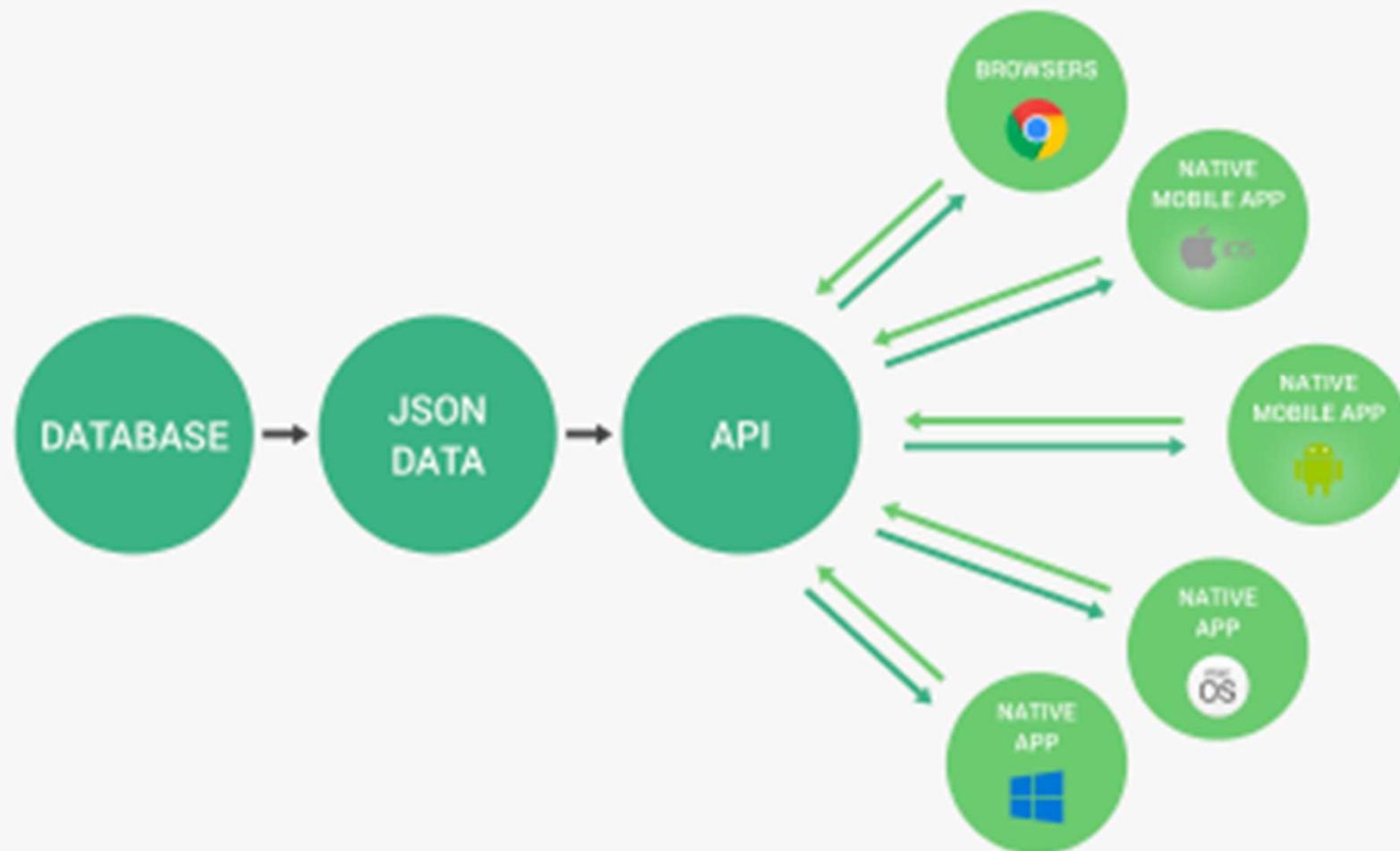
Funkcja obsługująca tą samą trasę “ / ”, ale dla dwóch różnych możliwych metod HTTP.

Bardzo często “trasa” w postaci adresu URL oraz metoda (np. GET) w literaturze określana jest jako “END-POINT”.

Podobnie jak poprzednio dwie usługi “END-POINT”.

# Jedno API wielu konsumentów

## 👉 Web APIs

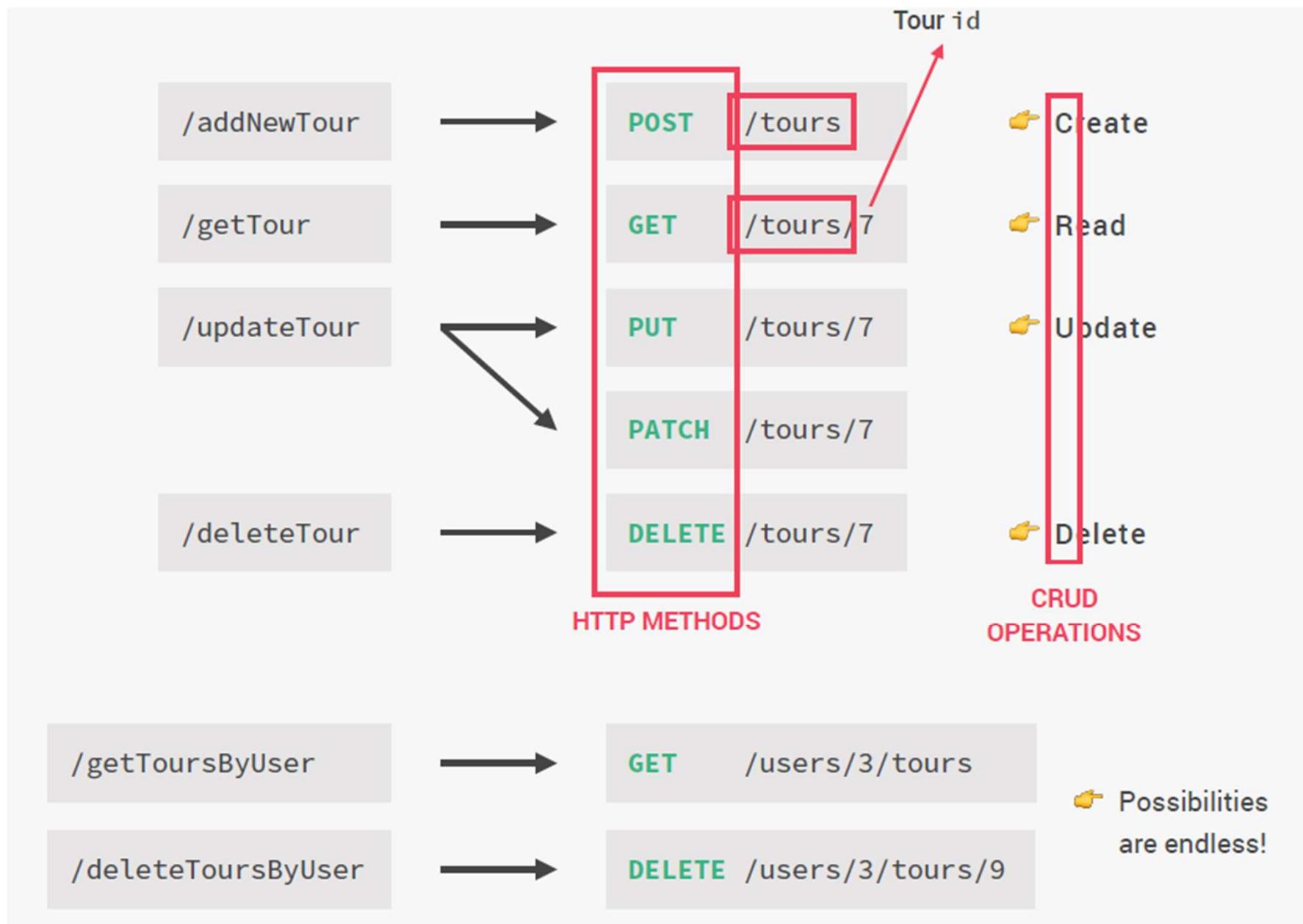


# REST API

Klasyczne API RESTful, zawiera cztery podstawowe operacje CRUD:

Resource (URI)	POST (create)	GET (read)	PUT (update)	DELETE (destroy)
/zadania	nowe zad.	lista zad.	błąd	błąd
/zadania/:id	błąd	zad. o :id	aktualizacja :id	usuń 1 zad. o ID

# Use HTTP methods (verbs) in REST API server



# Przykładowe API dla kolekcji produkty

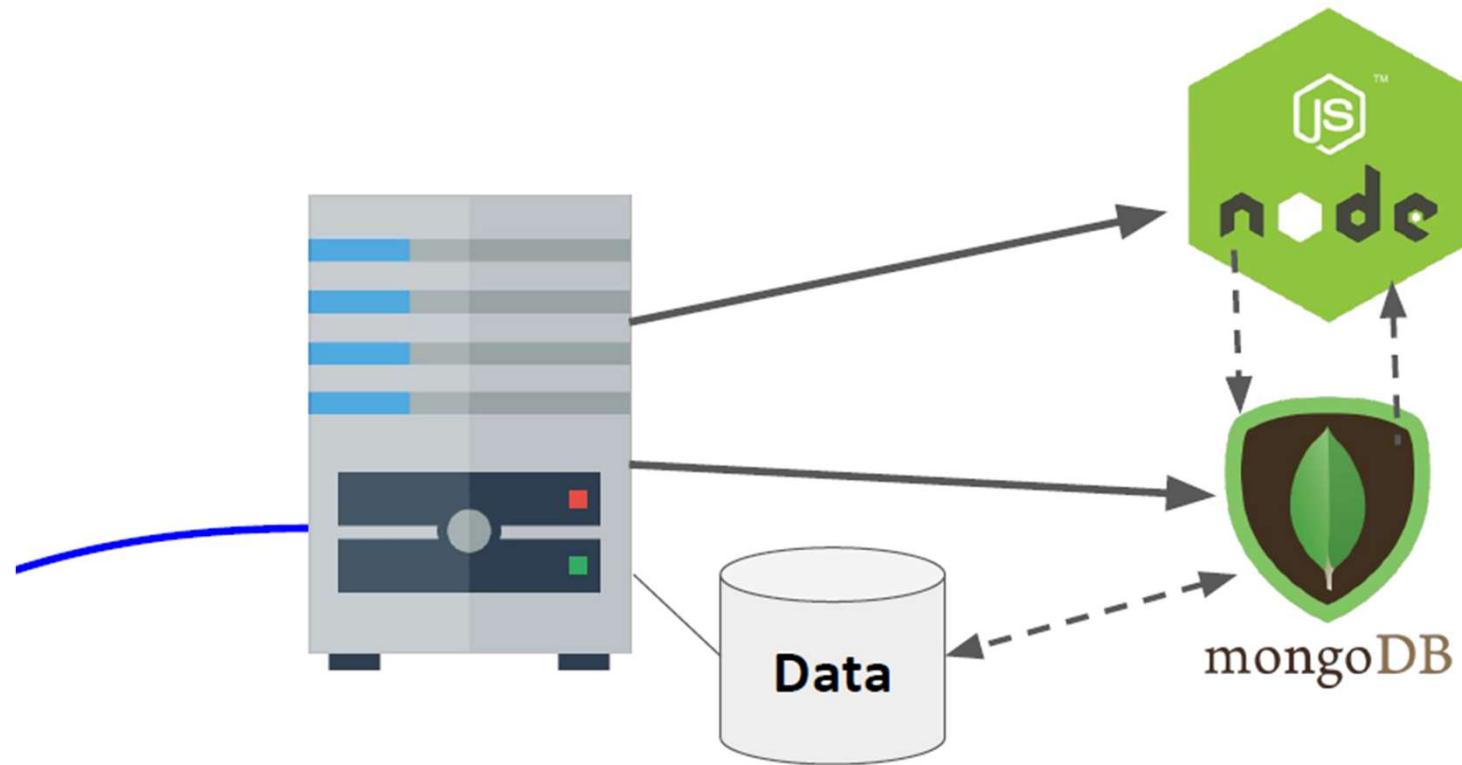
Zadanie	Metoda	Ścieżka	Przyjmuje	Zwraca
Pobieranie danych	GET	/produkty	nic	tablica obiektów
Tworzenie obiektu	POST	/produkty	pojedynczy obiekt	zapisany obiekt (albo błąd)
Pobieranie obiektu	GET	/produkty/<id>	nic	pojedynczy obiekt
Aktualizacja obiektu	PUT	/produkty/<id>	pojedynczy obiekt	zapisany obiekt (albo błąd)
Usuwanie obiektu	DELETE	/produkty/<id>	pojedynczy obiekt	nic

# Przykładowy routing w Express Web Server

```
var customers = require('../controllers/customer.controller.js');

// Create a new Customer
app.post('/api/customer', customers.create);
// Retrieve all Customer
app.get('/api/customers', customers.findAll);
// Retrieve a single Customer by Id
app.get('/api/customer/:customerId', customers.findOne);
// Update a Customer with Id
app.put('/api/customer/:customerId', customers.update);
// Retrieve Customers Age
app.get('/api/customers/age/:age', customers.findByAge);
// Delete a Customer with Id
app.delete('/api/customer/:customerId', customers.delete);
}
```

# Czas na przechowywanie danych



# Integracja z bazami danych

Jednym z istotnych aspektów tworzenia nowoczesnych aplikacji internetowych jest przechowywania i wymiana danych. Najczęściej dane które są używane w aplikacjach są przechowywane w bazach danych.

Node.js oraz Express.js posiadają interfejsy do najpopularniejszych typów baz danych:

- MySQL
- PostgreSQL
- Oracle
- MongoDB (noSQL)



# Integracja z bazami danych



Wszystko zależy od typu problemu jaki chcemy rozwiązać w oparciu o bazy danych.

- **brak związków encji**
- **nie możliwe do wykorzystania w niektórych złożonych problemach.**
- **skalowalne,**
- **rozproszone,**
- **niski stopień złożoności**
- **elastyczność**

# Integracja z bazami danych

Jedną z najszerzej wykorzystywanych silnikiem baz danych NoSQL w aplikacjach Node.js jest Mongo DB. Jest to aplikacja bazodanowa która bardzo dobrze sprawdza się w zastosowaniach “cloud” (w chmurze).



[www.mongodb.org](http://www.mongodb.org)

MongoDB jest rozwiązaniem które można wykorzystywać na zasadzie otwartej licencji do wszystkich zastosowań.

W MongoDB dane przechowywane są w postaci dokumentów o bardzo elastycznej strukturze przypominającej format JSON:

```
...  
{  
    "name": "Jan",  
    "lname": "Kowalski",  
    "email": "jan@kowalski.pl",  
    "age": 21,  
    "groups": ["users", "mail", "root"]  
}  
...
```

Pojedynczy dokument  
przechowujący dane