



UNIVERZITET U NOVOM SADU
PRIRODNO- MATEMATIČKI FAKULTET
DEPARTMAN ZA MATEMATIKU I INFORMATIKU

CAR INSURANCE

Seminarski rad iz predmeta „Informatički projekat“

Mentor profesor: Doni Pracner

Student: Jelena Capik 645/18

Mentor prakse: Milan Jović

Synechron	3
Dolazak do prakse u Synechron-u	3
Lični utisak:	4
Tehnologije:	4
Java	4
Maven	6
Spring	7
Hibernate	8
Git.....	9
Angular5.....	12
Izrada Aplikacije	15
Baza.....	15
Veze između entiteta:.....	16
Backend.....	19
Repository	21
Service.....	22
Controller	26
DTO	27
Front.....	27
Zaključak	35
Resursi.....	36

The logo for Synechron is a yellow square with the word "Synechron" in a stylized, italicized font. The letter 'o' is replaced by a circular icon containing a stylized 'S'.

Synechron

Synechron je kompanija za informacione tehnologije koja je osnovana 2001.godine u Njujorku. Kompanija je fokusirana na industriju finansijskih usluga: Bankarstvo, osiguranje, digital. Osnivači kompanije su: Faisal Husain, Zia Bhutta i Tanveer Saulat koji su vremenom razvijali poslove kompanije i otvarali nove kancelarije u SAD-u, Australiji, Kanadi, Velikoj Britaniji, Japanu, Holandiji, Singapuru, Ujedinjenim Arapskim Emiratima, Irskoj, Nemackoj, Svajcarskoj, Luksemburgu, Italiji, Francuskoj...Novosadska kancelarija kompanije Synechron počela je sa radom u julu 2014. godine sa samo troje zaposlenih na jednom projektu, dok sada kompanija ima više od 130 zaposlenih koji su angažovani na preko 20 projekata. Synechron trenutno predstavlja jednu od najbrže rastućih IT kompanija u Srbiji. Sada Synechron ima preko 5500 zaposlenih i kompanija se i dalje razvija i teži otvaranju dodatnih razvojnih centara.

Dolazak do prakse u Synechron-u

Synechron je kompanija koja ima dugogodišnju saradnju sa Prirodno-Matematičkim fakultetom u Novom Sadu, koja omogućava da studenti koji žele mogu doći na praksu u firmu u trajanju od 2 meseca. Radno vreme je fleksibilno, što znači da dolaske i odlaske sam praktikant određuje.

Za vreme trajanja prakse radite u timu koji se sastoji dva člana zbog situacije sa Covidom, imate mentora koji vas vodi kroz projekat i koji je tu za sva vaša pitanja.

Praktikant radi 4 sata svakog dana, praksa se odvija u kancelarijama firme i online. Kompanija Synechron je organizovala praksu Web-Full Stack developer što znači da se praktikant bavi razvojem aplikacije od samog početka. Njegov zadatak je da modelira bazu, implementira servisni deo aplikacije u Javi i implementira sam dizajn aplikacije u Angularu.

Prednost kompanije Synechron u odnosu na druge firme koje se nalaze u Novom Sadu je da veliki broj asistenata i profesora sa PMF-a rade baš u Synechronu, gde sam student ima mogućnost da dobije pouzdane informacije o samoj kompaniji kao i direktnu preporuku profesora za studenta koji želi da realizuje praksu u kompaniji.

Lični utisak:

Do same prakse sam došla uz pomoć jednog od asistenata sa PMF-a. Poslala sam mejl sa CV-em i on me je uputio ka drugoj osobi iz same firme koja mi je dala detaljnije informacije. Praksu sam počela sredinom oktobra i trajala mi je tačno dva meseca. Praksa je organizovana da se praktikanti podele u timove, u našem slučaju, po dvoje. Dosta stvari koje se zahtevaju na praksi smo radili na samom fakultetu u možda manjem intezitetu. Kroz praksu i rad sa mentorom smo naučili neke “programerske cake”, nove tehnologije kao i malo promenili način razmišljanja i dolaska do ideja za rešenje. Jedina mana prakse po mom mišljenju je što traje kratko. Nema dovoljno vremena da se nove tehnologije lepo shvate i sam projekat lepo odradi, bar kada je reč o nama samim početnicima koji ništa van fakulteta nisu radili.

Tehnologije:

Java



Java je objektno-orjentisan jezik, koji je razvila kompanija Sun Microsystems početkom 1990-ih godina. Java je jedan od najaktuelnijih programskih jezika današnjice. Java ima kompletno svoju sopstvenu sintaksu i nije derivat, niti verzija bilo kog drugog programskog jezika. Važno je reći da je Java case sensitive programski jezik, što u prevodu znači da pravi razliku kod velikih i malih slova u pisanju komandi, dok kod nekih drugih jezika to i nije slučaj, kao na primer SQL. Dakle u Javi neće biti svejedno da li ste napisali System ili system, prvo će biti u redu, dok će drugo praviti grešku prilikom pokretanja samog programa.

Programski jezik Java treba da bude:

- jednostavan - da bude sistem u kome bi se lako programiralo, bez potrebe za komplikovanim uhadavanjem i koji koristi postojeći način razmišljanja. Sintaksa jezika Java je unapređena verzija sintakse C++
- objektno orijentisan - objektno orijentisano projektovanje predstavlja tehniku programiranja fokusiranu na podatke(objekte) i na interfejse ka tim objektima
- distribuiran - Java poseduje iscrpnu biblioteku rutina za rad sa TCP/IP protokolima, kao što su HTTP i FTP. Java aplikacije mogu da pristupaju objektima preko mreže i preko URL-a, sa podjednako lakoćom kao da pristupaju lokalnom sistemu datoteka
- robustan - Java je namenjena za pisanje programa koji moraju biti pouzdani na mnogo načina. Ističe se u ranoj proveru mogućih problema, kasnijoj dinamičkoj proveru(tokom izvršavanja) i eliminaciji situacija u kojima lako dolazi do pojave grešaka
- bezbedan - Java je namenjena korišćenju u mrežnim/distribuiranim okruženjima. Prema tome, mnogo je truda uloženo u bezbednost. Omogućava konstrukciju sistema koji su zaštićeni od virusa i zlonamerne modifikacije. Od samog početka, Java je projektovana da potpuno onemogući određene vrste napada, kao što su prekoračenje izvršnog steka, pristup memoriji izvan dela dodeljenog procesu, čitanje ili upisivanje datoteka bez dozvole...
- neutralan - kompajler stvara objektnu datoteku, čiji je format nezavisan od operativnog sistema na kome se pokreće. Kompajlirani kod se može izvršavati na mnogim procesorima, pod pretpostavkom prisustva izvršnog sistema Java. Java kompajler ostvaruje ovo tako što generiše bajtkod instrukcije, koje nemaju nikakve veze sa arhitekturom korišćenog računara, već se podjednako lako interpretiraju na svakoj mašini, a isto tako lako se prevode i u odgovarajući mašinski kod.
- prenosiv - ovde ne postoje aspekti koji su zavisni od implementacije. Veličine primitivnih tipova su fiksne, kao i njihovo ponašanje u aritmetici. Takođe, biblioteke koje su deo sistema definišu interfejse koji su prenosivi
- interpretiran - isti Java bajtkod se može izvršavati na svakom računaru za koji postoji Java interpreter. Budući da je linkovanje postepen i lakši postupak, sam razvoj može biti brži
- performantan - mada su performanse prevedenog bajtkoda obično više nego dovoljne, postoje situacije kada su potrebne bolje performanse. Bajtkod može da bude preveden tokom izvršavanja u mašinski jezik
- višenitan - prednosti višenitne obrade su bolji interaktivni odzivi i ponašanje u realnom vremenu
- dinamičan - Java je projektovana tako da se prilagođava okruženju koje se stalno unapređuje. Biblioteke mogu slobodno da dodaju nove metode i polja, bez uticaja na klijente. U Javi je prilično jednostavno pronalaženje informacija prilikom izvršavanja programa.

Maven



Apache Software Foundation je razvio Maven. Maven se temelji na POM-u (Project Object Model).

Maven se koristi za izgradnju i upravljanje projektima koji se temelje na Javi. Pomaže Java programeru pri razumevanju bilo kojeg Java projekta. Maven je alat koji se koristi pri razvoju aplikacije u cilju lakše integracije sa već postojećim bibliotekama klasa. Za nas su najbitnije sledeće funkcionalnosti:

1. Pravljenje gotovih šablona aplikacije,
2. Ubacivanje svih mogućih biblioteka za rad sa Spark-om i svih njihovih depedency-ja jednostavnim konfigurisanjem pom.xml datoteke.

Maven obavlja mnogo korisnih zadataka kao što su:

- Jednostavno možemo izgraditi projekat,
- Možemo dodati JAR i druge depedency projekta,
- Pruža nam informacije o projektu (log dokumenta, listu depedency-ja, izveštaje...),
- Pomaže nam da izradimo bilo koji broj projekata u izlazne tipove poput JAR, WAR, ...

Maven je alat sa kojim se radi iz konzole: mvn install i mvn test. Mnogi IDE u Javi imaju podršku za Maven. Maven je interne instalacije, tj. Ne treba instalirati na računaru i ima GUI podršku kroz menije, prozore, itd.

Standardna struktura projekta;

- src/main/java - Izvorni kod običnih Java klasa,
- src/main/resources - Resursi neophodni za rad,
- Src/test/java - Izvorni kod Java test klasa,
- src/test/resources - Resursi neophodni za testove,
- pom.xml - Maven konfiguracioni fajl,
- target - Putanja gde se smešta izlaz – JAR sa spakovanim projektom, rezultati testova, dokumentacija.

Rad sa drugim bibliotekama:

- Pronalaženje u Maven Central repozitorijumu preko 3 koordinate:
<https://search.maven.org/>
- Ili u nekom drugom Maven repozitorijumu
- Unošenje u pom.xml kao „dependency“

```
<dependency>  
  <groupId>dom4j</groupId>  
  <artifactId>dom4j</artifactId>  
  <version>1.6.1</version>  
</dependency>
```

Spring



Spring je okvir za Java platform, koji obezbeđuje infrastrukturu za razvoj aplikacije. Zahvaljujući tome, softverski inženjeri mogu da se fokusiraju na razvoj biznis zahteva aplikacije. Spring je po dizajnu modularan.

Nudi više od dvadeset projekata, koji obezbeđuju različite infrastrukture:

- ♦ osnovni Spring okvir (engl. Spring framework),
- ♦ okvir koji postavlja i podiže aplikaciju (engl. Spring Boot),
- ♦ okvir za rad sa bazama podataka (engl. Spring Data),
- ♦ okvir koji obezbeđuje alate za rad u oblaku (engl. Spring Cloud),
- ♦ okviri za sigurnost aplikacije (engl. Spring Security, Spring LDAP),
- ♦ okviri za upravljanje porukama (engl. Spring Integration, Spring AMQP) i drugi.

Osnovni Spring okvir (engl. Spring Core) omogućava kreiranje skladišta objekata, korišćenjem dizajna uzorka Inverzija kontrole (engl. Inversion of control container).

Spring minimizuje svoje zavisnosti ka drugim bibliotekama i okvirima. Jedina zavisnost osnovnog Spring okvira je ka biblioteci za upis u dnevnik aktivnosti (engl. logging). Integracija Springa u aplikaciju je jednostavna uz korišćenje alata, kao što je, na primer, Maven (što je korišćeno i u prototipskoj aplikaciji).

Spring Boot je okvir koji omogućava kreiranje infrastrukture za samostalne aplikacije, spremne za produkciju. Ima ugrađen veb server (Tomcat, Jetty ili Undertow), pa je aplikaciju dovoljno pokrenuti kao Java aplikaciju. Spring Boot, takođe, postavlja inicijalni pom.xml fajl, za konfiguraciju sa Maven alatom.

Spring Security je okvir koji nudi rešenje za autentifikaciju i autorizaciju aplikacije. Spring Boot okvir, kao podrazumevanu autentifikaciju, implementira osnovni tip Spring Security okvira. . Za implementaciju REST kontrolera, koristi se Spring Boot MVC okvir. REST kontroleri komuniciraju sa klijentskom stranom aplikacije preko HTTP/HTTPS protokola. Podaci su u JSON formatu. Za konfiguraciju se koriste anotacije. Na primer, anotacija @RestController označava da je klasa REST kontroler; anotacija @RequestMapping vezuje kontroler za URL preko koga je dostupan. Ista anotacija sa odgovarajućim atributima se koristi i na nivou metoda kontrolera da označi URL do metode, kao i tip zahteva: . GET, POST, DELETE i PUT.

Hibernate



Hibernate omogućava Java programerima da u svojim aplikacijama implementiraju poslovnu logiku, dok se operacije niskog nivoa, kao što su čitanje, skladištenje, brisanje i menjanje podataka, izvršavaju u pozadini, čime se veliki deo posla olakšava.

Hibernate je proizvod kompanije JBoss (kompanija koja pripada Red Hat-u). Hibernate je servis za objektno-relaciono perzistiranje podataka i dizajn upita.

Za Hibernate okvir se može reći da je jedno od najmoćnijih i najfleksibilnijih rešenja za objektnorelaciono perzistiranje podataka, kompleksni projekat koji namjerava da postane kompletno rešenje za perzistenciju podataka u Javi. Predstavlja medijatora između aplikacije i baze podataka, a ostavlja programera da se koncentriše na poslovne procese aplikacije, dok neke procese izvršava sam alat.

Hibernate okvir samostalno mapira podatke iz objektnog modela u relacioni model ili obrnuto, tako da programer ne mora da razmišlja o ovom poslu. Hibernate eliminiše ponovno i zamorno pisanje koda i ostavlja programerima vremena za rešavanje poslovnih problema.

Bez obzira koju strategiju razvoja aplikacije koristimo odozgo na dole, počevši od rešavanja poslovnih problema i izrade domenskog modela, ili odozdo na gore, počevši od postojeće šeme baze podataka. Hibernate se tipično koristi za Swing aplikacije, Servlet bazirane aplikacije, J2EE aplikacije koje koriste EJB (Entity Java Beans). EJB 3.0 Java Persistence API(JPA) koristi Hibernate 3.0 tak o da značajno smanjuje perzistentni model u EJB standardu.

HIBERNATE ANOTACIJE

Hibernate zahtjeva meta podatke koji vrše transformaciju od jedne reprezentacije podataka u drugu. Hibernate obezbeđuje, pored xml fajlova, i meta podatke za mapiranje u vidu anotacija. Iznad svake domenske klase se postavlja anotacija `@Entity`, što znači da svaka domenska klasa predstavlja jedan entitet (objekat) u sistemu. Sljedeća anotacija koja se postavlja iznad klase je `@Table(applyTo = "naziv tabele")`. Njome se povezuje entitet sa odgovarajućom tabelom u bazi. Sljede neke od anotacija za attribute domenske klase: `@Id` – Identifikator identiteta `@GeneratedValue` – Određuje strategiju za generisanje identifikatora. Neke od strategija su: `AUTO`, `TABLE`, `IDENTITY`, `SEQUENCE`, itd `@Column(name = "ID")` – Anotacija za kolonu Anotacija se može staviti na atribut ili na getter atributa.

HIBERNATE UPITNI JEZIK

Hibernate Query Language (u daljem tekstu HQL) je objektno orijentisan upitni jezik. Sličan je SQL-u, ali pored operacije nad tabelama i kolonama, HQL radi sa perzistentnim objektima i njihovim atributima. HQL-om se mogu dobijati podaci iz baze (select naredba) ili se mogu ažurirati podaci u bazi (inset, update, delete). HQL-om se ne može promeniti struktura baze (alter table, alter column, itd). HQL je jezik sa svojom sintaksom i gramatikom. HQL upiti se prevode u SQL upite, a Hibernate okvir ima mogućnost da direktno koristi SQL upite. HQL se preporučuje kada je god to moguće kako bi se izbegli problemi sa portabilnošću SQL-a sa bazom podataka. HQL je mnogo kompaktniji jezik od SQL jer koristi informacije definisane u Hibernate mapiranju.

Git



Git je distribuirani sistem:

- Ne postoji centralni repozitorijum
- Svako ima lokalni repozitorijum
- Moguće je praviti lokalne brenčeve (i jako važno)
- Lako je deliti kod
- Odličan za model zajedničkog razvoja softvera koji postoji na projektima otvorenog koda

Sve se radi preko git komande. Git ima mnogo komandi koje se pozivaju sa `$ git <komanda>`. Primeri komandi: `clone`, `checkout`, `branch` i dr. Pomoć u vezi komande `$ git help <komanda>`. Na početku rada na projektu potrebno je klonirati repozitorijum. Sa gitom dobija se potpuna kopija repozitorijuma, uključujući i istoriju, što omogućava izvršavanje većine operacija u offline režimu.

```
$ git init
Initialized empty Git repository in /tmp/tmp.IMBYSY7R8Y/.git/
$ cat > README << 'EOF'
> Git is a distributed revision control system.
> EOF
$ git add README
$ git commit
[master (root-commit) e4dcc69] You can edit locally and push
to any remote.
 1 file changed, 1 insertion(+)
 create mode 100644 README
$ git remote add origin git@github.com:cdown/thats.git
$ git push -u origin master
```

Klonirani repozitorijum će se vremenom menjati.

Preuzimanje izmena urađenih na serveru se postižu uz pomoć komande **`$ git pull`**.

Dobavljanje novih promena na udaljenom repozitorijumu **`$ git fetch`**.

Spajanje promena u trenutni brenč **`$ git merge`**.

Za početak rada najbolje je napraviti novi brenč:

- ☺ Postoji samo u lokalu, niko drugi ga ne vidi
- ☺ Brzo je
- ☺ Omogućava podelu rada na različite celine
- ☺ Omogućava isprobavanje nove funkcionalnosti i odbacivanje ukoliko nije zadovoljavajuće
- ☺ Nije skupa operacija (i ako je zadatak mali i brzo se završi treba napraviti brenč)

Kreiranje brenča **\$ git branch <ime brenča>**.
Prelazak na brenč **\$ git checkout<ime brenča>**.

Provera statusa radne kopije **git status**.

\$ git init - Ova komanda praktično pretvara postojeći direktorijum u repozitorijum. U osnovi on govori Git-u da započne praćenje svih fajlova i foldera unutar repozitorijuma.

Razvojno okruženje Git-a se sastoji od tri sekcije. Važno je da ih ovde prođemo kako biste bolje razumeli sam koncept Git-a jer sve dalje aktivnosti zavise od njega.

1. **Radni direktorijum** se odnosi na trenutno stanje fajlova i foldera unutar vašeg sistema fajlova. U ovom delu Git još uvek ne prati vaše fajlove.
2. **Staging area** je mesto u koje smeštate fajlove pre nego ih sačuvate u repozitorijumu. To je kao neka privremena lokacija za vaze foldere i fajlove pre nego pokrenete commit. Fajlove unutar staging area možete jednostavno dodavati i uklanjati. Ako ste modifikovali neke fajlove i želite da ih dodate u staging area jednostavno izvršite sledeću komandu: **\$ git add**.
3. **Repozitorijum** sadrži sve vaše commit-ovane fajlove. Sve ove informacije Git čuva u skrivenom folderu pod nazivom .git. Kada commit-ujete nešto, šta god da je u vašoj staging area se trajno čuva u repozitorijumu. Možda je lakše da commit shvatite kao checkpoint. Kada commit-ujete on poredi vaš prethodni checkpoint sa sadašnjim i tako čuva samo modifikovane fajlove. Dobra je praksa da prilikom commit-ovanja pokrenete komandu sa kratkom porukom šta je izmenjeno:

\$ git commit -m "ovde unesite kratak opis izmene"

Commit poruke su veoma korisne (nešto slično komentarisanim redovima prilikom pisanja koda), jer ostalim članovima tima daju više informacija o tome šta ste menjali).

Ako ste kreirali repozitorijum prvo na vašem lokalnom računaru i nakon toga želite da ga povežete sa serverom, onda je potrebno da pokrenete sledeću komandu:

\$ git remote add origin <https://github.com/nmh/bootstrap.git>

Sada ste povezali vaš lokalni repozitorijum sa udaljenim. Vreme je da upload-ujete izmene na server. Evo sintakse koju koristite u ovom slučaju:

\$ git push [remote_name] [branch_name]

Angular5

Angular je kreiran od strane kompanije Google kao platforma za kreiranje web aplikacija. Angular je razvojno okruženje koje se koristi za kreiranje klijentskog dela web aplikacije, korišćenjem HTML-a i CSS-a i TypeScripta. Svaka Angular aplikacija se sastoji od modula (NgModule), koji osnovni gradivni blok svake angular aplikacije. Moduli su logičke celine angular aplikacije, blokovi funkcionalnosti (koreni modul, modul za rutiranje i sl.). Modul se sastoji od komponenti, svaka komponenta obezbeđuje deo funkcionalnosti za aplikaciju.



JavaScript je programski jezik koji ne može da se samostalno koristi već je za njegovo izvršavanje potrebno specijalno okruženje tzv. **“JavaScript runtime environment”**. Najpoznatiji *“JavaScript runtime environment”* je browser, ali ukoliko želimo da izvršavamo JavaScript i van browser-a tu na scenu nastupa **node.js** kao drugi *“JavaScript runtime environment”*.

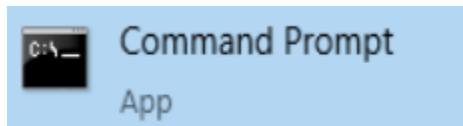


Kada završite sa instalacijom potrebnih programa, potrebno je instalirati Angular CLI (Command line interfejs)

```
npm install -g @angular/cli
```

Kao što naziv govori, to je alat komandne linije za kreiranje angular aplikacija. Preporučuje se korišćenje angular CLI-a da ne bismo uzaludno trošili vreme na instaliranje i konfigurisanje potrebnih dependency-ja i povezivanja svega.

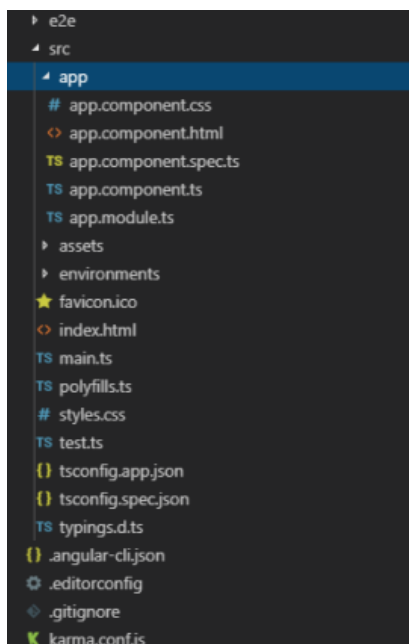
Kako bi kreirali novi aplikaciji potrebno je da koristeći Command Prompt u Windowsu se pre svega pozicioniramo na folder gde želimo da kreiramo novu aplikaciju I da komandom ng new kreiramo novu Angular aplikaciju.



Aplikaciju možete otvoriti pretragom preko start menija.

Nakon otvaranja Visual Studio Code-a sa leve strane pojaviće nam se folderi I fajlovi koje je angular ubacio u naš projekat. Trenutno nas zanima šta se prikazuje na stranici koja nam je otvorena u browseru. Za to je zadužena glavna komponenta app.component, koji možemo naći tako što ćemo ući u folder src/app. U app folderu nam se prikazuje pet fajlova, ali nas trenutno interesuju samo par fajlova u okviru projekta:

- app.component.css (CSS za komponentu)
- app.component.html (template komponente, napisan u HTML)
- app.component.ts (klasa komponente napisana u TypeScript-u u kojoj se definiše logika komponente)
 - app.module.ts (globalna modul klasa u kojoj importujemo sve što nam je potrebno za aplikaciju)
- package.json (u njemu se beleže sve eksterne biblioteke koje su potrebne za rad ove aplikacije)



Ostale korisne ng komande:

- **ng add** – dodajemo npm paket neke eksterne biblioteke i konfiguriše je unutar projekta.
- **ng build** – ažurira kod i promene u aplikaciji
- **ng help** – prikazuje koje su ng komande dostupne.
- **ng** (naziv komande) –help - dobijamo informacije o komandi.
- **ng generate** – generiše novi deo aplikacije (komponenta, servis, pipe, itd.). Takođe možemo da koristimo ng g (vrsta dela) kao skraćenicu.
- **ng new** – kreira novu aplikaciju. Format je ng new (ime aplikacije). Skraćenica je ng n (ime aplikacije).
- **ng update** – ažurira verziju aplikacije i sve njene zavisnosti.
- **ng serve** – ažurira kod i promene u aplikaciji, ali takođe i “servira” aplikaciju na default <http://localhost:4200/>. Ako koristimo ng serve –open, nakon build-a će se otvoriti u browser-u.
- **ng version** – proverava u kojoj su verziji frameworks i jezici koje koristite u aplikaciji.

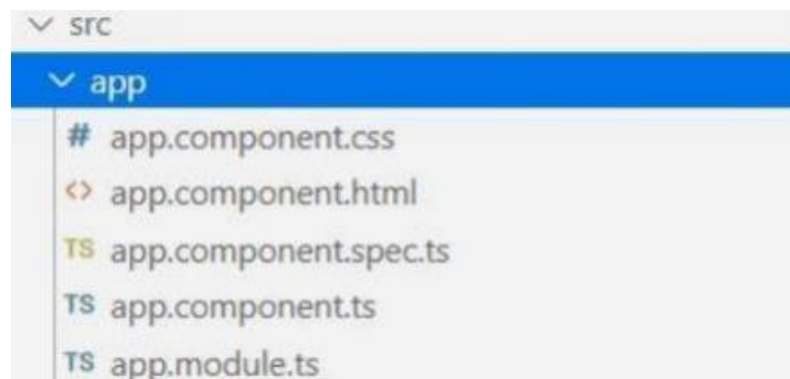
Struktura kreirane Angular aplikacije: Folder angular sadrži konfiguracione fajlove za testiranje aplikacije (unit test). Folder node_modules sadrži preuzete pakete. Folder src sadrži kod i ima 3 podfoldera:

- 1) App folder sadrži fajlove angular projekta: Komponente, sablone, pogled..
- 2) Assets folder sadrži staticke fajlove kao sto su npr. Slike
- 3) Environments folder sadrži fajlove vezane za okruženje, postoji razvojno I produkciono okruženje.

Fajl tsconfig.json sadrži typescript konfiguraciju Fajl package.json je JSON fajl koji sadrži informacije o potrebnim paketima u projektu Index.html predstavlja glavnu html stranu koja se učitava pri poseti sajta.

Fajl main.ts predstavlja glavnu ulaznu tačku aplikacije, tu se specificira koreni modul aplikacije koji se pokreće pri pokretanju aplikacije. Favicon.ico je ikona koja se prikazuje u tabu browsera Fajl angular.json sadrži konfiguracione opcije za kompajliranje i razmeštanje projekta.

Kada se kreira Angular aplikacija automatski se generiše glavni koren (root) modul aplikacije AppModule. Da bi neka klasa bila modul, mora da ima anotaciju @NgModule.



Izrada Aplikacije

Aplikacija Car Insurance je aplikacija koja se sastoji od tri dela:

- Baza
- Backend
- Frontend

Baza



Za kreiranje baze koristili smo MySQL, koji je najpopularniji sistem za upravljanje SQL bazama podataka otvorenog koda, razvija ga, distribuira i podržava Oracle Corporation. Modeliranje baze predstavlja prvi deo zadatka prilikom izrade Car Insurance aplikacije, koja je već postojala, ali je zahtevala neke izmene. Prvobitna baza je imala greške u nazivima entiteta, veze između entiteta su bile pogrešne, neki entiteti su bili nepotrebni, kao i atributi koji su se nalazili u njima.

Primeri nekih grešaka: Veza između Country i City je bila N:1, entitet Card je imao atribut maritalstatus itd... Proces popravljavanja šeme baze podataka smo započeli tako što smo prvo uočili koji entiteti nisu pravilno napisani, zatim smo proverili veze između entiteta i nakon toga obrisali entitete koji su višak i napravili logički funkcionalnu bazu podataka. Ispravljena baza podataka sadrži 20 entiteta: Accident, Authority, Card, Car, Country, City, Driver, Franchise, InsurancePlan, InsuranceItem, MaritalStatus, PaymentMode, Person, Policy, Proposal, Report, Risk, Role, Subscriber, User.

Pojašnjenje pojedinih entiteta:

CAR – predstavlja automobil koji se osigurava. Auto ima attribute: IdCar, color, brand, model, year.

INSURANCEITEM – predstavlja stavku koju obuhvata osiguranje (npr. krađa). Entitet sadrži attribute: idInsuranceItem, nameItem, amount.

FRANCHISE – predstavlja iznos koji klijent treba da plati od štete. Osiguravajuća kuća pokriva razliku između stvarne štete i franšize. Svaka stavka osiguranja ima samo jednu tačno franšizu.

INSURANCEPLAN – predstavlja paket osiguranja, u našoj bazi je tačno definisano 4 paketa koji imaju svoje stavke. Nazivi paketa su: Dopunski, Delimični, Potpuni i Premium Kasko paket.

PERSON – predstavlja entitet koji još nije kreirao svoju policy-u. Entitet ima attribute: IdPerson, firstName, lastName, jmbg, dateOfBirth.

SUBSCRIBER – predstavlja entitet koji je kreirao svoju policy-u.

MARITALSTATUS – predstavlja entitet koji služi kao nabranje mogućih bračnih statusa (npr. -> oženjen). Entitet ima attribute: idMaritalStatus i description.

ROLESUBSCRIBER – predstavlja koje uloge subscriber može da ima. Postoje dve uloge koje su definisane u bazi, a to su: Prospekt (za one subscibere koji su odbili ugovor na samom kraju kreiranja policy-e) i klijent (koji su potpisali ugovor sa kompanijom).

DRIVER – je entitet koji prilikom kreiranja osiguranja za automobil je registovan kao osoba koja upravlja tim automobilom. Entitet sadrži attribute koji ga detaljnije opisuju, a to su : IdDriver, LicenceObtain, licenceNumber, yearsInsured.

PROPOSAL – je entitet koji je najvažniji entitet baze podataka. Koji čuva sve podatke koji su potrebni za kreiranje osiguranja. Ima jedinstveni id, datum kreiranja osiguranja, registracione tablice, cenu osiguranja odabranog paketa i način plaćanja.

POLICY – predstavlja entitet koji se odnosi na ugovor klijenta, atributi entiteta su: idPolicy, cena paketa osiguranja, datum kupovine paketa osiguranja, krajnji rok plaćanja osiguranja i način plaćanja.

Veze između entiteta:

Subscriber – Person -> 1:1,

Subscriber – Role -> N:1,

Subscriber – Risk -> 1:N,

Subscriber – City -> N:1

Subscriber – Car -> 1:N,

Driver – Risk -> 1:N

Driver – Accident -> N:1

Driver – Person -> N:1

Driver – Proposal -> N:N

Person – MaritalStatus -> N:1

Country – City -> 1:N

Proposal – Policy -> 1:N

Policy – PaymentMode ->1:N

Policy – Report ->1:N

PaymentMode – Card ->1:1

Proposal – Car -> 1:N

Proposal – InsurancePlan ->N:1

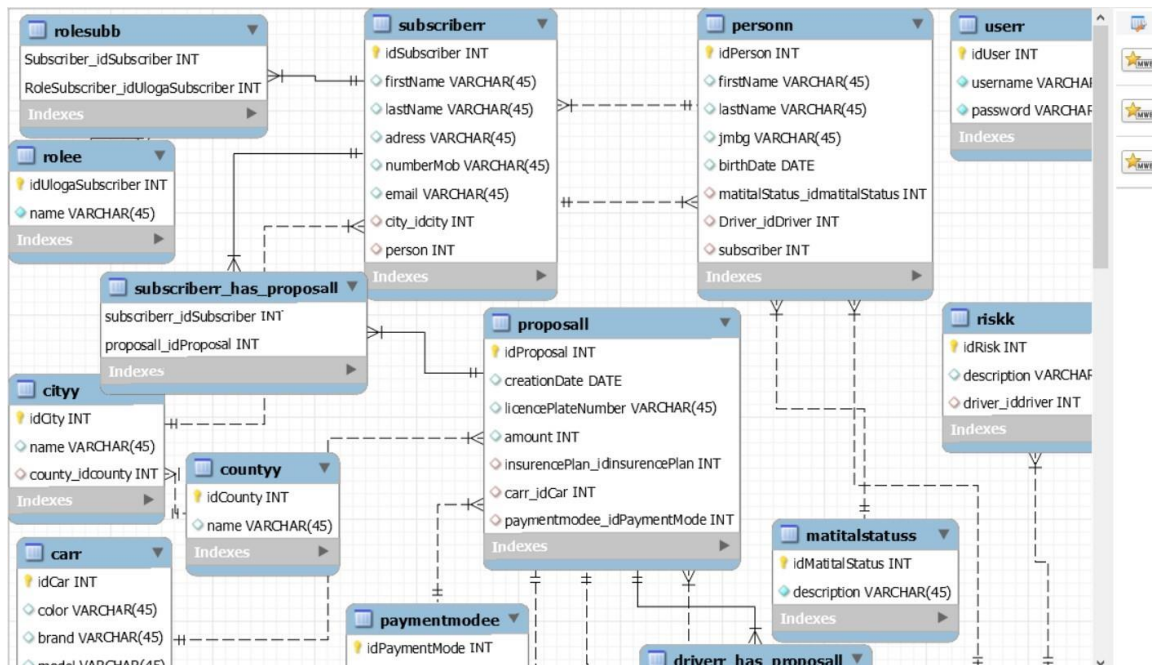
Proposal – Franchise -> N:N

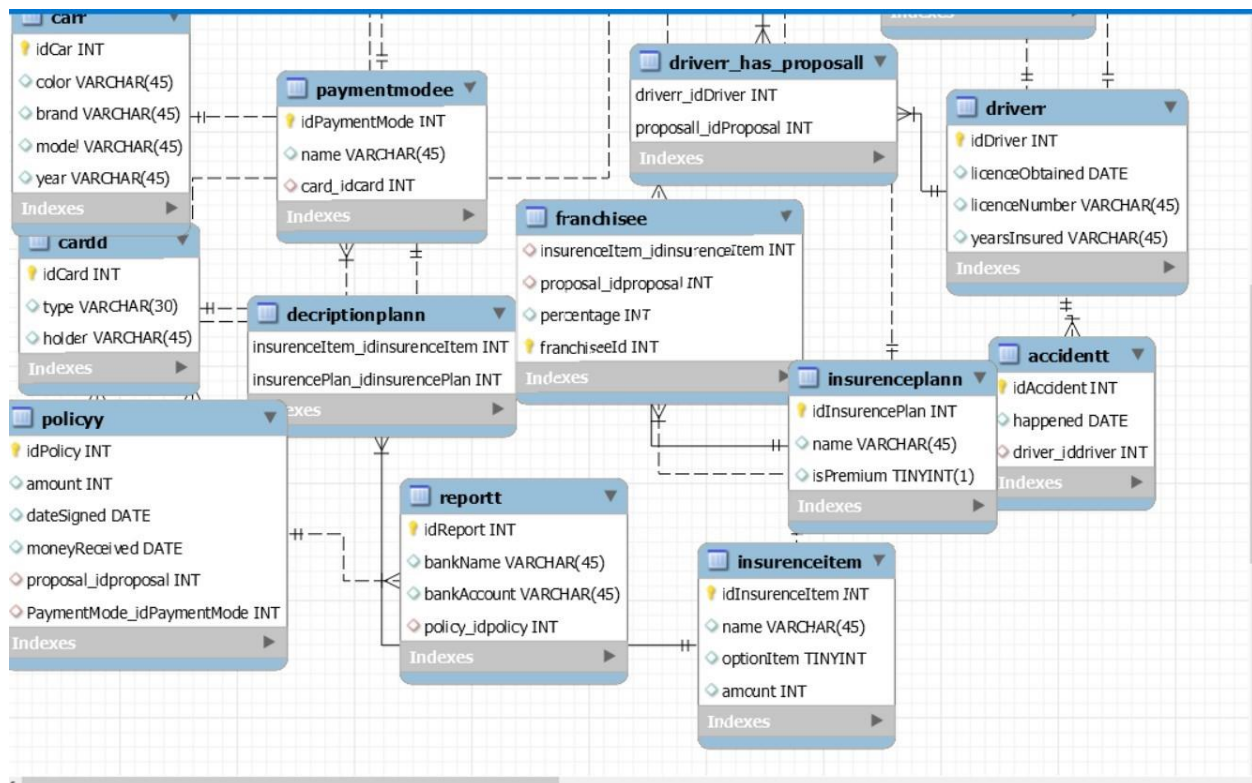
Franchise – InsuranceItem ->N:N

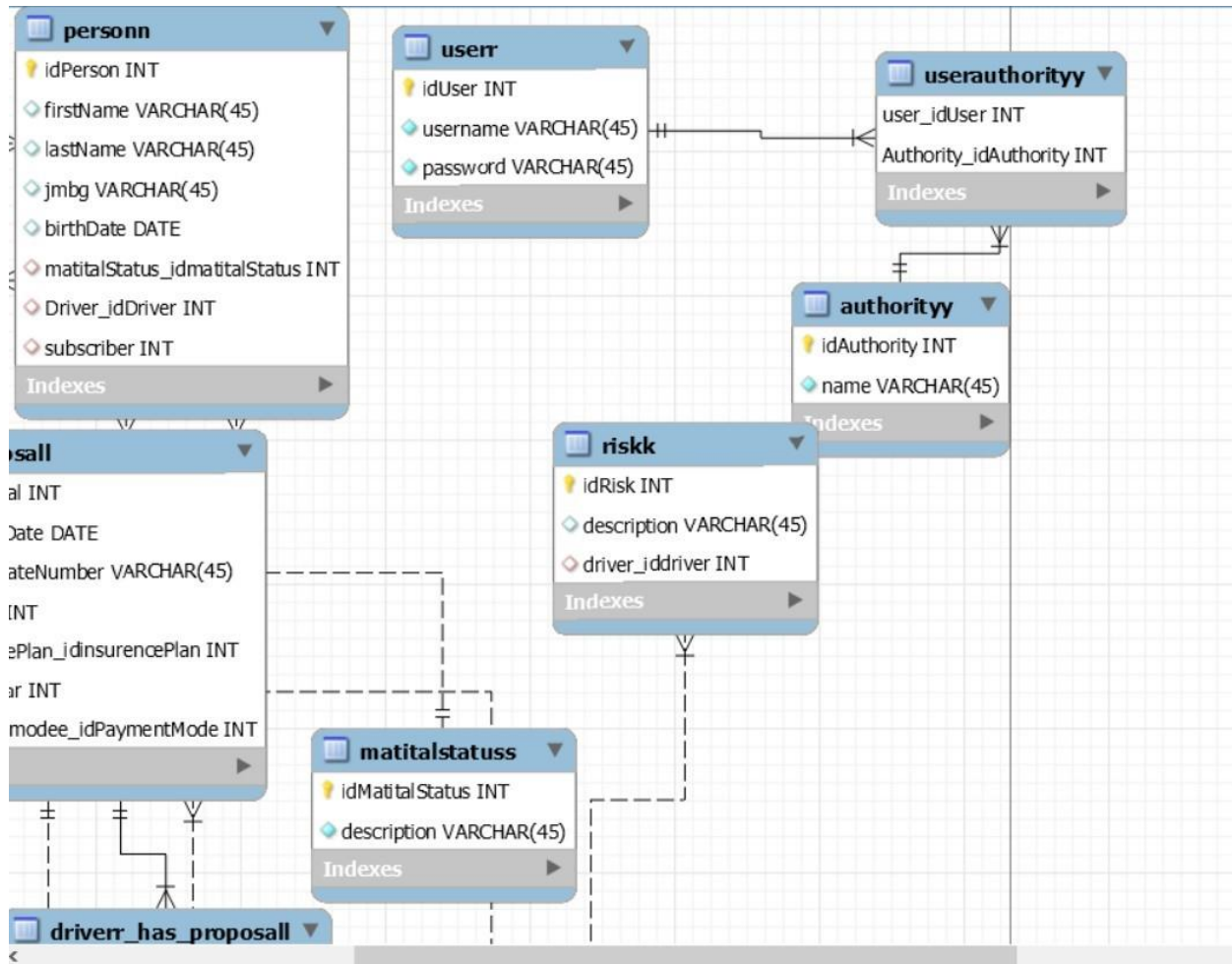
InsurancePlan - InsuranceItem ->N:N

User -UserAuthority ->1:N

Authority – UserAuthority -> 1:N



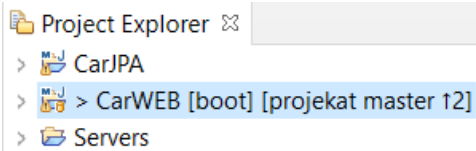




Prilikom izmene baze podataka i kreiranja logički funkcionalne baze podataka nismo imali problema, jer smo na fakultetu bili upoznati već sa MySQL-om i radom u njemu na predmetu Baze podataka.

Backend

Backend je deo Car Insurance aplikacije koju korisnici ne mogu da vide i nemaju direktan kontakt sa njom. Backend je kod koji se pokreće na serveru, koji prima zahteve od klijenta i sadrži logiku za slanje odgovarajućih podataka nazad klijentu. Serverska strana aplikacije je implementirana u Java jeziku u Eclipse okruženju, server koji je korišćen je Tomcat v9.



Mapiranje Java objekta u tabele baze podata I obrnuto naziva se objektno-relaciono mapiranje (ORM). Java persistence API (JPA) je jedan od mogućih pristupa ORM-u. Preko JPA možemo da mapiramo, skladištimo, I preuzimamo podatke iz relacionione baze podataka u Java objekat I obrnuto. Hibernate predstavlja implementaciju JPA specifikacije. CarJPA ima paket model koji sadrži sve entitete koji su generisani iz tabela baze podataka, da bismo imali mogućnost generisanja entiteta potrebno je projekat povezemo sa šemom baze podataka. CarJPA projekat koji ima paket model koji sadrži sve entitete koji su generisani iz tabela baze podataka.

```
@Entity
@NamedQuery(name="Subscriberr.findAll", query="SELECT s FROM Subscriberr s")
public class Subscriberr implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int idSubscriber;

    private String address;

    private String email;

    private String firstName;

    private String lastName;

    private String numberMob;

    //bi-directional many-to-one association to Personn
    @OneToMany(mappedBy="subscriberr", fetch = FetchType.LAZY)
    @JsonIgnore
    private List<Personn> personns;

    //bi-directional many-to-one association to Cityy
    @ManyToOne
    @JoinColumn(name="city_idcity")
    private Cityy cityy;

    //bi-directional many-to-one association to Personn
    @JsonBackReference
    @ManyToOne
    @JoinColumn(name="person")
    private Personn personn;
}
```

Klasa "Subscriberr" iz paketa "model" koja u sebi sadrži sve podatke i veze iz baze podataka vezane za subribera. Takođe sadrži I konstruktor sa potrebnim podacima , kao I get I set metode. Klasa Subscriberr je samo primer kako klasa u JPA modelu izgleda. Svaki entitet u bazi ima svoju klasu u ovom paketu.

```

    public Subscriberr(int idSubscriber, String address, String email, String firstName, String lastName,
        String numberMob, List<Personn> personns, Cityy cityy, Personn personn, List<Proposall> proposalls,
        List<Rolee> rolees) {
        super();
        this.idSubscriber = idSubscriber;
        this.address = address;
        this.email = email;
        this.firstName = firstName;
        this.lastName = lastName;
        this.numberMob = numberMob;
        this.personns = personns;
        this.cityy = cityy;
        this.personn = personn;
        this.proposalls = proposalls;
        this.rolees = rolees;
    }

    public int getIdSubscriber() {
        return this.idSubscriber;
    }

    public void setIdSubscriber(int idSubscriber) {
        this.idSubscriber = idSubscriber;
    }

    public String getAddress() {
        return this.address;
    }
}

```

CarWEB projekat omogućava da upravljamo infomacijama i kreiramo složene i napredne funkcije web stranice. Projekat CarWeb ima paket praksa koji u sebi ima podpakete: Repository, Service, Controller i Dto.

Repository

Repository je zadužen za perzistenciju podataka (dobavljanje i čuvanje podataka), obično definiše kao interfejs koji nasleđuje JpaRepository, svaki repozitorijum ima anotaciju @Repository. Jedan Repository je odgovoran za jedan entitet i operacije nad njim zbog toga dajemo naziv po tome na koji entitet se odnosi i dodajemo repository.

```

1 package com.praksa.repository;
2
3 import java.util.List;
4
5 public interface CarRepository extends JpaRepository<Carr, Integer> {
6     @Query("select c from Carr c where c.brand like :brand")
7     List<Carr> findByBrand(@Param("brand")String brand);
8
9     @Query("select c from Carr c where c.model like :model")
10    List<Carr> findByModel(@Param("model")String model);
11
12    @Query("select year from Carr where model like :model and brand like :brand")
13    List<Carr> findByMB(@Param("model")String model, @Param("brand")String brand);
14
15    @Query("select DISTINCT c.brand from Carr c")
16    List<String> AllBrand();
17
18    @Query("select distinct c.year from Carr c where c.brand = :brand and c.model = :model")
19    List<String> getYear(@Param("model")String model, @Param("brand")String brand);
20
21    @Query("select DISTINCT c.model from Carr c where c.brand like :brand")
22    List<String> findModelForBrand(@Param("brand")String brand);
23
24    List<Carr> findByBrandAndModelAndYear(String brand, String model, String year);
25 }
26

```

Kao što sam gore navela unutar interfejsa se nalaze operacije koje se odnose na taj određen entitet. Operacije mogu biti odrađene preko upita koji uz pomoć anotacije ‘Query’ pribavlja potrebne podatke na osnovu uslova koje zadajemo unutar upita. Takođe, postoji i operacije koje napišemo kao zaglavlje metode bez upita iznad i tu sam metod, na osnovu povratne vrednosti i ulaznih podataka, zna šta treba da nam vrati.

Service

Servis je komponenta iznad repozitorijuma koja koristi njegove usluge, kao i usluge drugih servisa. Servisi enkapsuliraju poslovnu logiku aplikacije i označeni su anotacijom `@Service`.

```
1 package com.praksa.service;
2
3 import java.util.ArrayList;
4
25 @Service
26 public class SubscriberService {
27     private static final int Subscriberr = 0;
28     @Autowired
29     private CityRepository crr;
30     @Autowired
31     private RoleRepository rr;
32     @Autowired
33     private PersonRepository pr;
34     @Autowired
35     private SubscriberRepository sr;
36
37     @Autowired
38     private MaritalStatusRepository msr;
39
40     @Autowired
41     private ProposalRepository propR;
42
```

Pomoću anotacije `@Autowired` omogućeno je korišćenje servisa i njegovih metoda.

```

80
81 // Update
82 List<Subscriberr> subsssss = new ArrayList<Subscriberr>();
83 subsssss.add(subb);
84 Proposall newProposal = new Proposall();
85 newProposal.setAmount(0);
86 newProposal.setCreationDate(new Date());
87 Proposall newProposall = propR.save(newProposal);
88 newProposall.setSubscriberrrs(subsssss);
89 System.out.print("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
90 System.out.print(subsssss);
91 propR.save(newProposall);
92 List<Proposall> props = new ArrayList<Proposall>();
93 props.add(newProposall);
94 subb.setProposalls(props);
95 sr.save(subb);
96 return newProposall;
97
98 }
99
100 // delete Subscriber
101 public void deleteSubscriber(Integer Id) {
102     sr.deleteById(Id);
103 }
104
105 //add subscriber
106 public Proposall saveSubscriber(SubscriberDTO subscriberDto) {
107     System.out.println(subscriberDto);
108     List<Personn> personList = pr.findByJmbg(subscriberDto.getJmbg());
109     Personn p = null;
110     if (personList.size() == 0) {
111         p = new Personn();
112         p.setBirthDate(subscriberDto.getDateOfBirth());
113         p.setFirstName(subscriberDto.getFirstName());
114         p.setLastName(subscriberDto.getLastName());
115         p.setJmbg(subscriberDto.getJmbg());
116         List<Matitalstatus> status = msr.findByDescription(subscriberDto.getMaritalStatus());
117         p.setMatitalstatus(status.get(0));
118
119         p = pr.save(p);
120     } else {
121         p = personList.get(0);
122     }
123
124     City con = crr.findByName(subscriberDto.city).get(0);
125     List<Rolee> role = new ArrayList();
126     role.add(rr.findById(2).get());
127     Subscriberr sub = new Subscriberr();
128     sub.setFirstName(subscriberDto.getFirstName());
129     sub.setLastName(subscriberDto.getLastName());
130     sub.setAdress(subscriberDto.getAddress());
131     sub.setEmail(subscriberDto.getEmail());
132     sub.setNumberMob(subscriberDto.getNumberMob());
133     sub.setRolees(role);
134     p.setSubscriberr(sub);
135     sub.setPersonn(p);
136     sub.setCityy(con);
137     System.out.print(sub);
138     Subscriberr subb = sr.save(sub);
139
140     System.out.print(sub);
141 }

```

Na slikama iznad se nalazi metod koji predstavlja čuvanje novog subscriber-a u bazu, a ujedno i sam id subscriber-a će biti sačuvan u proposal-u. Id tog proposal-a će se prosleđivati dalje na sledeće funkcionalnosti naše aplikacije i svaka sledeća funkcionalnost (jednog kola, tj. Jednog prolaska kroz tunel) će biti sačuvana u isti proposal.

Na slikama iznad se nalazi i metod za brisanje postojećeg subscriber-a iz baze na osnovu prosleđenog id-a.

```
152 public List<Subscriber> getSubscriberForNameAndJmbg(String jmbg, String firstName, String lastName) {
153
154     List<Subscriber> subs = sr.getSubscribersByFirstAndLastName(jmbg, firstName, lastName);
155
156     List<Subscriber> sub = new ArrayList<>();
157     System.out.println(subs.size());
158     for(Subscriber s : subs) {
159
160         sub.add(s);
161     }
162
163     return sub;
164
165 }
166
167
168 }
169
```

Metod na slici iznad radi pretreživanje subscriber-a po JMBG-u, Imenu I Prezimenu.


```

public Proposal paymetForCheck(PaymentModeDTO paymentModeDTO) {
    Proposal p= pr.getById(paymentModeDTO.getIdProp());
    Policy policy=new Policy();
    Paymentmode paymentMode= new Paymentmode();
    Reportt report=new Reportt();
    paymentMode.setName("Check");
    report.setBankName(paymentModeDTO.getRepDTO().getBankName());
    report.setBankAccount(paymentModeDTO.getRepDTO().getBankAccount());

    Reportt r = rp.save(report);
    List<Reportt>reports= new ArrayList<Reportt>();
    reports.add(r);
    int cena=paymentModeDTO.getPolicy().getAmount();
    policy.setReportts(reports);
    policy.setAmount(cena);
    policy.setDateSigned(paymentModeDTO.getPolicy().getDateSigned());
    p.setAmount(cena);
    p.setCreationDate(new Date());

    Paymentmode pMode = pmr.save(paymentMode);
    p.setPaymentmode(pMode);

    pr.save(p);
    policy.setProposal(p);
    polR.save(policy);

    return p;
}

```

Na slici iznad se nalazi metod koji predstavlja metod preko kojeg se vrši plaćanje kešom. Potrebni podaci se unose i setuju u bazi na odgovarajućim mestima. Samo plaćanje se setuje u Proposal-u.

Controller

Kontroler ima anotaciju @Controller i njegova uloga je da prima poruku od View komponente, da ih prosleđuje ka servisu i obrnuto (da odgovor servisa prosledi View komponentama).

```
CityController.java
1 package com.praksa.controller;
2
3 import java.util.List;
4
5 @RestController
6 //("/city")
7 @RequestMapping("/CityController")
8 @CrossOrigin("**")
9 public class CityController {
10     @Autowired
11     private CountryService countryService;
12
13     @Autowired
14     private CityService cityService;
15
16     //save
17     @RequestMapping(value = "/saveCity", method = RequestMethod.POST)
18     public ResponseEntity<City> saveCity(@RequestBody CityDTO cityDTO, HttpServletRequest request) {
19
20         City saveCity = cityService.saveCity(cityDTO);
21
22         if (saveCity != null) {
23             return new ResponseEntity<City>(saveCity, HttpStatus.CREATED);
24         }
25
26         return new ResponseEntity<City>(HttpStatus.INTERNAL_SERVER_ERROR);
27     }
28
29     //update
30     @PutMapping("/{cityUpdate/{id}}")
31     public ResponseEntity<City> updateCity(@PathVariable("id") Integer id,
32     @RequestBody CityDTO cityDTO) {
33
34         return new ResponseEntity<City>(cityService.updateCity(id, cityDTO), HttpStatus.OK);
35     }
36
37 }
```

```
CityController.java
38 // delete
39 @DeleteMapping("/{cityDelete/{id}}")
40 public ResponseEntity<City> deleteCity(@PathVariable("id") Integer Id) {
41     cityService.deleteCity(Id);
42     return new ResponseEntity<>(HttpStatus.NO_CONTENT);
43 }
44
45 //read
46 @GetMapping("/{readCity/{id}}")
47 public ResponseEntity<City> readCity(@PathVariable Integer id) {
48     return new ResponseEntity<>(cityService.readCity(id), HttpStatus.OK);
49 }
50
51 //get all city
52 @GetMapping("/{getAllCity}")
53 public ResponseEntity<List<String>> getAllCity() {
54     try {
55         List<String> modelCity = cityService.getAllCity();
56         return new ResponseEntity<>(modelCity, HttpStatus.OK);
57     } catch (Exception e) {
58         return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
59     }
60 }
61
62 @GetMapping("/{getAllCityOfCountry/{country}}")
63 public ResponseEntity<List<String>> getAllBrand(@PathVariable("country") String country) {
64     try {
65         List<String> city = cityService.getAllCityOfCountry(country);
66         return new ResponseEntity<>(city, HttpStatus.OK);
67     } catch (Exception e) {
68         return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
69     }
70 }
71
72 }
```

Unutar klasa koje su kontroleri aplikacije se nalaze @Autowired anotacije uz pomoć kojih pristupamo funkcijama koje se nalaze unutar Service klasa. Te funkcije obrađujemo unutar Controller klasa. Controller klase služe za komunikaciju slanja zahteva korisnika servis klasama i odgovor korisnicima.

DTO

Objekat prenosa podataka je objekat koji se koristi za enkapsulaciju podataka i njihovo slanje iz jednog podsistema aplikacije u drugi. Dto se koristi u projektu za prenos podataka između backend-a i korisničkog interfejsa. Glavna prednost DTO je što smanjuje količinu podataka koje treba da se šalju, što znači da imamo mogućnost da smanjimo broj parametara iz Modela kao i da promenimo tip podataka parametra. DTO smo formulisali samo sa potrebnim podacima koji se koriste na front-u.

```
1 package com.praksa.dto;
2
3 import java.util.Date;
11
12 public class SubscriberDTO {
13
14     public int idSubscriber;
15     public String address;
16     public String email;
17     public String firstName;
18     public String lastName;
19     public Date dateOfBirth;
20     public String numberMob;
21     public String city;
22     public String jmbg;
23     public List<Rolee> rolees;
24     public List<Proposall> preposalls;
25     public int person;
26     public String maritalStatus;
27
28     public String getAddress() {
29         return address;
30     }
31
32     public void setAddress(String address) {
33         this.address = address;
34     }
35
36     public String getEmail() {
37         return email;
38     }
39
40     public void setEmail(String email) {
41         this.email = email;
42     }
43 }
```

Klasa DTO ima potrebne podatke koje su potrebne za frontend kao i get i set metode tih podataka.

Front

Frontend je deo Car Insurance aplikacije sa kojom korisnik direktno stupa u interakciju, naziva se još i klijentska strana aplikacije. Angular je razvojno okruženje koje se koristi za kreiranje klijentskog dela web aplikacije, korišćenjem HTML-a i CSS-a i TypeScripta.

```

import { MaritalStatus } from "../maritalStatus";
import { Person } from "../person";
import { Role } from "../role";

export class Subscriber{
  constructor(
    public maritalStatus:String,
    public firstName:String,
    public lastName:String,
    public dateOfBirth:Date,
    public jmbg:String,
    public country:String,
    public cityy:String,
    public address:String,
    public numberMob:String,
    public email:String,
    //public rolees:String,
    public personn:Person,
    public idSubscriber:number
  ){
  }
}

```

Prikaz kako model izgleda u Angular-u. Klasa model ima sve potrebne podatke za dati entitet iz baze koje su potrebne za front stranu.

```

public add(sub: Subscriber): Observable<Proposal> {
  return this.http.post<Proposal>(`${this.subURL}/proposalController/saveProposalJ`, sub
  );
}

```

Metod add služi za čuvanje Subscriber-a u proposal-u. Unutar ovog metoda se poziva metod koji se nalazi na beck-u, a samu lokaciju potrebnog metoda smo dali uz pomoć potrebnog URL-a.

```

found: boolean = false;

returnedSubscriber: Subscriber = new Subscriber('', '', '', new Date(new Date()), '', '', '', '', '', '', this.temp_p
getSubscriber() {
  alert('usao')
  this.service.getByNamesAndJMBG(this.jmbg, this.firstName, this.lastName).subscribe((result) => {

    console.log(result)
    if (result == null) {
      alert("User not found.Please try again")
      this.found = false
    } else {
      console.log(result[0]);

      this.returnedSubscriber = result[0];
      console.log(this.returnedSubscriber);

      this.found = true;
    }
  });
}
}

```

Tretraga subscriber-a po jmbg, imenu i prezimenu.

```

goToCarPage() {
  if (!this.added) {
    //sacuvaj proposal sa ovim subscriberom i prosledi se na car

    this.propService.add({
      maritalStatus: this.sms,
      firstName: this.firstName,
      lastName: this.lastName,
      dateOfBirth: this.dateOfBirth,
      jmbg: this.jmbg,
      country: this.selectedCountry,
      city: this.selectedCity,
      address: this.address,
      numberMob: this.numberMob,
      email: this.email,
      personn: this.person,
      idSubscriber: this.subsciriber_id
    }).subscribe((result) => {

      this.retProp = result
      console.log(this.retProp)
      alert(this.retProp.idProposal)
      localStorage.setItem('id-proposala', "" + this.retProp.idProposal)
      console.log(localStorage.getItem("id-proposala"))
      this.added = true;
      this.router.navigate(['car'])
    })
  } else {
    this.router.navigate(['car'])
  }
}
}

```

Prestavljen prelazak na sledeću stranicu, tj. Funkcionalnost. Na sledeću stranicu se prosleđuje i idProposala na koji je prethodno sačuvan subscriber, a biće i svaki sledeći podakatak sa narednih funkcionalnosti.

Proposal predstavlja registraciju, tako npr. i ako registracija nije do kraja završena, jer se korisnik predomislio ili nije zadovoljan sa ponudom osiguranja za svoj automobil svi podaci koji su uneti do tada su sačuvani.

Za sam prelazak na drugu stranicu morali smo da u Angularu uvedemo Router koji omogućava navigaciju od jedne stranice ka drugoj. App.routing.module koji sadrži sve rute koje se koriste u aplikaciji, svaka ruta ima svoju: Path-String koji odgovara URL adresi u traci za adresu pregledaca i component- koju ruter treba kreira prilikom navigacije do ove rute (`this.router.navigate(['car'])`).



Prilikom pokretanja aplikacije svi klijenti mogu da vide HomePage koja ima dva dugmeta: Registration i Description. Klikom na Registration dugme dobijate mogućnost registracije i mogućnost da kao registrovani korisnik kreirate svoju polisu osiguranja za svoje vozilo. Klikom na dugme Description dobićete poruku kojoj je detaljno opisano koja je funkcija sajta i koje su prednosti korišćenja našeg osiguranja.

Username:

Password:

Login in



Izborom registracije dobijamo mogućnost unosa username i password-a postajemo registrovani korisnik. Ovde uočavamo da imamo grešku prilikom kreiranja aplikacije jer dolazi do zbrke između registracije i logovanja.

First Name

Last Name

Phone

Country

City

Market status

Address

Number Mobile

Email

Add Search Next

Registrovani korisnik ima mogućnost kreiranja polise za svoje vozilo i pretraživanje već postojeće polise. Unosom traženih podataka i klikom na dugme biramo funkcionalnost koju želimo da odradimo. Mogućnost kreiranja polise, dodavanje Subscriber-a (u slučaju da se ne unesu svi potrebni podaci, ili ne popune sva polja korisnik će biti obavešten o tome). Subscriber je osoba koja želi da kreira polisu.

Za izbor podataka Country, City, MaritalStatus koristili smo combobox koji je sadrži podatke iz baze npr. sve zemlje koje postoje, da bismo imali pristup podacima potrebno je bilo da na backu napišemo upit u repozitorijumu, u servisu implementiramo metodu, pozovemo u controlleru i zatim u servisu Angulara pozovemo metodu iz controllera. Sve te korake smo prešli gore uz pomoć određenih fotografija.

Klikom na Next prebalazi na sledeću stranicu aplikacije na kojoj se nastavlja kreiranja polise. Sada birate polja koja su obeležja vašeg automobila (model, brand, godina proizvodnje i registracija vozila). Izborom save dugmeta čuvaju se izabrani podaci, zatim kada završite taj deo kliknete na Next koji omogućava da nastavite kreiranje polise.

Nakon što smo završili unos podataka vezanih za automobil I prelaskom na sledeću stranicu otvara se nova forma na kojoj korisnik treba da odluči koliko vozača ce imati mogućnost korišćenja automobila (1-4 vozača) za automobil na koji se kreira osiguranje.

Choose how many drivers you add:


1


FirstName


LastName

JMBG

Licence Number

Licence Obtained 

risk: 

Driver happened 

Years

Add driver Search Next

Kada izaberete koliko vozača ima mogućnost upravljanja automobilom za koji kreirate registraciju potrebno je da se njihovi licni podaci unesu. Pored osnovih podataka potrebno je da se izabere da li je vozač imao neki rizik. U slučaju da je imao saobraćajnu nezgodu potrebno je uneti datum kada se ona dogodila.

```
public save() {  
  let selectedRisks = [];  
  selectedRisks.push(this.selectedRisk);  
  console.log(this.person)  
  console.log(this.person2)  
  console.log(this.person3)  
  console.log(this.person4)  
  this.drivers.push({  
    licenceObtained: this.licenceObtained,  
    yearsInsured: this.yearsInsured,  
    licenceNumber: this.licenceNumber,  
    personDTO: this.person,  
    happened: this.happened,  
    risk: selectedRisks,  
    idDriver: this.idDriver,  
    idProposal: parseInt(""+localStorage.getItem('id-proposala')),  
  });  
}
```

Dodavanje vozača se razlikuje od dodavanja automobila i subscribera gde se osiguranje odnosi samo na jedno vozilo i jednog Subscrbera, dok polisa moze da ima više vozača...Da bi smo uspešno dodali 1-4 vozača koristimo listu.

Pored unosa podataka o vozačima postoji mogućnost pretrage vozača po broju vozačke dozvole koji je jedinstven za svakog vozača.

Prelaskom na sledeću stranicu dobijamo tabelu na kojoj su nam ponuđeni razni paketi. Cena svakog paketa je različita. Cena poslednjeg paketa zavisi od mogućnosti koje izaberemo za dati paket.

Dopunski Kasko	Delimicni Kasko	Potpuni Kasko	Premium Kasko
Kradje	Saobracajne nesrece Kradje Fleksibilnost placanja	Saobracajne nesrece Kradje Pozare Poplave Elementarne nepogode Pada drveca fasada i drugih predmeta Eksplozije Lom i ostecenja stakala	Saobracajne nesrece Kradje <input type="checkbox"/> Pozare <input type="checkbox"/> Poplave <input type="checkbox"/> Elementarne nepogode <input type="checkbox"/> Pada drveca fasada i drugih predmeta <input type="checkbox"/> Eksplozije <input type="checkbox"/> Lom i ostecenja stakala <input type="checkbox"/> Cukanje na parkingu <input type="checkbox"/> Fleksibilnost placanja
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

save

Nakon odabira paketa prelazimo na sledeću stranicu na kojoj se vrši plaćanje. Ponuđena su tri načina plaćanje: keš, kartica ili ček. Odabirom jednog od tri načina dobijamo prozor sa podacima koje trebamo popuniti.

Method of payment: Check

Bank name: Komercijalna

Amount: 28990

Date Signed: 12.02.2022.

Creation Date: 11.01.2022.

PAY

```

this.propsService.paymentForCash({idPaymentMode: this.idPaymentMode,
  name: this.selectedName,
  idProp:parseInt("" + localStorage.getItem('id-proposala')),
  carDTO:({idCard:this.idCard,type:this.selectedType,holder:this.holder}),
  repDTO:({idReport:this.idReport,bankAccount:this.bankAccount,
    bankName:this.bankName,idPolicy:this.idPolicy}),
  policyy:({idPolicy:this.idPolicy,amount:this.amount,dateSigned:this.dateSigned,
    moneyReceived:this.moneyReceived,paymentmodee:parseInt("" + localStorage.getItem('id-PaymentModea')),
    proposall:parseInt("" + localStorage.getItem('id-proposala')),reportts:[{}}]}))
}).subscribe((res)=>{
  this.retProp=res;
  console.log(res);
});
}

```

Kada ste kliknuli na dugme PAY dobijate poruku koja sadrži sve podatke koje ste uneli za kreiranje polise : ime subscibera, auto, vozac, izabran paket osiguranja i koje stavke sadrži, provera plaćanja. Ako su svi podaci koji su u poruci ispravni kliknete ok -> time ste završili svoju polisnu.

Zaključak

Tehnologije koje su korišćenje za izgradnju ovog projekta su predstavljene na samom početku seminarskog rada u kratkim crtama. Predstavljena je firma za koju je rađen ovaj projekat, način na koji se može doći do same prakse kao i način na koji sam ja došla do prakse. Prikazana je baza, backend I frontend projekta kroz slike I kratka objašnjenja slika I pojedinih koraka ili ideja.

Projekat "Car Insurance" predstavlja aplikaciju za kreiranje polise i proces same registracije za auto sa sigurnošću čuvanje privatnih podataka. U budućnosti aplikacija može da se nadogradi, da se dodaju nove funkcionalnosti koje su od velike koristi. Neki od njih je dostava računa na email, da posle isteka polise, ta polisa se briše iz baze, da se korisnici putem mejla obaveste o raznim popustima I pogodnostima. To su neke od mogućih budućih funkcionalnosti ovog projekta, ali ne I jedini.

Resursi

<http://index-of.es/Java/Maven-The-Complete-Reference.pdf>

<https://infoteh.etf.ues.rs.ba/zbornik/2012/radovi/STS/STS-33.pdf>

http://www.racunarstvo.matf.bg.ac.rs/MasterRadovi/2016_06_14_Ljiljana_Lazic/rad.pdf

<https://www.rt-rk.uns.ac.rs/sites/default/files/materijali/predavanja/P3.3.pdf>

<https://www.mcloud.rs/blog/git-komande-za-pocetnike/>

<https://www.bpa.edu.rs/FileDownload?filename=21eb3f9d-2c65-474b-8735-0d74cc9578cb.pdf&originalName=02%20-%20Uvod%20u%20Angular.pdf>