

# Opis

Opisuje se od gore na dole.

## Frontend ( veza sa backend-om )

Single-spa se sastoji od 7 mikrofrontend-a. Svaki mikroforntend ima svoje servis klase i interseptor klasu sa kojima komunicira sa backend-om. Interseptor klasa stavlja u svaki HTTP zahtev, u njegov header, token i fingerprint ( u browser-u se čuvaju kao kolačići ). Oni su važni za RBAC sigurnosni sistem koji je u backend-u i bez njih se ne može obaljati većina operacija.

## Java

Java je objektno-orijentisani programski jezik koji je razvila kompanija Sun Microsystems (sada deo Oracle Corporation). Karakteristike i koncepti Java jezika uključuju:

1. Java je potpuno objektno-orijentisani jezik, što znači da se sve u Javi smatra objektima koji imaju svoje atribute i metode.
2. Java je platformski nezavisan jezik, što znači da se Java programi mogu izvršavati na različitim operativnim sistemima, pod uslovom da je instalirana Java Virtual Machine (JVM).
3. Sintaksa Jave je slična C i C++ jezicima, što olakšava programerima prelazak na Javu ako su upoznati sa ovim jezicima.
4. Java koristi garbage collector koji automatski upravlja memorijom, olakšavajući programerima rad sa memorijom i smanjujući mogućnost curenja memorije.
5. Java ima bogatu kolekciju biblioteka i okvira koji olakšavaju razvoj aplikacija, kao što su Spring, Hibernate, JavaFX, itd.
6. Java ima ugrađene mehanizme za sigurnost, uključujući sigurnosne sandbox okoline i digitalno potpisivanje koda.

Java se često koristi za razvoj velikih i kompleksnih sistema, a Java Virtual Machine (JVM) omogućava prenosivost koda na različite platforme.

## Spring

Spring je popularan open-source okvir (framework) za razvoj softverskih aplikacija u programskom jeziku Java. Ovaj okvir pruža različite alate, biblioteke i komponente koje olakšavaju razvoj sigurnih, skalabilnih i visoko performantnih aplikacija. Ključne karakteristike Spring-a uključuju:

1. Inverzija kontrole (IoC - Inversion of Control): Spring primenjuje koncept inverzije kontrole gde je kontejner (Spring kontejner) odgovoran za upravljanje životnim ciklusom objekata i njihovim zavisnostima.
2. Upravljanje zavisnostima (DI - Dependency Injection): Spring koristi Dependency Injection mehanizam za ubacivanje zavisnosti između komponenata, čime se omogućava fleksibilnije i testabilno programiranje.
3. AOP (Aspect-Oriented Programming): Spring omogućava primenu aspektno-orijentisanog programiranja radi bolje separacije koncerna, kao što su transakcije, logovanje i sigurnost.
4. Modularnost: Spring je modularan i organizovan u niz modula koji se mogu koristiti po potrebi, što doprinosi fleksibilnosti i lakoći upotrebe.
5. Sigurnost: Spring obezbeđuje podršku za implementaciju sigurnosnih aspekata aplikacije, uključujući autentifikaciju, autorizaciju i enkripciju.
6. MVC (Model-View-Controller): Spring MVC je deo Spring-a koji omogućava razvoj veb aplikacija koristeći MVC arhitekturni obrazac.
7. Podrška za transakcije: Spring omogućava deklarativno upravljanje transakcijama, što pojednostavljuje upotrebu transakcija u aplikacijama.

Spring se koristi za razvoj raznovrsnih aplikacija, uključujući veb aplikacije, mikroservise, desktop aplikacije, serverske aplikacije i još mnogo toga.

## Role-based access control (RBAC)

Ovaj model se koristi za upravljanje pravima pristupa korisnicima na osnovu njihovih uloga u organizaciji. Svaka uloga ima *set* dozvoljenih akcija i aktivnosti koje korisnici koji imaju tu ulogu mogu da obavljaju u sistemu.

Osnovne komponente RBAC modela uključuju:

* Korisnici: Osobe, sistemi ili procesi koji zahtevaju pristup informacionom sistemu.
* Uloge: Skupovi prava koji određuju tipične zadatke ili funkcije u organizaciji. Uloge grupišu korisnike na osnovu sličnih potreba za pristupom.
* Dozvole ( autoritet ) : Prava koja se dodeljuju ulogama, određujući šta korisnici sa određenom ulogom mogu ili ne mogu da rade u sistemu.

U implementaciji RBAC-a, važno je pažljivo definisati uloge, njihove ovlašćenja i način dodeljivanja uloga korisnicima kako bi se postigla efikasna i sigurna kontrola pristupa u informacionim sistemima.

Kada se naziv uloge šalje na frontend, prvo se dodaje u token pa hešira, onda se taj token hešira sa fingerprint-om. Zatim se token i fingerprint stavljaju u header i šalju.

Kada HTTP zahtev dolazi sa frontend-a, uzimaju se token i fingerprint iz header-a. Dešifruje se token sa fingerprint-om i proverava da li je uloga pravilna i da li uloga ima potreban autoritet. Ako je sve uredu počinju se operacije za koje je zahtev namenjen.

## API Gateway

API Gateway (šljunak za API-jeve) je komponenta u arhitekturi softvera koja deluje kao posrednik između klijenata i više mikroservisa ili veb servisa. Nekoliko ključnih aspekata API Gateway-a:

1. Rutiranje (Routing): API Gateway omogućava usmeravanje zahteva sa klijenata ka odgovarajućim servisima na osnovu definisanih pravila. Ovo je posebno korisno u mikroservisnim arhitekturama gde postoji više servisa koji rade nezavisno.
2. Autentikacija i Autorizacija: API Gateway često rukuje procesom autentikacije (provera identiteta korisnika) i autorizacije (dodeljivanje prava pristupa) kako bi zaštitio API-jeve od neovlašćenog pristupa.
3. Kontrola pristupa (Access Control): API Gateway može implementirati politike kontrole pristupa kako bi ograničio pristup određenim resursima ili operacijama, obezbeđujući tako sigurnost sistema.

RBAC se koristiti na nivou API Gateway-a za kontrolu pristupa API resursima. To znači da se uloge dodeljuju korisnicima ili klijentima na osnovu njihovih privilegija i zaduženja. API Gateway zatim koristi ove informacije o ulogama kako bi autorizovao ili odbio pristup određenim API resursima.

## Mikroservisi

U Spring-u predstavljaju arhitekturni pristup u razvoju softvera koji podrazumeva izgradnju aplikacija kao skupa manjih, autonomnih servisa (svaki servis može biti razvijan, ažuriran i skaliran odvojeno od ostalih servisa), svaki fokusiran na određeni poslovni kontekst.

## Controller klase

### Opis

Kod većine metoda iznad njih ima provera da li korisnik ima autoritet (RBAC) za tu metodu. Za komunikaciju sa frontend-om koriste se DTO klase. One su skoro iste kao model klase (klase sa kojima se povezujemo sa bazom), jedina razlika je u tome što su uklonjena referenciranja koja dovode do beskonačnog pozivanja. Service klase zahtevaju model klase, zbog toga se koriste mapper klase koje konvertuju DTO klase u model klase. Mapper klase se pozivaju kod mnogih metoda unutar kontroler klasa.

### AuthenticationController

Služi za pozivanje metoda logovanja, registracije, verifikacije naloga i odjavljivanja. Samo kod odjavljivanja ima provera autorizacije. Metode se nalaze u CustomUserDetailsService klasi.

### KorisnikController

Služi za pozivanja metoda:

* prikaza svih mušterija, prodavaca, admina preko paginacije
* filtriranje korisnika
* povlačenje i vraćanje korisnika
* davanje specifičnog korisnika
* ažuriranje ličnih informacija vezanih za korisnika

Metode se nalaze u CustomUserDetailsService klasi.

### ProduktController

Služi za pozivanje metoda:

* kreiranje produkta
* dodavanje, izbacivanje i provera da li je u wishlist-u
* ažuriranje informacija vezanih za produkt
* davanje specifičnog produkta
* prikaz i filtriranje svih produkata preko paginacije
* prikaz i filtriranje istorije kupljenih produkata mušterije
* prikaz i filtriranje wishlist-a produkata mušterije
* prikaz i filtriranje svih produkata koji pripadaju prodavcu
* povlačenje i vraćanje produkta od strana prodavca i admina
* postavljanje akcije za produkt

Metode se nalaze u ProduktService klasi.

### RacunController

Služi za pozivanje metoda:

* kreiranje računa
* dodavanja artikla u račun
* ažuriranje računa
* prikaz i filtriranje svih računa
* prikaz i filtriranje računa mušterije
* davanje korpe
* uklanjanje artikla iz računa
* plaćanje

Metode se nalaze u RacunService klasi.

### RecenzijaController

Služi za pozivanje metoda:

* kreiranje recenzije
* ažuriranje recenzije
* davanje specifične recenzije
* provera da li ima recenzija za produkt
* prikaz i filtriranje recenzija za produkt
* prikaz i filtriranje svih recenzije mušterije
* prikaz i filtriranje svih recenzija
* brisanje recenzije

Metode se nalaze u RecenzijaService klasi.

### TipController

Služi za pozivanje metoda:

* pravljenje kategorije
* ažuriranje kategorije
* prikaz i filtriranje svih kategorija

Metode se nalaze u TipService klasi.

## Mapper

### Opis

Služi za konvertovanja DTO klasa u model klase i suprotno. One su:

* ArtikalMapper
* KorisnikMapper
* MusterijaMapper
* ProdavacMapper
* ProduktMapper
* RacunMapper
* RecenzijaMapper
* TipMapper

## Service

### Opis

U servis klasama se implementiraju gore navedene metode iz kontroler klasa. Komunikacija sa bazom se vrši preko metoda repository interfejsa. Te metode se pozivaju unutar servis klasa.

Kod nekih metoda unutar klasa CustomUserDetailsService, ProduktService, RacunService potrebno je slanje mejla. Pošto je operacija sa mejlom I/O bound daje se poseban thread za izvršavanje.

* CustomUserDetailsService

Pozivaju se metode iz KorisnikRepository, UlogaRepository, VerificationTokenRepository i ProduktRepository.

* ProduktService

Pozivaju se metode iz ProduktRepository, TipRepository, KorisnikRepository.

* RacunService

Pozivaju se metode iz RacunRepository, ArtikalRepository, ProduktRepository i KorisnikRepository.

* RecenzijaService

Pozivaju se metode iz RecenzijaRepository, KorisnikRepository, ProduktRepository.

* TipService

Pozivaju se metode iz TipRepository, ProduktRepository.

## Repository

### PostgreSQL

PostgreSQL je moćni i napredni open-source sistem za upravljanje bazama podataka (SGBD) koji se često koristi za čuvanje i manipulaciju podacima u raznim aplikacijama. Ključne karakteristike PostgreSQL-a uključuju:

* Relacioni model podataka: PostgreSQL je relacionalni SGBD, što znači da organizuje podatke u relacije (tabele) i koristi SQL (Structured Query Language) za upravljanje tim podacima.
* Podrška za kompleksne upite: PostgreSQL pruža napredne i kompleksne mogućnosti upita, uključujući spajanje tabela, podupite, podizraze, prozorske funkcije i još mnogo toga.
* Transakcioni model: PostgreSQL podržava ACID (Atomicity, Consistency, Isolation, Durability) princip transakcija, čime se garantuje pouzdanost i doslednost baze podataka.
* Funkcionalnosti za proširenje (extensibility): Ovaj SGBD omogućava korisnicima da definišu svoje tipove podataka, funkcije, operatore i jezike, čime se prilagodjavaju specifičnim potrebama aplikacija.

### JPA (Java Persistence API)

JPA (Java Persistence API): JPA je standardni Java API koji omogućava mapiranje Java objekata na relacijske baze podataka. Koristi se za definisanje entiteta, njihovih svojstava i relacija između njih. Model klase se pravi sa njim.

### JPA Repository

JPARepository je deo Spring Data JPA. To je interfejs koji pruža usluge za olakšavanje pristupa i manipulacije podacima u bazi podataka. Spring Data JPA automatski generiše SQL upite na osnovu potpisa metoda u interfejsu.

### JPQL (Java Persistence Query Language)

Ovo je upitni jezik koji se koristi u okviru Java Persistence API (JPA) za izvršavanje upita nad objektno-relacionim modelom podataka. JPA omogućava programerima da rade sa bazama podataka koristeći objektno-orijentisane principe (pišu upite koristeći entitete i njihove atribute umesto SQL upita koji rade sa tabelama i kolonama).

### Opis

Skup interfejsa preko kojih se komunicira sa PostgreSQL bazom podataka. Za neke zahteve se moralo dodatno napisati JPQL upit.

Repository klase su :

* ArtikalRepository
* KorisnikRepository
* ProduktRepository
* RacunRepository
* RecenzijaRepository
* TipRepository
* UlogaRepository
* VerificationTokenRepository