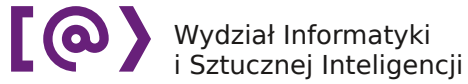


POLITECHNIKA CZĘSTOCHOWSKA
WYDZIAŁ INFORMATYKI I SZTUCZNEJ INTELIGENCJI



PRACA DYPLOMOWA INŻYNIERSKA

Aplikacja klient-serwer

client-server application

Dawid Ziora

Nr albumu: 136700

Kierunek: Informatyka

Studia: stacjonarne

Poziom studiów: I

Promotor pracy:

dr inż. Bartosz Kowalczyk

Praca przyjęta dnia:

Podpis promotora:

Częstochowa, 2024

Spis treści

Wstęp	3
1 Technologie, z których korzysta aplikacja	5
1.1 Spring Framework [1]	5
1.1.1 Spring boot	5
1.1.2 Spring MVC	6
1.2 angular	6
1.3 MongoDB	8
2 Wymagania i diagramy	9
2.1 wymagania funkcjonalne	9
2.2 wymagania нефункционалне	9
2.3 Diagramy przypadków użycia	9
2.4 Diagramy sekwencji	9
3 Kod, funkcje, itd. CORE	10
4 podsumowanie oraz wnioski	11
Zakończenie	11
Co osiągnąłem	11
Streszczenie	11
Bibliografia	12
Spis rysunków	13
Spis tabel	14
Spis listingów	14

Wstęp

Aplikacje webowe są bardzo powszechnie używanymi aplikacjami naszych czasów. Coraz częściej chcemy aby na naszych urządzeniach nie pojawiała się wiele różnego rodzaju aplikacji, tylko by wszystko mogło zostać upakowane w przeglądarkę internetową z której prosto i szybko uzyskamy dostęp do programów i aplikacji których potrzebujemy. W tej pracy postaram skupić się na aplikacji typu fintech, czyli aplikacji związanej z usługami finansowymi.

cel pracy

Celem mojej pracy będzie uzyskanie działającej aplikacji bankowej w której klient będzie porozumiewał się z serwerem przy pomocy różnego rodzaju zapytań.

zakres pracy

Językiem z którego skorzystam podczas pisania pracy będzie Java oraz Spring (Do logiki serwerowej - Backendu), baza danych MongoDB do obsługi bazy danych oraz Angular i npm do obsługi frontendu.

opisówka

1. Technologie, z których korzysta aplikacja

1.1. Spring Framework [1]

Jest to struktura, która dostarcza funkcjonalność dla aplikacji. Odpowiada ona między innymi za:

- wstrzykiwanie zależności (Dependency Injection)
- Obsługa zdarzeń (Events)
- Zarządzanie zasobami (Resources)
- Walidacje (Validation)
- Wiązanie danych (Data binding)
- Konwersje typów (Type conversion)
- SpEL
- AOP

Spring jest bardzo popularnym frameworkiem do budowania aplikacji webowych. Jego popularność zawdzięcza takim rzeczom jak

- łatwe wstawianie zależności (Dependency Injection)
- dobrze zintegrowany z innymi frameworkami javy takimi jak JPA/Hibernate
- Posiada framework MVC do budowania aplikacji webowych

1.1.1. Spring boot

- **Starter** pozwala skorzystać z wcześniej skonfigurowanych zależności
- przykłady starterów
 - spring-boot-starter-data-jpa
 - spring-boot-starter-web

- **Automatyczna konfiguracja** - spring boot jest wyposażony w narzędzie, które na bazie zależności **jar**, które dodaliśmy do projektu próbuje skonfigurować projekt

1.1.2. Spring MVC

Jest on zaawansowanym frameworkiem webowym. Składa się on z takich elementów jak:

- Dyspozytor Serwletu (DispatcherServlet)
- Mapowanie żądań (Request Mapping)
- Metody obsługi żądań (Handler methods)
- Obsługa wyjątków (Exceptions handling)

1.2. angular

Ważne komendy:

- Tworzenie komponentów
ng g c [nazwa]
ng generate component [nazwa]
ng generate component sciezka/[nazwa]
- sprawdzenie co wykona komenda
-dry-run
- DataBinding - tworzenie zmiennych w tekst tak jakby
zmienna - tradycyjny sposob
funkcja () - signal
- Routing - bardzo ważna rzecz
służy do tworzenia strony na jednej stronie

Wszystko w app.routes

W templatce podajemy <routing> i wtedy w zależności od adresu przenosi nas na odpowiednią stronę

item Tworzenie klas

ng g class [nazwa]

item Tworzenie serwisów
ng g s [nazwa]

- łączenie się z serverem - Tworzymy serwis o jakiejś nazwie następnie w środku serwisu umieszczamy adres http z którego skorzystamy oraz prywatny adres klienta

Listing 1.1. Łączenie z serverem

```
1  @Injectable({  
2    providedIn: 'root'  
3  })  
4  export class ServiceName {  
5    //bazowy adres serwera  
6    private baseUrl = 'http://localhost:8080/list';  
7  
8    constructor(private httpClient: HttpClient) { }  
9  
10   getUsersList(): Observable<User[]>{  
11     return  
12       this.httpClient.get<User[]>(`${this.baseUrl}`);  
13   }
```

- Wstrzykiwanie zależności jest możliwe poprzez polecenie @Injectable oraz narzędzia dostarczania usług (providers). Tworzymy usługę w niej właśnie znajduje się możliwość wstrzyknięcia tejże usługi. Następnie w jakimś komponencie może zostać wstrzyknięta usługa.

Listing 1.2. Wstrzyknięcie serwisu

```
1  export class NazwaKomponentu implements OnInit {  
2    constructor(private userService: UserService) {  
3      //Wstrzykiwanie usługi  
4    }  
5    //Dalsza przykładowa funkcjonalność (użycie usługi)  
6    ngOnInit() {  
7      this.getUsers();  
8    }  
9    getUsers() {
```

```
10   this.userService.getUsersList().subscribe(data => {  
11     this.users = data;  
12   });  
13 }
```

- obietnice (promise) - to taki asynchroniczny element, który pozwala odroczyć wczytanie pewnej zależności w czasie i wczytać ją dopiero wtedy gdy będzie potrzebna.

Listing 1.3. Promise

```
1 {path: 'home',  
2   loadComponent: async () => {  
3     const m = await import('./home/home.component');  
4     return m.HomeComponent;}},
```

Wczytuje komponent home tylko jeśli zostanie wywołana ścieżka home. (Przynajmniej na ten moment tak to rozumiem).

1.3. MongoDB

2. Wymagania i diagramy

2.1. wymagania funkcjonalne

2.2. wymagania niefunkcjonalne

2.3. Diagramy przypadków użycia

2.4. Diagramy sekwencji

3. Kod, funkcje, itd. CORE

4. podsumowanie oraz wnioski

Co osiągnąłem

Bibliografia

- [1] H. Kopka and P. W. Daly. *A Guide to LaTeX*. Addison-Wesley, Reading, MA, 1999.

Spis rysunków

Spis tabel