Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004:007
DOI: https://doi.org/10.24867/05OI02Bojkic

ANALIZA NAPREDNIH MOGUĆNOSTI RADNIH OKVIRA OTVORENOG KÔDA

ANALYSIS OF THE ADVANCED POSSIBILITIES OF OPEN SOURCE FRAMEWORKS

Marko Bojkić, *Fakultet tehničkih nauka, Novi Sad*

Oblast - INŽENJERSTVO INFORMACIONIH SISTEMA

Kratak sadržaj – *U ovom radu predstavljena je višeslojna veb aplikacija namenjena za podršku poslovanja ratarskog gazdinstva. Aplikacija sadrži implementaciju različitih koncepata i obrazaca. Aplikacija se sastoji iz serverske i klijentske strane, napisanih u programskom jeziku Java i TypeScript. Za skladištenje podataka aplikacije korišćen je sistem za upravljanje bazom podataka PostgreSQL.*

Ključne reči: *Spring, Angular, Java, TypeScript*

Abstract – *This paper presents a multilayered web application designed to support the farming business process. The application contains the implementation of different concepts and patterns. The application consists of server and client side, written in Java and TypeScrip programming languages. The PostgreSQL database management system was used to store the application data.*

Keywords: *Spring, Angular, Java, TypeScript*

1. UVOD

U okviru ovog rada, predstavljen je, po etapama, razvoj višeslojne veb aplikacije, u daljem radu „višeslojna aplikacija". Aplikacija predstavlja prototip složene veb aplikacije, namenjene za potrebe poslovanja ratarskog gazdinstva.

Serverski deo (engl. *Backend*) višeslojne aplikacije napisan je u *Java* programskom jeziku, a klijentski deo (engl. *Frontend*) u *TypeScript* programskom jeziku.

Osnovne tehnologije koje su korišćene su *PostgreSQL*, *Spring* i *Angular* 7.

Aplikacija je realizovana tako da su za svaku tabelu obezbeđene operacije za rad nad podacima. Ove operacije podrazumevaju čitanje, kreiranje, modifikaciju i brisanje.

Naredno poglavlje obuhvata prikaz i analizu ključnih koncepata veb aplikacije, sa osvrtom na bezbednost veb informacionog sistema, obrazac injektovanja zavisnosti, zatim reaktivno programiranje, te asinhroni način programiranja, sa analizom mehanizama implementacije. Potom u poglavlju **Error! Reference source not found.** opisane su tehnologije korišćene u kreiranju višeslojne aplikacije. Na samom kraju rada predstavljen je zaključak i spisak literature korišćen u ovom radu, kao i biografija autora.

2. KLJUČNI KONCEPTI

U okviru ovog poglavlja, prikazani su osnovni koncepti na kojima se zasniva višeslojna aplikacija. Pod ključnim konceptima podrazumevaju se bezbednost veb informacionog sistema, obrazac injektovanja zavisnosti, pojam reaktivnog programiranja, asinhrono izvršenje programa, te rutiranje zahteva klijentske aplikacije.

2.1 Bezbednost veb informacionih sistema

Kada je reč o bezbednosti veb orijentisanog informacionog sistema, dva aspekta su od ključne važnosti.

Prvi aspekt predstavlja kontrolu pristupa u zavisnosti od identiteta korisnika koji pristupa aplikaciji. Proces utvrđivanja identiteta korisnika koji pristupa sistemu naziva se autentifikacija (engl. *Authentication*) [1].

Navedeni proces se standardno vrši zahtevanjem da se korisnik pre korišćenja sistema prijavi na sistem slanjem kredencijala, odnosno korisničkog imena i lozinke. Svaki korisnik koji koristi aplikaciju može imati različita prava pristupa u aplikaciji. To podrazumeva i činjenicu da su određene funkcionalnosti dostupne samo određenim korisnicima. Stoga je neophodno, pri pristupu funkcionalnostima, prvo proveriti da li korisnik ima pravo da izvrši konkretnu funkcionalnost. Ovaj se proces naziva autorizacija (engl. *Authorization*). S obzirom da bi definisanje kontrole pristupa na nivou pojedinačnog korisnika bilo komplikovano, jer se za korišćenje sistema može prijaviti veliki broj korisnika, standardna realizacija je da se prava pristupa dodeljuju njihovim ulogama (npr. administrator, standardni korisnik, gost, ...). Svakom korisniku se dodeljuju uloge (engl. *roles*) koje mu pripadaju [1].

Drugi važan aspekt bezbednosti je kako se podaci u aplikaciji prenose. U slučaju veb aplikacija, podaci se preko mreže razmenjuju putem poruka između klijenta i servera. Međutim, tokom razmene postoji i opasnost da neko presretne mrežnu komunikaciju i pristupi podacima koje poruka, koja se razmenjuje, sadrži. Kako bi se komunikacija odvijala na siguran način, neophodno je izvršiti šifrovanje (engl. *Encryption*) poruke pre slanja, a potom i dešifrovanje (engl. *Decryption*) poruke nakon prijema [1].

Osnovna ideja šifrovanja poruka je transformacija poruke u drugu poruku, koja se može dešifrovanjem transformisati u izvornu poruku. Sam proces šifrovanja poruka se vrši zamenom svakog karaktera u poruci drugim karakterom prema nekoj definisanoj šemi šifrovanja. Parametar koji određuje pravilo transformacije izvornog podatka u šifrovani podatak naziva se ključ. Jedan od načina enkriptovanja korisničkih lozinki, biće prikazan u nastavku.

2.1.1. Načini autentikacije u veb aplikaciji

Autentifikacija se standardno vrši tako što klijent putem korisničkog imena i lozinke informiše server o svom identitetu. Za svaku akciju koju klijent izvršava, server mora posedovati informaciju o njegovom identitetu. HTTP protokol je *stateless* protokol, što znači da se svaki

---

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio prof. Đorđe Pržulj.

novi zahtev tretira nezavisno od prethodnih zahteva. To znači da ukoliko klijent uputi zahtev za logovanjem u kojem je poslao svoje kredencijale i nakon toga uputi zahtev za izvršenjem određene funkcionalnosti, ne postoji direktna veza između ta dva zahteva. Usled toga, server ne bi mogao da donese odluku da li korisnik ima pravo da drugi zahtev izvrši ili ne.

Postoji nekoliko načina autentifikacije:

- Osnovna autentifikacija – kod osnovne autentifikacije, klijent uz svaki zahtev u zaglavlju mora da dostavi i korisničko ime i lozinku korisnika koji zahtev izvršava. Time server za svaki zahtev jednostavno može da utvrdi ko je korisnik koji ga izvršava. Međutim, bez dodatnih mehanizama zaštite, osnovni tip autentikacije je vrlo nesiguran, s obzirom da se presretanjem zahteva može doći do informacije o korisničkom imenu i lozinci.
- Kolačić – drugi način rešavanja pomenutog problema je da klijent serveru pošalje korisničko ime i lozinku samo jednom, prilikom prijave na sistem, a za svaki naredni zahtev, server mora da utvrdi da li dolazi od prijavljenog korisnika. Jedan način da se ovo realizuje je korišćenje kolačića. Kolačić predstavlja *string* koji server izgeneriše prilikom prijave korisnika i koji se klijentu vrati kao odgovor na prijavu. Internet pretraživači standardno podržavaju korišćenje kolačića za komunikaciju između klijenta i servera. Kolačić dobijen od servera pretraživač skladišti u lokalnom računaru i šalje ponovo serveru pri svakom izvršenom zahtevu. Server mora posedovati odgovarajući mehanizam kako bi dobijeni kolačić povezao sa podacima o korisniku koji su poslati prilikom prijave. Ovo se najčešće rešava tako što server u memoriji sadrži mapu koja skladišti parove kolačić-korisnik. Na ovaj način, kada preuzme kolačić iz zahteva, server može da utvrdi na kojeg se korisnika odnosi i da sprovede kontrolu pristupa. Kolačić obično ima ograničen rok važenja i može se reći da kolačić identifikuje jednu sesiju korisnika.
- Autentifikacija bazirana na tokenima – token je, kao i kolačić, *string*, ali umesto identifikovanja sesije, token služi za identifikaciju korisnika direktno, na taj način što u sebi skladišti informacije o samom korisniku. Usled toga, server ne mora da skladišti dodatne informacije o sesiji korisnika, tako da je komunikacija potpuno *stateless*, što pruža bolje performanse servera. Takođe, još jedna prednost pri korišćenju tokena bila bi ta što se kolačićima automatski upravlja od strane veb pretraživača, što uvodi određena ograničenja prilikom upravljanja komunikacijom. JWT je *string* koji se sastoji iz zaglavlja (tip tokena i algoritam kriptovanja), glavnog sadržaja (subjekat na koga se token odnosi, uloge subjekta, rok trajanja tokena) i potpisa (*string* koji je generisan šifrovanjem zaglavlja, sadržaja i serverove tajne lozinke).

2.2 Injektovanje zavisnosti

Objektno-orijentisana paradigma implicira da se svaki realni sistem može posmatrati kao organizovani skup međusobno povezanih objekata, sa sopstvenim stanjima i ponašanjima, koji, zahvaljujući međusobnoj interakciji, mogu postići unapred definisane ciljeve sistema kojem pripadaju.

Kako bi svoje zadatke izvršili, objekti mogu zavisiti od drugih objekata. Zavisnost predstavlja slučaj u kojem je objektu, koji je instanca jedne klase, potreban drugi objekat, koji je instanca iste ili druge klase, da bi svoju funkcionalnost izvršio.

U softverskom inženjerstvu, dizajnerski obrazac predstavlja ponovljivo rešenje za probleme koji se često javljaju u dizajnu softvera [2].

Koristeći dizajnerski obrazac injektovanja zavisnosti, objekti se povezuju sa instancama klasa od kojih su zavisni. Injektovanje zavisnosti, kao jedan od mehanizama za postizanje principa inverzije kontrole, predstavlja dizajnerski obrazac pomoću kojeg jedna klasa zahteva zavisnosti od eksternih izvora, umesto da te zavisnosti kreira sama [3]. Inverzija kontrole se može definisati kao inženjerski princip koji prenosi kontrolu nad objektom ili delovima sistema na kontejner.

Prednosti upotrebe principa inverzije kontrole bile bi:

- lakša tranzicija između različitih implementacija.
- Veća modularnost programa.
- Lakše testiranje programa izolovanjem njegovih komponenti.

Dakle, injektovanje zavisnosti predstavlja vrlo značajan dizajnerski obrazac kroz koji se inverzija kontrole implementira. Ključna prednosti upotrebe injektovanja zavisnosti jeste slabo sprezanje. Primenom ovog obrasca, softversko rešenje postaje održivo, pri čemu se podrazumeva lakoća modifikovanja komponenti samog softverskog rešenja [4]. Komponente se mogu dodavati i testirati nezavisno od drugih komponenti, jer su labavo spregnute. Upotrebom ovog obrasca, testiranje je značajno pojednostavljeno jer omogućuje korišćenje *mock* objekata. *Mock* objekti su simulirani objekti koji imitiraju ponašanje realnih objekata u kontrolisanim uslovima. Najčešće se kreiraju kako bi testirali ponašanje nekih drugih objekata.

2.3. Reaktivno programiranje

Reaktivno programiranje je programska paradigma orijentisana ka protoku podataka i osluškivanju promena. Može se definisati kao programiranje sa asinhronim tokovima podataka. Koristeći reaktivni pristup u programiranju, tokovi podataka postaju "kičma" aplikacije. Događaji, poruke, pozivi, ali i greške će biti isporučeni tokovima podataka. Reaktivnim programiranjem posmatraju se tokovi podataka i reaguje se na emitovane (engl. *emitted*) vrednosti. U okviru aplikacije, kreiraju se različiti tokovi podataka, kao na primer na osnovu događaja na klik (engl. *Click events*), pojava HTTP zahteva, promena vrednosti varijable, merenja očitanih senzorom, ili bilo čega drugog što se menja ili dogodi. Naravno, ovo ima efekte i na samu aplikaciju time što ona postaje suštinski asinhrona.

2.4 Asinhrono izvršenje programa

U ovom poglavlju, prikazana je razlika između sinhronog i asinhronog načina izvođenja operacija. Takođe, prikazan je jedan od mehanizama radnog okvira *Angular* pod nazivom *Observable*, kroz koje je moguće asinhroni pristup realizovati.

U sinhronom načinu izvođenja operacija, obavlja se jedna operacija, te program čeka dok se ta operacija ne završi i zatim prelazi na sledeću.
Međutim, jedan od glavnih nedostataka sinhronog načina izvođenja operacija je gubitak vremena. Ako prva operacija traje neko vreme ili čeka dok se neki zahtev ne ispuni, bolje je krenuti dalje na sledeću operaciju. Ovaj način bi predstavljao asinhroni način obrade.

Tok može definisati kao niz događaja koji se emituju iz izvora. Tokovi mogu emitovati tri stvari: vrednosti (npr. povratne podatke iz API-ja), grešku (npr. 401 neovlašćeni pristup API-ju) i konačno "završni" signal koji označava da je tok završen.

RxJS je biblioteka za reaktivno programiranje pomoću *observable*-i [5]. U RxJS-u posmatrač se pretplaćuje funkcijom *subscribe()* na *observable* ili jednostavno rečeno, na tok. Ovaj posmatrač je u mogućnosti da reaguje na stavke ili vrednosti koje emituje *observable*, i na primer popuni tabelu dobijenim podacima. Funkcijom *subscribe()*, prilikom osluškivanja toka, vrši se tzv. pretplata na taj tok i nastavlja se primanje vrednosti sve dok se ne izvrši odjava ili kada tok emituje kompletnu poruku.

Na listingu 1, prikazana je servisna klasa *VehicleService*. *Readonly* API_URL_PAGINATION polje predstavlja putanju do URL-a na kojem se nalazi metoda na serverskoj aplikaciji koja isčitava sva vozila po određenom broju stranice. Promenljiva *dataChange*, koja je tipa *BehaviorSubject<Vehicle[]>*, služi za privremeno smeštanje podataka, i u okviru ovog primera, osluškuje *observable*-u, odnosno tok. Podaci vraćeni iz *observable*-e će biti smešteni u ovu promenljivu.

Za metodu *getVehiclesByPage()*, kao povratni tip, definisan je upravo *Observable*. Funkcijom *subscribe()* izvršena je pretplata na *observable*-u. Metodi *getVehiclesByPage()* se kao parametar prosleđuje stranica po osnovu koje se isčitavaju vozila, te broj stavki koje će biti izlistane po jednoj stranici. Kako bi metoda *getVehiclesByPage()* vratila sva vozila, poziva se funkcija *next()* za svaku vrednost koja se emituje iz toka, odnosno za svako vozilo koje bude vraćeno. *Observable*-a će *next()* funkcijom podatke vraćati sve dok oni postoje, odnosno dok tok ne emituje kompletnu poruku. Na kraju, metoda vraća *dataChange*, prilikom čega se poziva funkcija *asObservable()*, koja kreira novi *observable* sa nizom vozila kao subjektom.

Na listingu 2, predstavljena je komponenta *MechanizationComponent*. U okviru komponente, kreirana je *loadData()* metoda. Ova metoda kroz polje *vehicleService*, koji je tipa *VehicleService*, a koji je u ovu klasu injektovan, poziva metodu *getVehiclesByPage()* iz te servisne klase. Kao parametri ovoj metodi prosleđeni su indeks prve stranice, odnosno vrednost 0, te broj stavki koji će po stranici biti vraćene, odnosno vrednost 5. Metoda *getVehiclesByPage()* u promenljivu *vehicles* dodaje sve vrednosti koje ova metoda vraća.

2.5 Paginacija

U okviru ovog poglavlja, predstavljen je koncept paginacije (engl. *pagination*), kao i razlike između paginacije na serverskoj i klijentskoj strani.

Paginacija predstavlja prikaz podataka u okviru više različitih stranica ili sekcija, kako bi se na taj način obezbedila veća preglednost i raspodela podataka po određenom uslovu ili prioritetu na korisničkom interfejsu. Paginaciju je moguće implementirati i u okviru serverske i klijentske aplikacije. Paginacija na klijentskoj aplikaciji predstavlja slučaj kada se podaci preuzmu i potom klijentska aplikacija izvrši segmentaciju podataka na više stranica. Paginacija na serverskoj aplikaciji predstavlja slučaj kada klijentska strana obezbedi ključ koji se prosledi serveru i tada server odabere konkretnu stranicu sa podacima.

Važno je napomenuti da je paginacija na serverskoj aplikaciji pogodna u slučajevima ogromnih količina podataka, iz razloga što je količina podataka koja se prosledi klijentskoj aplikaciji znatno manja nego u slučaju kada klijentska aplikacija povuče sve podatke koji postoje pa potom ona izvrši paginaciju. Takođe, smanjuje se količina potrebne memorije za klijentsku aplikaciju.

```
@Injectable()
export class VehicleService {
    readonly API_URL =
    'http://localhost:8083/vehicle/';
    readonly API_URL_PAGINATION =
    'http://localhost:8083/vehicleList/';
    dataChange: BehaviorSubject<Vehicle[]> = new
    BehaviorSubject<Vehicle[]>([]);
  constructor(private httpClient: HttpClient) {
}
    public getVehiclesByPage(page: number, size:
    number): Observable<Vehicle[]> {
        this.httpClient.get<Vehicle[]>(this.API_U
        RL_PAGINATION + '?page=' + page +
        '&size=' + size).subscribe(data => {
      this.dataChange.next(data);
  });
   return this.dataChange.asObservable();
  }
  public addVehicle(vehicle: Vehicle): void {
        this.httpClient.post(this.API_URL,
        vehicle).subscribe();
  }
  public updateVehicle(vehicle: Vehicle): void {
        this.httpClient.put(this.API_URL,
        vehicle).subscribe(data => {
     });
  }
  public deleteVehicle(id: number): void {
        this.httpClient.delete(this.API_URL +
        id).subscribe();
  }
}
```

Listing 1. *VehicleService* klasa

```
@Component({
 selector: 'app-mechanization',
 templateUrl: './mechanization.component.html',
 styleUrls: ['./mechanization.component.css']
})
export class MechanizationComponent implements
OnInit {
 private vehicles: any;
 this.page = 0;
 this.size = 5;
 constructor(private vehicleService:
VehicleService) {}
 public loadData() {
        this.vehicleService.getVehiclesByPage(thi
        s.page,  this.size).subscribe(res => {
    const content = 'content';
    const totalPages = 'totalPages';
    this.vehicles = res[content];
         this.pages = new Array(res[totalPages]);
```

```
  });
 }
}
```

Listing 2 - *MechanizationComponent* klasa

3. TEHNOLOGIJE

U okviru ovog poglavlja, biće prikazane i opisane tehnologije korišćene prilikom kreiranja višeslojne aplikacije u ovom radu. Baza podataka kreirana je pomoću *PostgreSQL* objektno-relacionog sistema za upravljanje bazama podataka. Serverska aplikacija kreirana je u *Spring* radnom okviru, dok je za kreiranje klijentske aplikacije korišćen *Angular* radni okvir.

3.1 PostgreSQL

PostgreSQL je moćan i otvoren objektno-relacioni sistem za upravljanje bazama podataka, koji je u upotrebi već 30 godina, te je tokom godina stekao reputaciju pouzdanosti i visokih performansi . *PostgreSQL*-om ne upravlja nijedna korporacija ili privatni entitet i njegov izvorni kôd je dostupan besplatno.

PgAdmin je najpopularnija otvorena platforma za administraciju i korišćenje *PostgreSQL*-a. *PgAdmin* je vrsta klijenta. Omogućava manipulaciju podacima na jednoj ili više instanci *PostgreSQL*-a.

3.2 Spring radni okvir

Spring predstavlja radni okvir, baziran na *Java* programskom jeziku, koji obezbeđuje infrastrukturu za razvoj aplikacije [6].

Integracija biblioteka *Spring*-a u aplikaciju je pojednostavljena uz korišćenje alata, kao što je, na primer, *Maven*, koji je i korišćen u višeslojnoj aplikaciji.

Spring Boot predstavlja okvir koji pruža mogućnost za izradu samostalnih aplikacija, spremnih za produkciju. Navedeni okvir primenjen je u izradi višeslojne aplikacije. Ovaj okvir poseduje ugrađeni veb server *Tomcat*. Ovaj okvir poseduje ugrađeni veb server *Tomcat*. Klasa višeslojne aplikacije, koja sadrži *main()*, naziva se *BackendMasterApplication.java*, te ona predstavlja početnu tačku izvršenja programa. *Spring Boot*, takođe, konfiguriše inicijalni *pom.xml* fajl, za konfiguraciju sa *Maven* alatom.

Spring Security je moćan i prilagodljiv okvir za autentifikaciju i kontrolu pristupa. *Spring Security* je okvir koji se fokusira na pružanje i autentifikacije i autorizacije *Java* aplikacijama.

U okviru višeslojne aplikacije u ovom radu, implementirani su REST kontroleri. REST kontroleri komuniciraju sa klijentskom stranom aplikacije preko HTTP/HTTPS protokola. Podaci koji se isporučuju su u *JSON* formatu. Za konfiguraciju se koriste anotacije (engl. *annotation*).

Za upravljanje podacima koriste se repozitorijumi (engl. *repositories*). Repozitorijumi nasleđuju klasu *JpaRepository*, čime sadrže već implementirane, podrazumevane metode za: čuvanje, brisanje i čitanje entiteta.

Kao radni okvir objektno-relacionog mapiranja (ORM), *Hibernate* se brine za perzistentnost podataka jer se odnosi na relacione baze podataka (putem JDBC-a). Pored sopstvenog "matičnog" (engl. *native*) API-ja, *Hibernate* je takođe implementacija *Java Persistence API* (JPA) specifikacije.

3.3 Angular radni okvir

Angular predstavlja *TypeScript MVC* (*Model-View-Controller*) radni okvir, dizajniran radi kreiranja jednostraničnih veb aplikacija, lakih za održavanje, sa pravilnom strukturom.

Angular radni okvir koristi *TypeScript* programski jezik za razvoj veb aplikacija.

TypeScript predstavlja programski jezik otvorenog kôda. *TypeScript* je nadskup *JavaScript* programskog jezika, te dodaje opcionu statičku tipizaciju i objektnu orijentisanost. Kompajlira se u *JavaScript* jezik.

4. ZAKLJUČAK

U okviru ovog rada, prikazan je razvoj višeslojne aplikacije po etapama, sa prikazom različitih ključnih koncepata, kao i konkretnih vidova njihove implementacije.

Razvoju veb, ali i svih drugih vrsta aplikacja, treba pristupiti sa velikom pažnjom, vrlo studiozno, počevši od odabira tehnologija koje će se koristiti, preko analize koncepata koji će biti primenjeni u okviru nje, potom njenog kreiranja, te do završne faze, procesa održavanja aplikacije.

Prikazana aplikacija može biti i dodatno razvijana. Da li će do toga doći, zavisi od rasta i razvoja samog preduzeća, kao i njegovih poslovnih rezultata. Pod dodatnim razvijanjem aplikacije, podrazumeva se pre svega proširenje njene baze podataka, a shodno tome i konfigurisanje serverske i klijentske aplikacije.

5. LITERATURA

[1] Savić Goran, Segedinac Milan, Tehnologije veb aplikacija, Novi Sad: FTN, 2018.

[2] A. Shvets, Design Patterns, 2015.

[3] M. Fowler, „Inversion of Control Containers and the Dependency Injection pattern," 2004.

[4] N. E. Matinlassi Mari, „The Impact of Maintainability on Component-based Software Systems," VTT Technical Research Centre of Finland, 2003.

[5] M. Clow, Observers, Reactive Programming, and RxJS., Berkeley: Apress Media LLC, 2018.

[6] J. Rod, „The spring framework–reference documentation.," 2004. [Na mreži]. Available: https://docs.spring.io/spring/-docs/3.2.18.BUILD-SNAPSHOT/spring-framework-reference/pdf/spring-framework-reference.pdf. [Poslednji pristup 28 Avgust 2019].

Kratka biografija:

Marko Bojkić je rođen 26. oktobra 1994. godine u Novom Sadu. Godine 2013. upisao je Fakultet tehničkih nauka u Novom Sadu, odsek Inženjerski menadžment. Osnovne akadmske studije završio je 2017. godine. Master akademske studije upisao je 2017. godine, odsek Inženjerstvo informacionih sistema.