Java web aplikacija za unapređenje organizacije poslovanja u Hrvatskoj vojsci

Kos, Marin

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

Java web aplikacija za unapređenje organizacije poslovanja u Hrvatskoj vojsci

Marin Kos

Zagreb, veljača 2019.

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada".

U Zagrebu, 26.02.2019.

Marin Kos

Predgovor

Zahvaljujem se mentoru Aleksanderu Radovanu na pomoći i savjetima tijekom izrade ovog završnog rada.

Zahvaljujem svojim prijateljima koji su mi bili motivacija, kolegama koje sam upoznao na Visokom učilištu Algebra te najviše ženi i sinu koji su mi bili neiscrpan izvor energije i najveća potpora.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

Ovim završnim radom i izradom web aplikacije za poboljšanje poslovanja u MORH-u obuhvatio se cijeli niz najmodernijih tehnologija. Počevši od Spring Boot programskog okvira uz opis i korištenje modula kao što su Spring DATA, Spring Security. Pristup podacima iz baze podataka pokrio se s JPA i HIBERNATE, a za skladištenje podataka je korišten MySQL. Za prezentacijski dio koristile su se biblioteke React i Redux. Arhitektura se bazirala na REST pozivima, tako da je razvoj bio podijeljen u dva zasebna projekta: u IntelliJ-u se razvijao *backend* dio, dok se u Visual Studio Code razvijao *frontend* dio.

Cilj je ovog završnog rada izrada prototipa softvera koji će olakšati i ubrzati poslovanje u Hrvatskoj vojsci. Isto tako, želi se detaljnije opisati Spring Boot programski okvir i njegovi moduli, kao i React i Redux biblioteke.

Ključne riječi: Spring Boot, Spring Data, Spring Security, Hibernate, React, Redux.

Summary

This final work and the development of a web application for improvement of business in the MORH included a number of cutting-edge technologies. Starting with the Spring Boot programming framework with the description and use of modules such as Spring Data, Spring Security. Database access was covered with JPA i HIBERNATE, and MySQL was used for data storage. For the presentation part the libraries React and Redux were used. Architecture was based on REST calls, so development was divided into two separate projects: a backend part was developed in IntelliJ, while Visual Studio Code developed a frontend part.

The aim of this final work is to create a protoype of software that will facilitate and speed up operations in the Croatian Army. It also wants to describe the Spring Boot framework and its modules as well as the React and Redux libraries.

Key words: Spring Boot, Spring Data, Spring Security, Hibernate, React, Redux.

Sadržaj

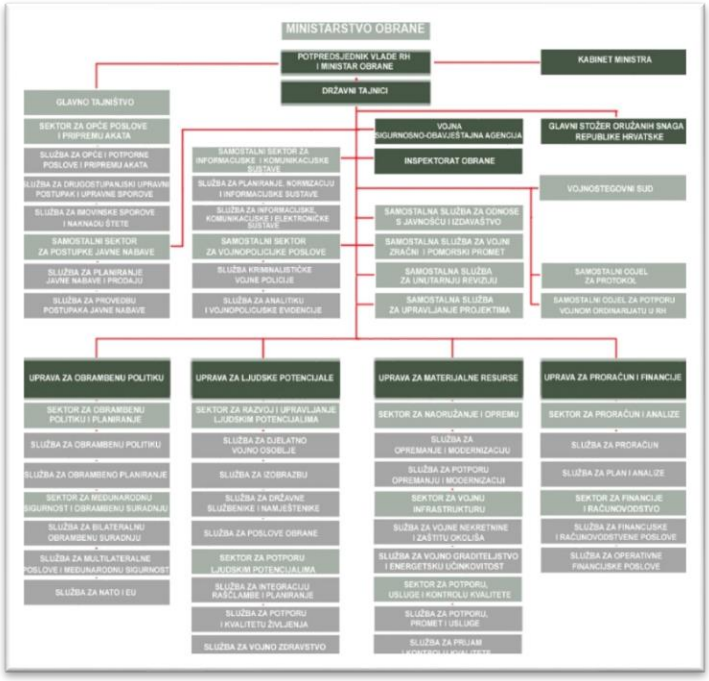
1. Uvod

Problem zbog kojeg je nastala ideja za pisanje ovog završnog rada zastarjelo je poslovanje unutar jedne postrojbe u Hrvatskoj vojsci, koji je raširen i na ostale postrojbe. Poslovanje se vodi bez ikakvog skladištenja podataka pa su glavni alati koji se koriste prilikom planiranja i vođenja papir i olovka uz pokoji Word dokument. Nakon uvoda u problematiku koja se opisuje u drugom poglavlju, u trećem poglavlju se opisuju sve tehnologije koje su se koristile prilikom izrade web aplikacije za unapređenje poslovanja u Hrvatskoj vojsci. Spring Boot programski okvir omogućuje brzo i jednostavno kreiranje projekata pa je zbog toga čest izbor kod izrade web aplikacija u Java programskom jeziku, pošto sadrži mnoštvo početnih modula koji se detaljno opisuju u trećem poglavlju. Uz Spring programski okvir o kojemu je riječ u trećem poglavlju, obradit će se: REST, ORM, Spring Security i način na koji je moguće odvojiti aplikacijski sloj od prezentacijskog. U četvrtom poglavlju se obrađuje klijentski dio aplikacije razvijan pomoću React i Redux biblioteka, tako da će se detaljno opisati navedene biblioteke i proći koraci potrebni za dohvat i prikaz podataka.

U petom poglavlju je opisana izrada aplikacije, obrađeni su podatkovni, poslovni i prezentacijski slojevi. Praktični dio završnog rada opisan u šestom poglavlju zamišljen je na način da zapovjednici u postrojbama imaju uvid u popis zaposlenika i mogućnost kreiranja raznih planova i zapovjedi koje će moći skladištiti po stupnju važnosti. Ispod zapovjedništva su satnije koje vode zaposlenike tako da će i oni imati uvid u raspoloživost i moći će određivati gdje je i u kojem razdoblju netko raspoređen.

2. Ustroj i poslovanje MORH-a

Kada se pogleda shematski prikaz ustroja Ministarstava obrane RH, vidi se kako se sastoji od puno ozbiljnih uprava, službi i sektora. Zbog nedovoljno informacija o poslovanju tih službi, više će biti riječi o poznatim činjenicama, a to je vojska RH, bez koje ne bi bilo svih tih službi. Grane oružanih snaga su: ratna mornarica, ratno zrakoplovstvo i kopnena vojska. Ustroj u kopnenoj vojsci kreće od Zapovjedništva HKoV-a, ispod kojeg su razne pukovnije i brigade, a u to spadaju pješaštvo, oklopništvo, topništvo, inženjerija i veza. Svaka od tih brigada i pukovnija u nadležnosti ispod sebe ima bojne koje se sastoje od satnija, satnija se sastoji od vodova, a vod od desetina. Brigade, pukovnije i bojne mogu se gledati kao zasebne tvrtke koje vode svoje poslovanje, imaju zaposlenike o kojima treba voditi brigu, bave se raznim zadaćama i svaka od tih zadaća je zaseban poslovni proces. Svaka postrojba ima svog zapovjednika čiji je posao provedba dobivenih zapovjedi od nadležnih te izrada internih zapovjedi i planiranje zadaća za pripadajuću postrojbu. Na slici (Slika 1) prikazan je kompletan ustroj Ministarstva obrane.



Slika 1. Ustroj Ministarstva Obrane [1]

[1] https://www.morh.hr/images/stories/_morh_2017/images/sheme/shema_morh_2017-01-23-v.jpg; 28.12.2018.

2.1. Problematika poslovnih procesa

Pionirska bojna, kao i ostale postrojbe u MORH-u, funkcionira tako da se jako malo poslovnih procesa odrađuje putem aplikacija. Počevši od nadzora zaposlenika i kontrole informacija, skladištenja podataka o zaposlenicima, praćenja iskorištenosti plaćenih dopusta i godišnjih odmora pa sve do izrade zapovjedi i bitnih dokumenata koji nestaju u hrpi papira. Razumljivo je da se ne može u potpunosti izbaciti poslovanje koje traje i funkcionira skoro trideset godina i da se papir ne može u cijelosti zamijeniti digitalnim izvorima. Takvo vođenje dovodi do problema, pošto toliku količinu informacija i dokumenata jedna osoba jednostavno ne može pratiti. Trenutno je poslovanje takvo da jedna do dvije osobe vode tekuće poslovne procese cijele bojne. Upoznati su sa svim poslovima koje moraju obavljati te znaju gdje se nalaze određeni Microsoft Word ili Excel dokumenti prilikom manipulacije podacima. Problemi nastaju kada se te osobe angažiraju na drugim zadaćama van postrojbe. Jedan je od najvećih problema u takvom poslovanju otežan i usporen pristup važnim informacijama, jer ne postoji jedno mjesto za skladištenje podataka.

2.2. Modernizacija poslovanja

Prototipom softvera, koji se u ovom radu opisuje i koji će se ustupiti Pionirskoj bojnoj na korištenje, želi se ubrzati i modernizirati svakodnevne poslovne procese. Zapovjednik će imati uvid u sve informacije o zaposlenicima. Izrada zapovjedi i dokumenata bit će jednostavnija te će dokumente odmah moći klasificirati po stupnju važnosti. Satnija će imati mogućnost dodavanja novih zaposlenika i mijenjanja informacija o istima. Također će za određenu osobu moći odrediti na kakvoj se zadaći nalazi i u kojem razdoblju. Sve će biti obuhvaćeno na jednom mjestu pa će poslovanje, manipulacija zaposlenicima i dokumentima biti značajno ubrzana i olakšana.

3. Uvod u Spring programski okvir

Spring programski okvir nastao je zahvaljujući velikoj kompleksnosti i kompliciranosti izrade aplikacija prije više od 15 godina. Korištenjem tadašnjih tehnologija, programeri su gubili puno vremena na pisanje konfiguracijskog koda, čak i za jednostavne aplikacije. Spring programski okvir danas je najpopularniji programski okvir za razvijanje Java aplikacija. Njegova srž leži u tome da je on kontejner za rješavanje *dependency injection-a*, koji omogućuje fleksibilnost u konfiguraciji komponenti (XML, *Annotations* i *JavaConfig*). Iz godine u godinu Spring je nudio sve više opcija i kvalitetnih rješavanja problema poslovnih aplikacija, kao što su sigurnost, podrška za *NoSQL* baze podataka, rukovanje velikim podacima, jednostavna integracija s drugim sustavima itd. Također, Spring programski okvir toliko je popularan zbog nekoliko razloga [2]:

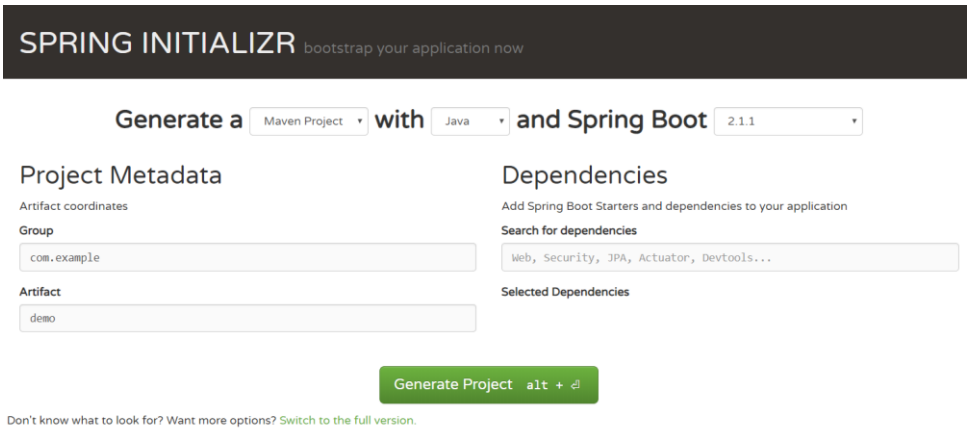
- *Dependency injection* potiče pisanje dobrog koda kojeg je lagano testirati.
- Jednostavno upravljanje transakcijama u bazi podataka.
- Jednostavna integracija s drugim Java programskim okvirima kao što su JPA i HIBERNATE ORM.
- Vrhunski web MVC programski okvir za izradu web aplikacija.

S obzirom da je Spring programski okvir vrlo fleksibilan u smislu da postoji više opcija konfiguriranja aplikacija, vrlo se lako može "zapetljati" prilikom postavljanja. Tim stručnjaka je sukladno takvim problemima stvorio Spring Boot programski okvir kako bi riješio problem konfiguracije i omogućio jednostavno i brzo postavljanje aplikacije u pogon.

3.1. Spring Boot programski okvir

Spring Boot jedan je od modula Spring programskog okvira, ali je specifičan po tome što sadrži sve ostale module. Ujedno automatski brzo i lagano konfigurira aplikaciju i instalira dodatne module koji su potrebni programeru za izradu aplikacije, uz mogućnost naknadnog dodavanja ili mijenjanja instaliranih modula u "`pom.xml`" datoteci. Generiranje Spring Boot projekata moguće je izvesti preko Spring Initializr-a, na web stranici *https://start.spring.io/*, a primjer kako to izgleda može se vidjeti na slici (Slika 2).

[2] Beginning Spring Boot 2; K. Siva Prasad Reddy, str. 1



Slika 2. Spring Initializr

Koraci generiranja Spring Boot projekta su sljedeći:

1. Odabere se *Maven project* i Spring Boot verzija (zadnja verzija prilikom pisanja ovog rada je 2.2.0 SNAPSHOT).
2. Pod *Project Metadata* upisuju se željene informacije o nazivu projekta.
3. Pod *Dependencies* upisuju se željeni moduli koji će se koristiti u projektu, a popis svih ostalih mogućnosti se vidi kada se prebaci na punu verziju generiranja projekta u dnu stranice.
4. Na kraju se klikne *Generate Project* nakon čega započinje skidanje ZIP datoteke na računalo koje se naknadno ručno otvora u razvojnom alatu (IntelliJ IDEA).

Druga opcija kreiranja projekta je korištenjem razvojnih alata (IntelliJ IDEA, Eclipse…). U IntelliJ razvojnom alatu postupak kreiranja projekta isti je kao kod kreiranja bilo kojeg drugog projekta, samo se u ovom slučaju odabere Spring Initialzr, gdje se slijedi korak za korakom do izbora `dependenciesa`, nakon čega se upiše ime projekta i pritiskom na *finish* kreira se projekt s gotovom strukturom i potrebnim paketima. Nakon uspješnog kreiranja projekta generiraju se tri bitne datoteke: "`pom.xml`", "`application.properties`" i glavna konfiguracijska klasa "`SpringBootApplication`" prikazana na programskom isječku (Programski isječak 1) koja pokreće aplikaciju:

```
@SpringBootApplication
public class CaosAppApplication {
    public static void main(String[] args) {
```

```
        SpringApplication.run(CaosAppApplication.class,
args);}}
```

Programski isječak 1. Glavna početna klasa u Spring Boot-u

Konfiguracijska datoteka "`application.properties`" predstavlja kontejner za konfiguracijske postavke poput definiranja konekcijskog `String-a`, korisničkog imena i šifre za pristup bazi podataka, ili mijenjanju tvornički postavljenog porta na kojem se Tomcat server pokreće. To se postavi s ubacivanjem `server.port=8082` u "`application.properties`" datoteku. Datoteka "`pom.xml`"služi kao *dependency resolver*. U početnim verzijama Spring programskog okvira ručno se tražio *dependency* putem Maven repozitorija na stranici https://mvnrepository.com/. Kada se našla odgovarajuća verzija potrebnog modula, ista se kopirala i ubacila u "`pom.xml`" datoteku koja je bila odgovorna za skidanje potrebnih datoteka. Danas zahvaljujući Spring Boot programskom okviru i jednostavnijem kreiranju projekata odabirom odgovarajućih modula za kreiranje web aplikacije dobije se primjer "`pom.xml`" datoteke koji se vidi na programskom isječku (Programski isječak 2):

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
              <artifactId>spring-boot-starter-data-
  jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

Programski isječak 2. Prikaz dijela "`pom.xml`" datoteke

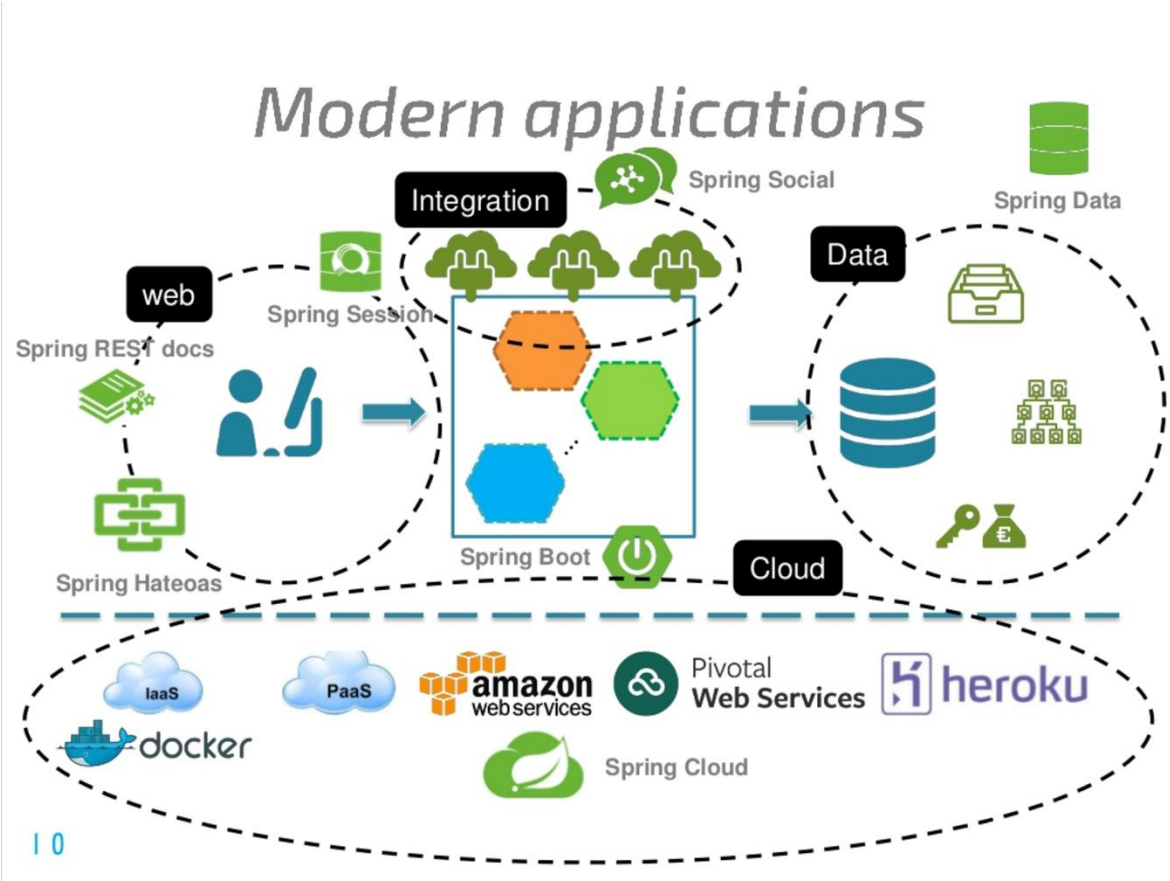
Neke od ključnih značajki Spring Boot programskog okvira su:

- Spring Boot starters.
- Spring Boot autokonfiguracija.

- Elegantno i pristupačno upravljanje konfiguriranjem.
- Spring Boot actuator.
- Podrška za ugrađene *servlet* kontejnere (Tomcat, Glassfish …).

3.2. Moduli

Spring programski okvir se sastoji od raznih modula koji olakšavaju razvoj modernih web aplikacija. Na slici (Slika 3) je prikazana podjela modula, u središtu razvoja je Spring Boot koji olakšava integraciju s *Web*, *Integration*, *Data* i *Cloud* slojem. Neki od modula koje osigurava Web sloj su Spring *Hateoas*, Spring *Session* i Spring REST *docs*. Spring *Integration* sloj je zadužen za komunikaciju s društvenim mrežama, dok *Data* sloj osigurava Spring *Data* modul koji omogućava skladištenje podataka. Spring *Cloud* osigurava konfiguraciju i izgradnju distribuiranih sustava.



Slika 3. Spring moderna aplikacija[3]

[3] https://ordina-jworks.github.io/spring/2017/06/07/Spring-IO-2017-The-Spring-ecosystem.html; 14.02.2019.

3.2.1. Spring Boot Starters

Spring Boot nudi različite početne module kako bi se brzo i jednostavno započeo rad koristeći tehnologije poput SpringMVC, Spring Data JPA i Spring Security. Početni moduli unaprijed su konfigurirani s najčešće korištenim ovisnostima, tako da se ne mora ručno pretraživati kompatibilne verzije i tako ih konfigurirati. Tako, na primjer, `spring-boot-starter-data-jpa` sadrži sve ovisnosti koje su potrebne da bi se koristio Spring Data JPA, zajedno s Hibernate-om. [4] Ako se radi web aplikacija, onda se prvo odabire Web modul kod kreiranja Spring Boot projekta. S tim odabirom u "`pom.xml`" datoteci, u ovisnostima, dobit će se `spring-boot-starter-web`. Taj komad koda pomaže automatski dohvatiti ovisnosti za *spring-webmvc*, *jackson-json*, *validation-api*, *i tomcat*. Uz ove osnovne početne module koji se koriste za izradu web aplikacija, zanimljivi su i moduli za *security, test, mail, logging, thymeleaf i actuator*. Tako se dolazi do zaključka da su prednosti korištenja početnih modula kreiran projekt spreman za produkciju, konfigurirane ovisnosti i smanjeno vrijeme potrebno za konfiguraciju projekta. Listu i opis nekih od najbitnijih modula mogu se vidjeti na tablici (Tablica 1).

[4] Beginning Spring Boot 2; K. Siva Prasad Reddy, str. 21

Tablica 1. Popis početnih modula[5]

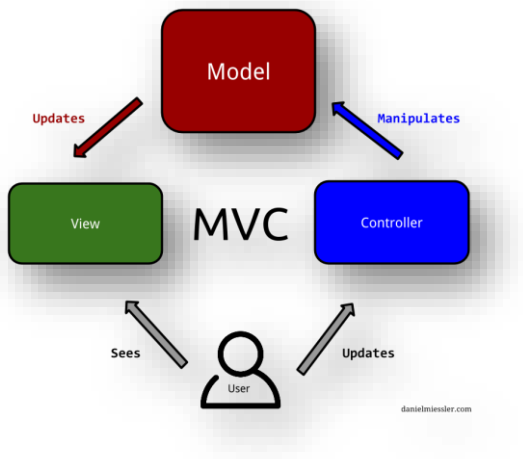
Starter	*Description*
spring-boot-starter	Ključni početni modul koji sadrži sve potrebne funkcionalnosti za početnu konfiguraciju projekta.
spring-boot-starter-actuator	Modul koji omogućuje nadgledanje funkcionalnosti aplikacije, a nudi i opciju za upravljanje i snimanje performansi.
spring-boot-starter-jpa	Modul koji osigurava potrebne biblioteke koje omogućuju korištenje Java Persistence API-a (Hibernate).
spring-boot-starter-security	Modul koji dohvaća sve ovisnosti potrebne za sigurnost aplikacije.
Spring-boot-starter-social-*	Modul koji osigurava integraciju s društvenim mrežama kao što su Facebook i Twitter i LinkedIn.
Spring-boot-starter-test	Modul koji osigurava ovisnosti za Spring testiranje, kao i programske okvire poput JUnit i Mockito.

3.2.2. Spring Boot Web modul

Jedna od ključnih značajki Spring Boot programskog okvira njegova je sposobnost automatskog konfiguriranja projekata, ovisno o odabranim početnim modulima. S obzirom na dodanu ovisnost `spring-boot-starter-web`, Spring prepoznaje da se želi raditi na izradi web aplikacije tako da automatski konfigurira projekte. Sukladno tome dodaje

[5] Spring-Boot-Cookbook; Alex Antonov; str. 3

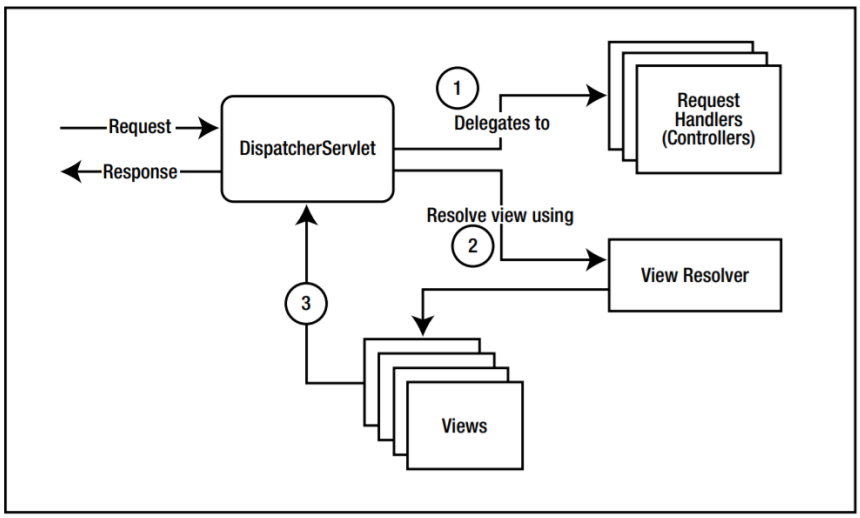
SpringMVC i Tomcat server. SpringMVC predstavlja skraćenicu za *model-view-controller*, općeprihvaćeni obrazac koji se koristi u softverskom inženjeringu. *Model* se sastoji od podataka, poslovnih pravila i logike. *View* je bilo kakav prikaz prethodno modeliranih podataka, dok *Controller* upravlja korisničkim zahtjevima, manipulira modelom koji osvježava *View*, kao što se vidi na slici (Slika 4). [6]



Slika 4. MVC arhitektura[7]

Spring Boot automatski konfigurira komponente, poput *DispatcherServlet* i *ViewResolver*, sa zadanim vrijednostima koje su se u prijašnjim verzijama Spring programskog okvira morale svaki put iznova konfigurirati. Također, Spring Boot podržava i automatski rješava konfiguraciju za JSP, *Thymeleaf* i druge *View* biblioteke. *DispatcherServlet* radi na način da dobije od korisnika zahtjev i taj zahtjev proslijedi *Controller-u*. Kada se zahtjev obradi, *ViewResolver* će po datotekama projekta pretražiti klase koje odgovaraju traženom zahtjevu i pronađeni *View* će se vratiti *DispatcherServlet*-u koji će korisniku vratiti odgovor. Opisana procedura je prikazana na slici (**Pogreška! Izvor reference nije pronađen.**). [8]

[6] https://danielmiessler.com/study/mvc/; 30.12.2018.
[7] Ibidem; 30.12.2018.
[8] Beggining Spring Boot 2; K.Siva Prasad Reddy; str. 109



Slika 5. *DispatcherServlet i ViewResolver*[9]

3.3. REST Arhitektura

Representational State Transfer (REST) je programski arhitektonski model koji sadrži skup ograničenja. Definirao ga je Roy Fielding 2000. godine u doktorskoj disertaciji. REST omogućuje izgradnju sustava koji pružaju interoperabilnost između različitih sustava. Drastičnim povećanjem mobilnih uređaja uvelike se povećala potreba za REST API servisima jer je postalo logično da web i mobilni klijenti konzumiraju API servise umjesto da razvijaju zasebne aplikacije. [10] Sistem koji je izveden u REST modelu naziva se RESTful.

RESTful ograničenja [11]:

- Klijent – poslužitelj.
- Neovisnost između zahtjeva.
- Priručna memorija.
- Jedinstveno sučelje.
- Slojeviti sustav.
- Kod na zahtjev.

[9] Beginning Spring Boot 2; K. Siva Prasad Reddy; str. 108
[10] Ibidem; str. 133
[11] https://urn.nsk.hr/urn:nbn:hr:143:056520; str. 8-10

Klijent – poslužitelj

Uloga klijenta razdvojena je od uloge poslužitelja. Klijent se ne bavi pohranom podataka, nego to radi poslužitelj. Poslužitelj se brine o poslovnoj logici, ne zanima ga korisničko sučelje kod klijenta. Odvajanjem klijenta od poslužitelja omogućilo se zasebno razvijanje poslužitelja pa tako i klijenta.

Neovisnost između zahtjeva

Svaki poslani zahtjev od strane klijenta prema poslužitelju i poslužiteljev odgovor mora biti neovisan o prijašnjim zahtjevima.

Priručna memorija

Podrška za priručnu memoriju omogućila je skalabilnost i brži odziv na zahtjev, jer su podaci podijeljeni na one koji se spremaju u priručnu memoriju kako bi ih klijent mogao kasnije koristiti te na one koji se ne spremaju jer se njihova vrijednost mijenja.

Jedinstveno sučelje

Temeljni koncept REST sustava jedinstvena je adresa i pristupna točka koja se definira kao *Uniform Resource Identifier* (URI).

Slojeviti sustav

Svaki sloj je skalabilan i zaseban. Klijent ne zna je li povezan direktno s poslužiteljem. Arhitektura je sastavljena od više slojeva, s čime se povećava složenost sustava, a samim time i potiče neovisnost o podlozi na kojoj se radi. Nedostatak takvih sustava je vrijeme obrade podataka.

Kod na zahtjev

Mogućnost prilagođavanja i proširivanja funkcionalnosti na klijentskoj strani dodavanjem JavaScript koda.

Najvažnije HTTP metode :

- `GET` - dohvaća kolekciju resursa ili jedan resurs po identifikatoru.
- `POST` - dodavanje novih resursa.
- `PUT` - ažuriranje željenog resursa.
- `DELETE` - brisanje resursa po identifikatoru.

REST API i SpringMVC

Kako SpringMVC pruža podršku za izgradnju RESTful servisa, tako i Spring Boot olakšava auto konfiguracijom. U programskom isječku (Programski isječak 3) se vidi SpringMVC i REST API na djelu gdje se dohvaća korisnik po identifikatoru i to na adresi "/api/users/{id}".

```
@RestController
@RequestMapping("/api/users")
public class UserController {
@Autowired
private UserService userService;
@GetMapping("/{id}")
public ResponseEntity<?>getUserById(@PathVariable Long id){
    User user=userService.findUserByID(id);
    return new ResponseEntity<>(user, HttpStatus.OK);
}}
```

Programski isječak 3. SpringMVC i REST *endpoint*

3.3.1. Prednosti Spring Boot-a i REST arhitekture

Prednosti Spring Boot-a

Dodajući razne funkcionalnosti, Spring programski okvir postao je previše kompleksan, osobito zbog dugotrajne konfiguracije, koja je bila potrebna za pokretanje novog projekta. Na podlozi od Spring programskog okvira predstavljen je Spring Boot, osmišljen kako bi uštedio vrijeme i ubrzao proces pokretanja novog Spring projekta. Prednost Spring Boot programskog okvira je ta da nema potrebu za razvijanjem WAR datoteka, nego je moguće razvijati JAR datoteku. Olakšava serverski dio posla jer automatski ugrađuje *Tomcat*, *Jetty* ili *Undertow* server, ovisno o potrebama i želji programera. Za razliku od Spring programskog okvira, čija se konfiguracija razvijala u XML formatu, Spring Boot ne zahtijeva takav pristup konfiguraciji. Osim što je pokretanje aplikacije postalo lagano, menadžment i modifikacije su olakšane, što samim time dovodi do projekta spremnim za stavljanje u produkciju. [12]

[12] https://dzone.com/articles/understanding-the-basics-of-spring-vs-spring-boot; 01.01.2019.

Prednosti REST arhitekture

Ključne prednosti u razvoju aplikacija korištenjem REST arhitekture su odvojenost između klijenta i servera (korisničko sučelje je odvojeno od poslužitelja i pohrane podataka). Velika je prednost u tome što razvoj korisničkog sučelja može biti na bilo kojoj platformi. Time se povećava skalabilnost te se omogućava samostalan razvoj različitih komponenti. Usporedbe radi, razvoj web stranica u Java programskom jeziku bio je ograničen uglavnom na razvoj JSP stranica koje su prikazivale sadržaj. Korištenjem REST servisa može se koristiti bilo koji popularni programski okvir za izradu web stranica (Angular, Vue ili React) i time se postiže fleksibilnost. Na slici (Slika 6) je prikazana klijentska strana aplikacije razvijena pomoću React-a, gdje klijent šalje zahtjev za dohvat svih projekata, te serverska strana aplikacije razvijena s Spring Boot programskim okvirom koji odrađuje poslovnu logiku, dohvaća podatke iz MySQL baze podataka i šalje odgovor s listom projekata klijentskoj strani aplikacije koja rezultat prikazuje klijentu.

Slika 6. REST komunikacija[13]

Razdvajanjem klijenta od poslužitelja postiže se da svaki razvojni tim može proširiti aplikaciju bez previše problema. Timovi mogu migrirati na druge programske okvire te izvršiti sve vrste promjena u bazi podataka, pod uvjetom da su podaci iz zahtjeva ispravno poslani. Najveća je fleksibilnost neovisnost o vrsti platforme ili jezika. REST API uvijek se prilagođava platformi koja se koristi. Jedino je bitno da se odgovori na zahtjeve uvijek odvijaju na jeziku koji se koristi za razmjenu informacija, i to u pravilu XML ili JSON. [14]

[13] Izvor: Vlastiti rad autora; 02.01.2019.

[14] https://bbvaopen4u.com/en/actualidad/rest-api-what-it-and-what-are-its-advantages-project-development; 02.01.2019.

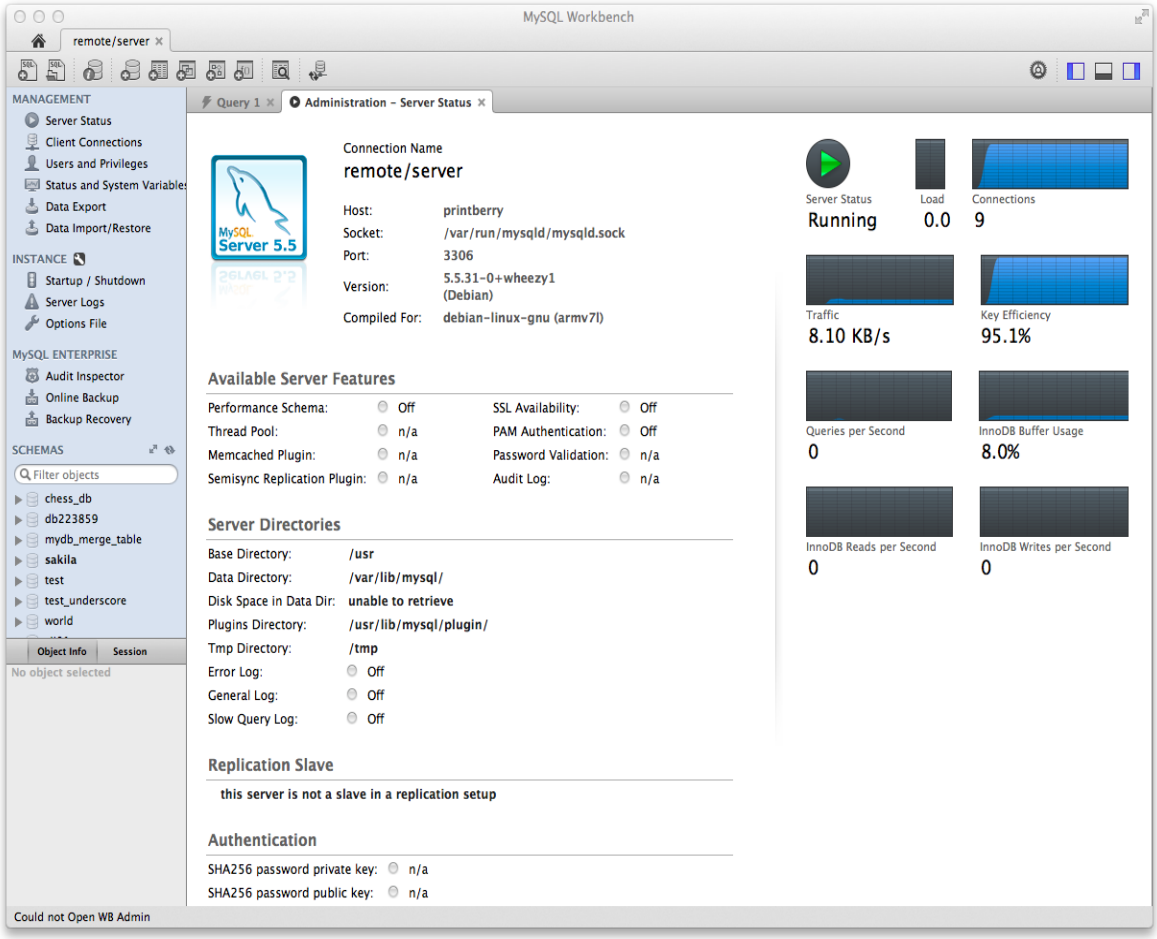
3.4. Baza podataka

Temelj i sastavni dio svake poslovne aplikacije je baza podataka. Baza podataka osigurava skladištenje informacija i manipulaciju podacima te je zbog toga neizostavni dio svake aplikacije. Spring Boot olakšava početak razvijanja web aplikacija, jer dodavanjem `spring-boot-devtools` početnog modula osigurava ugrađenu H2 bazu podataka za koju nije potrebna nikakva konfiguracija. H2 baza podataka vrlo je korisna u početnim fazama razvoja jer se pokreće zajedno s Tomcat serverom te joj se pristupa na stranici `localhost:8080/h2`. Kada se prijavi, vide se sve tablice koje su se kreirale i lako se može testirati unos podataka. Razvojem aplikacije i povećanjem kompleksnosti treba se prebaciti na ozbiljnije relacijske baze podataka.

3.4.1. MySQL

MySQL je besplatan program otvorenog koda koji služi za upravljanje bazom podataka i zbog toga je stekao veliku popularnost. Na tržištu je od 1995. godine, tako da ga koriste mnoge web aplikacije te ima podršku od brojnih programskih jezika, kao što su: Java, Python, PHP itd. Da bi se koristila MySQL baza, prvo je potrebno preuzeti *MySQL Community Server* s https://dev.mysql.com/downloads/mysql/. Prilikom instalacije postavlja se inicijalna šifra za MySQL *root*. Nakon toga se kreiraju korisnici koji će moći kreirati i koristiti baze podataka. Instalacija dohvaća i instalira dodatne proizvode koji su potrebni za ispravno korištenje i funkcioniranje servera i baza podataka. Neki od programa su: *MySQL Connector, MySQL Shell, MySQL Server, MySQL Workbench* itd. Pomoću *MySQL Shell-a,* upisuje se inicijalnu šifru koja je zadana za *MySQL root*. *MySQL Server* omogućuje komunikaciju jer pokreće server kojemu se pristupa iz Java web aplikacije. *MySQL Workbench*, čiji izgled se može vidjeti na slici (Slika 7), vizualni je alat za dizajn baze podataka koji omogućuje modeliranje podataka, razvoj baza podataka, konfiguraciju poslužitelja, administraciju korisnika. [15]

[15] https://hr.wikipedia.org/wiki/MySQL; 03.01.2019.



Slika 7. MySQL Workbench[16]

3.4.2. Spring Data JPA

JPA, *Java Persistance* API programski je okvir koji služi za relacijsko mapiranje objekata (ORM). Najpopularnija implementacija ORM-a je Hibernate. ORM omogućava kreiranje tablica i entiteta u bazama podataka koji konfiguracijom pomoću Hibernate-a postaju POJO (*Plain Old Java Objects*) klase u Java aplikaciji. Također je moguće kreirati klase unutar Java aplikacije koje se mapiraju u bazu podataka i postaju tablice sastavljene od onih vrijednosti koje su se koristile prilikom kreiranja klasa. `String` varijabla imena *username* postaje *nvarchar* varijabla naziva *username* u bazi podataka. Spring Data JPA pruža podršku za implementaciju korisnih sučelja kao što su "`CrudRepository`", "`JpaRepository`" i "`PagingAndSortingRepository`".[17] Nakon kreiranja klasa i rješavanja entiteta,

[16] https://www.mysql.com/common/images/products/MySQL_Workbench_Mainscreen_Mac.png;03.01.2019.
[17] Beginning Spring Boot 2; K. Siva Prasad Reddy, str. 83

sljedeći korak je kreiranje sučelja. Korištenjem "PagingAndSortingRepository" sučelja mogu se koristiti metode findAll(Pageable pageable)- koja vraća stranicu entiteta i findAll(Sort sort) – koja vraća sve entitete sortirane po željenom uvjetu. Pošto "JpaRepository" sučelje nasljeđuje "PagingAndSortingRepository" sučelje, koji nasljeđuje "CrudRepository", korištenjem "JpaRepository" sučelja automatski se nasljeđuju sve metode iz sva tri sučelja. Popis metoda iz "JpaRepository" sučelja koje su preuzete s "Javadoc" dokumentacije su prikazane na slici (Slika 8):

Slika 8. "JpaRepository" metode[18]

[18] https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html; 04.01.2019.

Korištenjem sučelja "`CrudRepository`", nasljeđuje se sučelje koje se želi kreirati pa se time dobiju metode koje su usko vezane s `CRUD` metodama. Popis metoda iz sučelja "`CrudRepository`" koje su preuzete s "Javadoc" dokumentacije se mogu vidjeti na slici (Slika 9). Bitna stavka kod korištenja triju navedenih sučelja je da se metode koje dolaze u paketu s njima ne moraju implementirati, ali moguće je kreirati i vlastite metode unutar kreiranog repozitorija.

Method and Description
`count()` Returns the number of entities available.
`delete(T entity)` Deletes a given entity.
`deleteAll()` Deletes all entities managed by the repository.
`deleteAll(Iterable<? extends T> entities)` Deletes the given entities.
`deleteById(ID id)` Deletes the entity with the given id.
`existsById(ID id)` Returns whether an entity with the given id exists.
`findAll()` Returns all instances of the type.
`findAllById(Iterable<ID> ids)` Returns all instances of the type with the given IDs.
`findById(ID id)` Retrieves an entity by its id.
`save(S entity)` Saves a given entity.
`saveAll(Iterable<S> entities)` Saves all given entities.

Slika 9. "`CrudRepository`" metode[19]

[19] https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html; 04.01.2019.

Na primjeru programskog isječka (Programski isječak 4) je prikazana klasa "User" s anotacijom @Entity:

```
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @NotBlank(message = "Please enter your full name")
    private String fullName;
    @NotBlank(message = "Please enter your rank")
    private String rank;
      //getters and setters
}
```

Programski isječak 4. Klasa "User"

U programskom isječku (Programski isječak 5) je prikazano Sučelje "UserRepository" koje nasljeđuje "CrudRepository", predajući mu klasu i tip podataka:

```
@Repository
public interface UserRepository extends
CrudRepository<User,Long> {

    User getById(Long Id);

    User findByVoib(String voib);

    @Override
    Iterable<User> findAll();
}
```

Programski isječak 5. Sučelje "UserRepository"

Osim što Spring Data JPA omogućuje navedene ugrađene metode, velika prednost je u generiranju metoda vezanih za varijable kreirane u klasama. U ovom slučaju, sučelje "UserRepository" ima sposobnost kreirati metode poput ispod navedenih, jer u "User" klasi postoje varijable naziva *Age*, *Rank i Voib*:

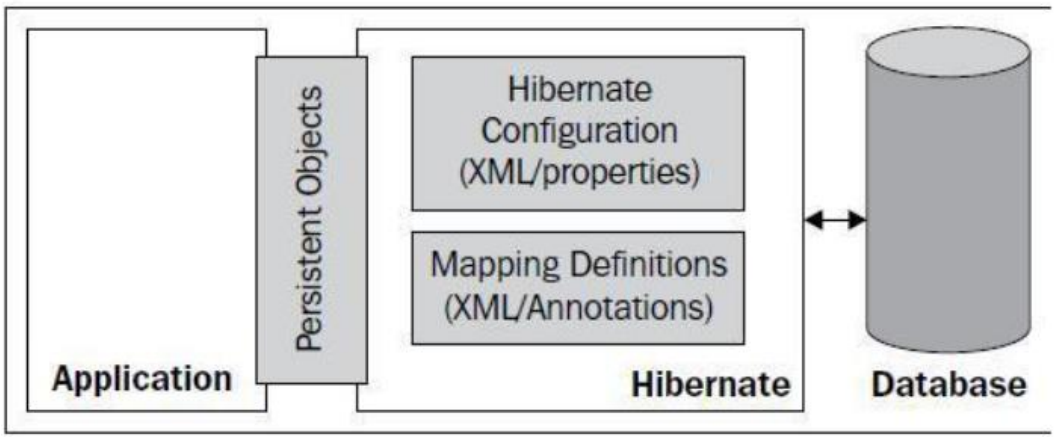
- `User getByAge(Long id);`
- `User findByRank(String rank);`
- `User findByVoib(String voib);`

3.4.3. Hibernate

Hibernate je besplatni programski okvir koji želi riješiti manipulaciju podacima u Javi i pojednostaviti razvoj Java aplikacija tako što povezuje aplikaciju s bazom podataka i omogućuje objektno relacijsko mapiranje (ORM). Hibernate se bazira na POJO objektima, čime smanjuje ovisnost o programskom kodu iz Spring programskog okvira, a time dobije produktivan i prenosiv razvoj. [20] U usporedbi s JDBC-om, kod Hibernate-a nije potrebno voditi brigu o spajanju na bazu podataka, o otvaranju i zatvaranju veza, jer Hibernate sve to odrađuje sam. Hibernate nije zamjena za JDBC, nego predstavlja alat kojim se spaja na bazu podataka koristeći JDBC.[21] On prezentira objektno orijentirani, aplikacijski pogled na bazu podataka. Hibernate omogućuje naprednije upravljanje podacima. Jedan od primjera je spremanje često pozivanih podataka u predmemoriju s čime se dobije brži odgovor i rezultat, jer se podaci dohvaćaju iz memorije, a ne iz baze podataka. Drugi primjer je *lazy loading* koji pruža mogućnost odgode inicijalizacije objekta sve dok se on ne pozove. [22]

Hibernate Query Language (HQL) je jezik kojim se dohvaćaju podaci iz baze podataka. Prilikom pokretanja aplikacije, HQL izrazi se pretvaraju u odgovarajuće SQL izraze, ovisno o bazi podataka koja se koristi. Nemaju sve baze podataka istu sintaksu za upite, stoga Hibernate sadrži funkcionalnost naziva SQL *dialect*.[23] U "`application.properties`" datoteci unutar aplikacije, osim postavki za spajanje na bazu, predaje se i određeni SQL *dialect* poziv. Sama srž Hibernate-a leži u njegovoj konfiguraciji, počevši od one u "`application.properties`" datoteci, gdje se postavljaju odgovarajuće informacije o bazi podataka: korisničko ime, šifra, klasa upravljačkog programa, Hibernate *dialect* itd. Sve te informacije su potrebne Hibernate-u za komunikaciju, spajanje, rad i izvođenje promjena na bazi podataka.

[20] Spring-Persistence-with-Hibernate; Ahmad Seddighi; str. 40
[21] Ibidem; str. 42-44
[22] Ibidem; str. 63
[23] Ibidem; str. 67-71



Slika 10. Hibernate arhitektura [24]

Na slici (Slika 10) se vidi da je druga temeljna stavka Hibernate-a mapiranje pomoću anotacija. Mapiranjem se postiže stvaranje tablica u bazi podataka temeljeno na imenu klasa. Prva anotacija koja se koristi prilikom izrade klasa je `@Entity`. Postavlja se odmah iznad imena klase, a označava klasu kao entitet u bazi podataka.[25] Nakon što se klasa označi s `@Entity`, idući korak je odrediti primarni ključ u bazi podataka, prikazan u programskom isječku (Programski isječak 6). To se radi s anotacijama `@Id`, koja predstavlja identifikator klase i `@GeneratedValue` koja označava tip kreiranja primarnog ključa. U ovom je slučaju odabran tip čuvanja `IDENTITY`, time se govori bazi podataka da je sama odgovorna o generiranju primarnih ključeva. Osim `IDENTITY` tipa, kod biranja strategije čuvanja primarnog ključa postoje još i `SEQUENCE`, `TABLE` i `AUTO`, koji dozvoljava Hibernate-u da odabere odgovarajući generator ovisno o bazi podataka. [26]

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
```

Programski isječak 6. Generiranje primarnog ključa

Ostale anotacije prikazane u programskom isječku (Programski isječak 7), koje nisu neophodne za mapiranje klasa u tablice, u bazi podataka, su [27]:

- `@Table` – služi za definiranje imena tablice.

[24] Spring-Persistence-with-Hibernate; Ahmad Seddighi; str. 71
[25] Ibidem; str. 235
[26] Ibidem; str. 237
[27] http://javaprogramiranje.blogspot.com/2012/01/lista-hiberante-anotacija.html; 6.1.2019.

- `@Transient` – služi za označavanje onih varijabli u klasi koje se ne žele prikazati u bazi podataka.
- `@NotNull` – anotacija kojom se daje do znanja da se ne želi `NULL` vrijednost u bazi podataka.
- `@Size` – definiranje veličine polja, kada se želi zadati minimum ili maksimum kod unosa.
- `@Column` – služi za imenovanje retka u tablici, ako se redak želi nazvati drugačije nego što se zove u klasi te ako se želi ograničiti redak da bude jedinstven i da se ne smije mijenjati.
- `@JoinColumns` – određivanje po kojoj će se osnovi vršiti mapiranje.
- `@OneToOne` – mapiranje objekata jedan naprama jedan.
- `@OneToMany` – mapiranje objekata jedan naprama više.
- `@ManyToOne` – mapiranje objekata više naprama jedan.
- `@ManyToMany` – mapiranje objekata više naprama više.

```
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank(message = "Please enter your full name")
    private String fullName;

    @NotBlank(message = "Please enter your rank")
    private String rank;

    @NotBlank(message = "Enter valid identify")
    @Size(min=5, max=5, message = "Please use 5 characters")
    @Column(updatable = false, unique = true)
    private String voib;

    @NotNull(message = "Please enter age")
    private Integer age;

    @OneToOne(fetch = FetchType.EAGER, cascade =
CascadeType.ALL, mappedBy = "user")
```

```
    @JsonIgnore
    private Category category;
//getters and setters
}
```

Programski isječak 7. "User" klasa i Hibernate anotacije

Baze podataka ne služe samo za skladištenje podataka, nego i za dohvat podataka, a ako se žele dohvatiti složenije stvari, potrebno je povezati tablice. U bazi podataka to se radi identificiranjem tablica i stupaca pomoću primarnih i stranih ključeva. Međutim, u Javi se to odrađuje zahvaljujući Hibernate-u i asocijacijama. Kao što je navedeno ranije, postoje četiri takve asocijacije, a to su: `@OneToOne`, `@OneToMany`, `@ManyToOne` i `@ManyToMany`. [28]

- `@OneToOne` - najčešće se koristi, i to u želji da se poveže redak iz jedne tablice sa stranim ključem u drugoj tablici, a to u praksi izgleda ovako:
  Klasa "`User`" sadrži varijablu `category`, a ona je označena određenim anotacijama i uvjetima, pa tako `fetch` označava na koji način će Hibernate dohvatiti podatke. U ovom slučaju, kada je `FetchType` postavljen na `EAGER`, učitava se sve vezano uz taj objekt. `Cascade` je postavljen na `All`, što znači da su dopuštene sve radnje na polju `category` (`delete, detach, merge, remove, persist`), dok `mappedBy` određuje na koju se varijablu odnosi ova anotacija.
  ```
  @OneToOne(fetch = FetchType.EAGER, cascade =
  CascadeType.ALL, mappedBy="user")
  @JsonIgnore
  private Category category;
  ```
- `@ManyToOne` - označava varijablu `user` u klasi "`Category`" kao strani ključ i veže ju na tablicu `User`, dok se `@JsonIgnore` postavlja da bi se izbjegla rekurzivna greška.
  ```
  @ManyToOne
  @JoinColumn(name = "user_id")
  @JsonIgnore
  private User user;
  ```

[28] https://thoughts-on-java.org/ultimate-guide-association-mappings-jpa-hibernate/; 06.01.2019.

- `@OneToMany` - ima istu funkcionalnost kao i `@ManyToOne`, samo s druge strane gledišta. U ovom primjeru se u klasi "`Board`" kreira strani ključ za `projectTasks`.

```
@OneToMany(cascade = CascadeType.REFRESH,fetch =
FetchType.EAGER,mappedBy = "board", orphanRemoval =
true)
private List<ProjectTasks> projectTasks=new
ArrayList<>();
```

3.5. Sigurnost web aplikacije

Sigurnost aplikacija bitna je stavka kod dizajna i izrade softverskih aplikacija, jer osigurava da pristup određenim resursima imaju samo autorizirani korisnici. Dvije ključne stavke kod osiguravanja aplikacija su autentifikacija i autorizacija. Provjera autentifikacije odnosi se na provjeru korisnika, što obično znači provjeru njegovih vjerodajnica. Provjera autorizacije se odnosi na provjeru prava, odnosno koje su aktivnosti odobrene za određenog korisnika.

3.5.1. Spring Boot security

Spring Security moćan je i fleksibilan programski okvir koji pruža osiguravanje web aplikacija, autentifikaciju, autorizaciju te enkodiranje podataka. Osim što se koristi za osiguravanje Java web aplikacija, može se koristiti i za osiguravanje aplikacija koje nisu bazirane na Spring programskom okviru. [29] Da bi se mogao upotrebljavati Spring Security, potrebno je dodati ovisnosti u "`pom.xml`" datoteku. Kada se doda `spring-boot-starter-security`, auto konfiguracija rješava sve probleme, pa se tako novim pokretanjem aplikacije dobije niz novih mogućnosti kao što su:

- Sve su `HTTP` krajnje točke osigurane osnovnom provjerom autentifikacije.
- Kreiran je jedan korisnik s korisničkim imenom i zaporkom.
- Rute i folderi, kao što su: `/css, /js, /images, /favicon.ico`, ignoriraju se prilikom sigurnosnih provjera.
- Sve sigurnosne radnje i akcije spremaju se u Spring Security kontejner, bilo da se radi o uspješnim zahtjevima ili neuspješnim autorizacijama i autentifikacijama.

[29] Beggining Spring Boot 2; K.Siva Prasad Reddy; str. 175

- Aktiviraju se sigurnosne značajke kao što su `XSS`, `CSRF`, kontrola predmemorije.

Dodavanjem Spring Security modula, pokretanjem aplikacije i pristupom zadanom URL-u `http://localhost:8080`, pojavit će se forma za unos korisničkog imena i zaporke. Zadano korisničko ime je `user`, a zaporka se generira svakim pokretanjem aplikacije i može se pronaći u ispisu logova u konzoli. Ako se želi postaviti statična zaporka i novo korisničko ime, kao što je prikazano u programskom isječku (Programski isječak 8), može se postaviti u datoteci "`application.properties`".[30]

```
spring.security.user.name=marin
spring.security.user.password=secret
```

Programski isječak 8. Sigurnosne postavke

Kako bi se iskoristio puni potencijal Spring Security-a, potrebno je kreirati klasu "`SecurityConfig`" koja nasljeđuje `WebSecurityConfigurerAdapter. @Configuration i @EnableWebSecurity` anotacije isključuju zadane postavke za sigurnost kako bi se mogle konfigurirati nove postavke za sigurnost unutar "`SecurityConfig`" klase, prikazane u programskom isječku (Programski isječak 9).

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(
        securedEnabled = true,
        jsr250Enabled = true,
        prePostEnabled = true
)
public class SecurityConfig extends
WebSecurityConfigurerAdapter {}
```

Programski isječak 9. Konfiguracijska klasa za Spring Security

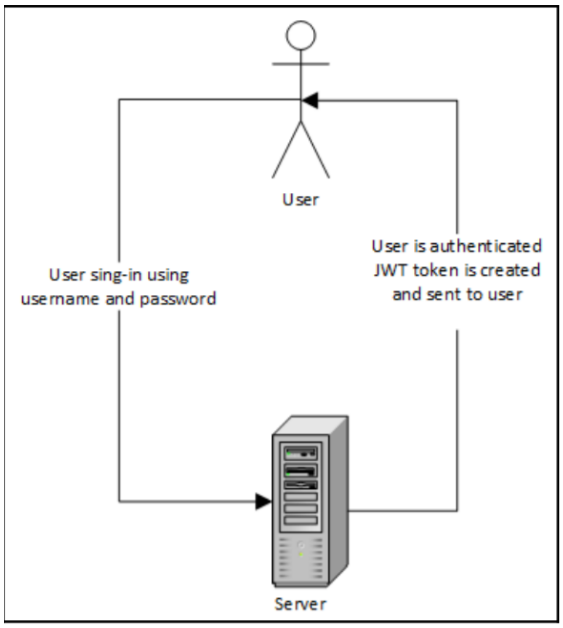
Nasljeđivanjem "`WebSecurityConfigurerAdapter`" klase, osiguraju se osnovne sigurnosne postavke. U situaciji kada se žele unijeti detaljnije postavke sigurnosti, koristi se metoda `configure()`. Unutar metode `configure(HttpSecurity http)`, mogu se definirati koje krajnje točke su osigurane, a koje nisu. Metoda `configure(AuthenticationManagerBuilder)` služi za konfiguraciju rola za

[30] Learning-Spring-Boot; Greg L. Turnquist; str. 363-396

korisnika te za autentifikaciju i autorizaciju, dok `configure(WebSecurity)` služi za konfiguraciju filtriranja. [31]

3.5.2. JSON web token

JSON web token standard je koji definira kompaktni način za sigurnu autentifikaciju web aplikacija u obliku JSON objekta. JWT se šalje putem URL-a u POST parametru, ili u zaglavlju, te sadrži sve potrebne informacije o korisniku. Za izradu ovog završnog rada koristi se React, prema tome proizlazi i korištenje JWT-a. JSON web token sastoji se od tri dijela. Prvi dio je zaglavlje koji opisuje tip tokena i koji je *hash* algoritam korišten. Drugi dio tokena su informacije o korisniku, dok je treći dio potpis koji potvrđuje da se token nije mijenjao na putu. Ta su tri dijela tokena odvojena točkom. Na slici (Slika 11) može se vidjeti proces kroz koji JWT token prolazi, gdje nakon uspješne autentifikacije zahtjev koji je poslan od korisnika uvijek mora sadržavati JWT token.



Slika 11. JWT proces[32]

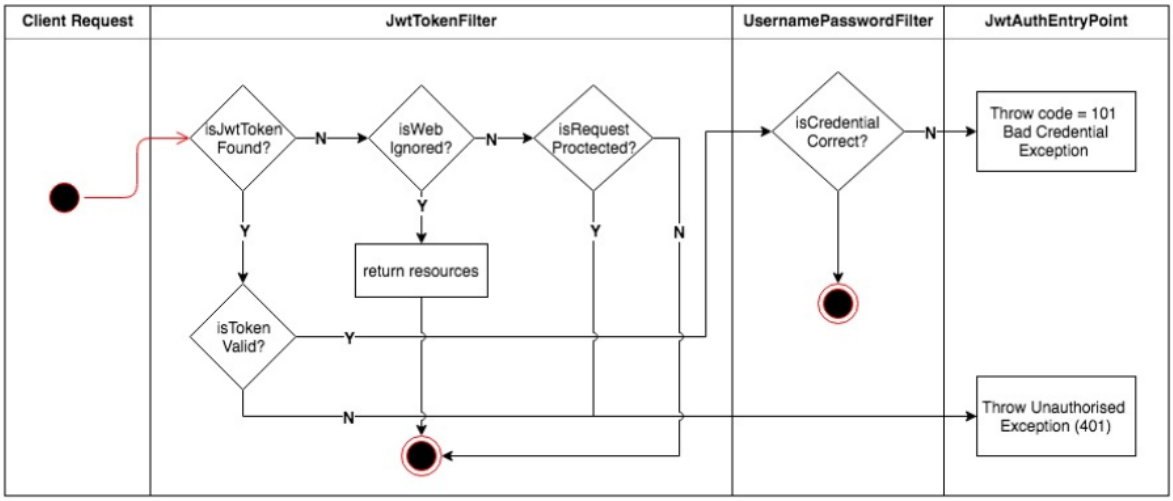
Postupak postavljanja i konfiguracije JWT-a za rad kreće od kreiranja klase `Constant`, gdje se postavljaju statičke vrijednosti potrebne za ispravan rad JWT-a. U toj klasi postavljaju se statične varijable: `EXPIRATION_TIME` - kojim se definira koliko dugo je token validan, `TOKEN_PREFIX` - koji definira prefiks ispred tokena, a obično se koristi

[31] Hands-On Full Stack Development with Spring Boot 2.0 and React; Juha Hinkula;(2018); str. 70
[32] Hands-On Full Stack Development with Spring Boot 2.0 and React; Juha Hinkula;(2018); str. 80

`Bearer` struktura, te `SIGNING_KEY` – koji određuje kojim se algoritmom potpisuje odgovarajući token. [33] Idući korak je kreiranje POJO klase koja čuva uvjerenje za autentifikaciju. Ona se sastoji od dvije varijable: `username` i `password`.

Idući korak je implementacija autentifikacijskog filtera za provjeru izdanih JWT tokena korisnicima koji šalju vjerodajnice. Kreiranjem klase "`JwtAuthFilter`", nasljeđuje se "`OncePerRequestFilter`" te se prepisuje metoda `doFilterInternal`. U trenutku kada korisnik pošalje zahtjev prema serveru, ovaj se filter okida i provjerava je li JWT token ispravan. Unutar metode `doFilterInternal`, bitan dio logike odrađuje `jwtTokenProvider`, kreirana klasa koja se koristi za validaciju tokena. Nakon autentifikacije potrebno je provjeriti jesu li korisnikove vjerodajnice ispravne, jer postoji šansa da je token ukraden od strane hakera koji želi ubrizgati virus kroz zahtjev. U ovoj situaciji pomaže Spring Security koji zahtijeva da programeri implementiraju "`UserDetails`" sučelje radi olakšavanja procesa. Implementacijom "`UserDetails`" sučelja dobiju se razne metode koje se mogu koristiti i modificirati unutar `User` klase. Nakon provjere autentičnosti i autorizacije korisničkih vjerodajnica, potrebno je obraditi iznimku tijekom tih procesa. Kreira se "`JwtAuthEntryPoint`" klasa koja implementira sučelje "`AuthenticationEntryPoint`". To sučelje omogućuje metodu `commence` koja odrađuje proces iznimki i povratnih poruka. Konfiguracijom filtera i validacijom tokena, osigurala se ispravna i sigurna provjera JWT tokena. Sljedeći korak je konfiguriranje ruta zahtjeva, tako da *backend* server može izvršiti sigurnosnu provjeru. Modifikacijom "`SecurityConfig`" klase unutar `configure(HttpSecurity)` metode, definira se koji su resursi javni, a koji su osigurani te se postavlja podrška za CORS (Cross-Origin Resource Sharing). Unutar `configure(AuthenticationManagerBuilder)` metode definira se način enkripcije zaporki te se implementira učitavanje podataka od korisnika. [34] Na slici (Slika 12) vidi se proces rada JWT autentifikacije sa Spring Security.

[33] Ibidem; str. 68-80
[34] http://kelvinleong.github.io/authentication/2018/06/06/JWT-Authentication.html; 13.01.2019.



Slika 12. JWT autentifikacijski proces [35]

[35] Ibidem; 13.01.2019.

4. Prezentacijski sloj

Kostur svake poslovne aplikacije je dohvat podataka, obrada podataka i prikaz podataka. Java programski jezik nudi vlastite implementacije prikaza podataka. Vrlo popularan i dugo korišten je JSP (*JavaServer pages*) predložak koji omogućuje kreiranje dinamičkih web stranica baziranih na HTML-u i XML-u.[36] Nešto modernije izdanje za prikaz web stranica je *Thymeleaf* predložak koji je čisti HTML, ali sadrži iste funkcionalnosti kao i JSP. Zbog fleksibilnosti i prednosti korištenja novih tehnologija i REST arhitekture, za potrebe završnog rada koristit će se React programski okvir.

4.1.1. React

Aplikacija bazirana na REST arhitekturi nudi fleksibilnost kod odabira određenog programskog okvira. Najpoznatiji i najpopularniji programski okviri trenutno su: React, Angular i Vue. Za potrebe izrade praktičnog dijela završnog rada koristio se React. React je JavaScript biblioteka razvijena i održavana od strane Facebook-a i zajednica pojedinih programera i tvrtki. Prvi korak koji je potreban za izradu React aplikacija je instalacija Node.js-a, okruženja poslužitelja na bazi otvorenog izvornog JavaScript koda. S instalacijom Node.js-a dolazi i `npm`, upravitelj paketa za JavaScript koji se koristi prilikom rada u terminalu za pokretanje aplikacije, dohvaćanje i instalaciju raznih modula i slično. Drugi korak potreban za izradu React aplikacija je instalacija Visual Studio Code-a (VS Code), Microsoftovog besplatnog alata koji se koristi za razne programske jezike. Nakon instalacije Node.js-a i VS Code-a, potrebno je otvoriti *Command Prompt* ili terminal u VS Code-u, upisati `npm install -g create-react-app` i tako globalno na cijeli sustav instalirati `create-react-app` pokretač koji se koristi za izradu React aplikacija. Tek nakon što se sve instaliralo i konfiguriralo može se kreirati React aplikacija, a to se radi naredbom `create-react-app firstapp`. Pozicioniranjem u terminalu, u direktorij gdje je aplikacija kreirana, koristi se naredba `npm start` te se pokreće aplikacija na portu 3000. Zatim se automatski pokreće Internet preglednik s početnim zadanim prozorom i porukom dobrodošlice. Početna struktura aplikacije sastoji se od nekoliko bitnih datoteka, a jedna od njih je `"index.js"`, koja s linijom `ReactDOM.render(<App />,`

[36] https://en.wikipedia.org/wiki/JavaServer_Pages; 20.1.2019.

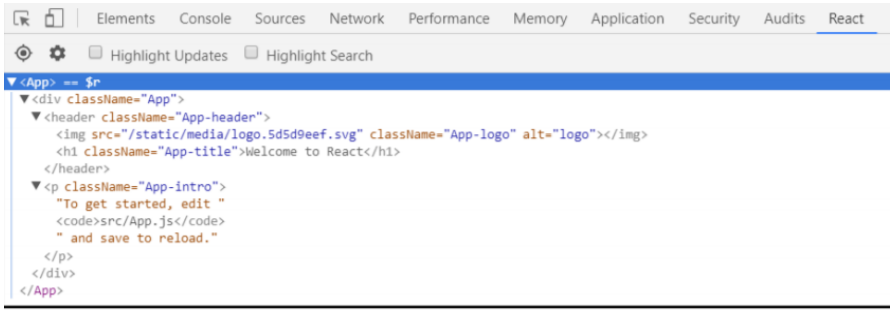
`document.getElementById('root'));` prikazuje glavnu aplikaciju s identifikatorom *'root'* te govori "`App.js`" klasi što se prikazuje u "`index.html`" klasi.

Skladište instaliranih modula i korištenih biblioteka u React aplikaciji nalazi se u klasi "`package.json`" koja je prikazana u programskom isječku (Programski isječak 10). Ono što je u Java aplikaciji "`pom.xml`" datoteka, to je "`package.json`" u React-u.

```
{
      "name": "firstapp",
      "version": "0.1.0",
      "private": true,
      "dependencies": {
      "axios": "^0.18.0",
      "react": "^16.6.1",
      "react-bootstrap-table": "^4.3.1",
      "react-dom": "^16.6.0",
      "react-redux": "^5.1.1",
      "react-router-dom": "^4.3.1",
      "redux": "^4.0.1",
      "redux-thunk": "^2.3.0",
                  },
      "proxy": "http://localhost:8082",
      "eslintConfig": {
      "extends": "react-app"
      }
}
```

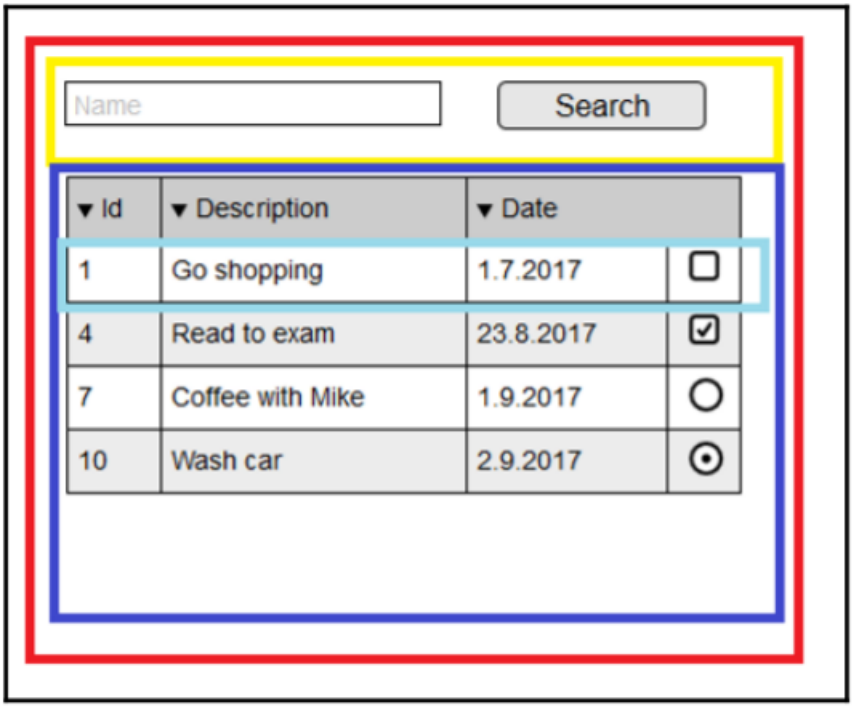
Programski isječak 10. `package.json` klasa s instaliranim modulima

Nakon upoznavanja s osnovnim klasama u React aplikaciji, potrebno je instalirati *React Developer Tools* pomoću *Google Chrome* trgovine. To je koristan alat za jednostavnije pronalaženje grešaka, čiji se izgled može vidjeti na slici (Slika 13).



Slika 13. React Developer Tools plugin[37]

Temeljna stvar koja je specifična za React su komponente. Osnovna ideja u razvoju React aplikacija je razvoj komponenti čiji primjer se može vidjeti na slici (Slika 14). Komponente se rade kreiranjem obične JavaScript klase koja se nakon kreiranja može izvoziti (engl. *export*). Dakle, na jednoj formi koja se sastoji od tražilice, tablice ili tipki, svaka od tih stavki treba biti zasebna komponenta koje se pozivaju unutar određene klase koja objedinjuje formu. Hijerarhija kreće od roditeljske komponente koja u sebi ima komponente tražilice i tablice, a tablica u sebi ima komponentu retka u tablici. Takav protok podataka i informacija je jedini ispravan i pravilan.



Slika 14. React komponente [38]

[37] Hands-On Full Stack Development with Spring Boot 2.0 and React; Juha Hinkula;(2018); str. 103
[38] Ibidem; str. 106

Korisna funkcionalnost u razvoju React aplikacija uvoz je i izvoz različitih komponenti i modula. Slično kao u Javi, kada se žele koristiti metode iz repozitorija, određeni repozitorij se uveze da bi se njegove metode mogle koristiti, tako je i u React-u. Kako je React baziran na komponentama, tako ima puno vanjskih komponenti za korištenje. Ako se želi koristiti `react-table` ili `semantic` modul, to se postiže dohvaćanjem tih modula s naredbom u terminalu `npm install semantic-ui`. Tek se tada u klasama mogu uvesti i koristiti metode i funkcionalnosti tih uvezenih modula.

- `import React from 'react'` // uvoz zadanih modula
- `import { Component } from 'react'` // uvoz komponenti pod određenim nazivom
- `export default React` // izvoz zadanih modula
- `export export {Component}` // izvoz komponenti pod određenim nazivom

Unutar svake React komponente nalazi se funkcija `render()`, ona određuje prikaz HTML-a unutar React komponente. Zanimljivo je to što se unutar `return()` metode piše JSX (JavaScript eXtension) kod, koji predstavlja ekstenziju React-a koja dozvoljava pisanje JavaScript koda, tako da izgleda kao HTML. U programskim isječcima (Programski isječak 11) i (Programski isječak 12) može se vidjeti način pisanja JSX koda te prikaz kako se JSX pretvara prilikom pokretanja u standardni JavaScript kod.

```
<div>
  <img src="profile.jpg" alt="Profile photo" />
  <h1>Welcome back Ari</h1>
</div>
```

Programski isječak 11. JSX primjer programskog isječka

```
React.createElement("div", null,
  React.createElement("img", {src: "profile.jpg", alt:
"Profile photo"}),
  React.createElement("h1", null, "Welcome back Ari")
);
```

Programski isječak 12. Generirani JavaScript kod nastao iz JSX primjera

Kao što se vidi na programskim isječcima iznad, pisanje JSX koda puno je jednostavnije i prirodnije jer izgleda kao čisti HTML uz neke sitne razlike. Recimo, umjesto `<a href to=""></a>`, koji služi za navigaciju kod HTML-a, u JSX-u se koristi `<Link to=""></Link>` element. Isto tako, u React-u, pri pisanju JSX koda, atributi se ne

označavaju s `class`, nego s `className`, jer je `class` ključna riječ kod pisanja JavaScripta. Za prikaz podataka unutar komponenata koriste se State i Props. JavaScript objekti utječu na komponentu u smislu da se ona osvježi svaki put kada se stanje State-a ili Props-a promjeni. Props-i su nepromjenjivi, tako da ih komponenta kao komponenta ne može mijenjati, odnosno Props-i su dobiveni od roditeljske klase. React komponenta ima pristup Props-ima putem ključne riječi `this.props`. To bi izgledalo ovako: komponenta `Hello` pristupa `user` Props-u s `<h1>Hello{this.props.user}</h1>;` , dok roditeljska komponenta može poslati Props `Hello` komponenti s `<Hello user="John"/>`. Prilikom pokretanja aplikacije, kada se `Hello` komponenta prikaže, na ekranu će se ispisati *Hello John*. Za razliku od Props-a, State se može mijenjati unutar komponente. Prvotno stanje State-a se preda u konstruktoru, a pristupit mu se može s `this.state` objektom. State se ne može kao Props predavati iz roditeljske komponente u dječju komponentu, nego mu je opseg unutar komponente gdje je definiran. Props-i se predaju konstruktoru kao argument, a State se inicijalizira te mu se kasnije u komponenti pristupa s `{this.state.user}`. Primjer inicijalizacije State objekta je prikazan u programskom isječku (Programski isječak 13).

```
class Hello extends React.Component {
constructor(props) {
 super(props);
this.state = {user: 'John'} }
render() {
return Hello World {this.state.user};}}
```

Programski isječak 13. Prikaz inicijalizacije i pristupa State objekta

Vrijednost State objekta može se mijenjati s `this.setState` metodom koja je asinkrona, što znači da se ne zna trenutak kada se State ažurira, odnosno `setState` metoda ima unutar sebe funkciju koja se okine u trenutku kada se State ažurira. [39]

4.1.2. Redux

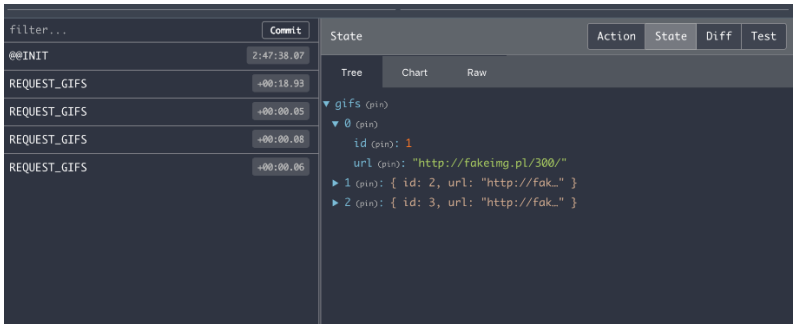
Redux je predvidljivi kontejner koji se koristi za čuvanje stanja u JavaScript aplikacijama. On nije programski okvir, nego je mala biblioteka koja ima određena pravila te zahtijeva određen način pisanja koda koji je vezan za stanje aplikacije. State je JavaScript objekt

[39] Hands-On Full Stack Development with Spring Boot 2.0 and React; Juha Hinkula;(2018); str. 95-125

kojem Redux nameće način izmjene tako da su sve informacije jednosmjerne, što znači da ne dolazi do mutacije objekta. Kada se želi napraviti promjena stanja State objekta, to se radi pomoću akcija, dok *reducer* izvršava samu promjenu. [40]

Redux se razvio kao podrška React-u, ali je počeo prikupljati veliku pažnju kod drugih programskih okvira kao što je Angular. Ključna prednost Reduxa je neovisnost o određenim programskim okvirima i bibliotekama jer se lagano može koristiti s bilo kojim JavaScript programskim okvirom kako bi obrađivao i mijenjao stanje u aplikacijama. Spajanje Reduxa s ostalim programskim okvirima odrađuje se tako što se koristi dodatna biblioteka, a u slučaju kada se želi spojiti React s Reduxom koristi se `react-redux` biblioteka. [41]

Prvi od koraka koji su potrebni da se počne koristiti Redux u React aplikaciji je instalacija Redux biblioteke pomoću `npm install redux`. Odmah zatim se instalira `react-redux`, te `redux-thunk` biblioteka koja rješava *middleware* kada akcije šalju podatke koji se moraju otpremiti. Nakon instalacije biblioteka poželjno je instalirati *Redux DevTools*, koji se može pronaći u *Google Chrome* trgovini. Na slici (Slika 15) vidi se izgled *plugina* koji olakšava rad i uvid u stanje JavaScript aplikacija.



Slika 15. Redux DevTools[42]

Ključne terminologije u Redux-u su: *actions i actions creators, reducers, middleware i store*. *Store* sadrži informacije o aplikaciji, ali ne sadrži korisničku logiku. Uloga *Store-a* primanje je akcija, prosljeđivanje akcija prema registriranom *middleware-u,* zatim, pomoću *reducers-a*, pregledavanje novog stanja aplikacije te spremanje. Jedini način na koji aplikacija mijenja stanje je procesiranje akcija. U većini slučajeva, akcije u Redux-u nisu ništa više od običnih JavaScript objekata koji su poslani u Store da bi promijenili stanje aplikacije. Budući da ti objekti sadrže određenu logiku i mogu se koristiti na nekoliko mjesta u aplikaciji, uglavnom

[40] https://medium.com/@ITesic/uvod-u-react-ekosistem-8ccfad0a1030; 23.01.2019.
[41] The Complete Redux Book; Ilya Gelman and Boris Dinkevich; str. 15.
[42] https://tighten.co/blog/react-101-using-redux, 24.01.2019.

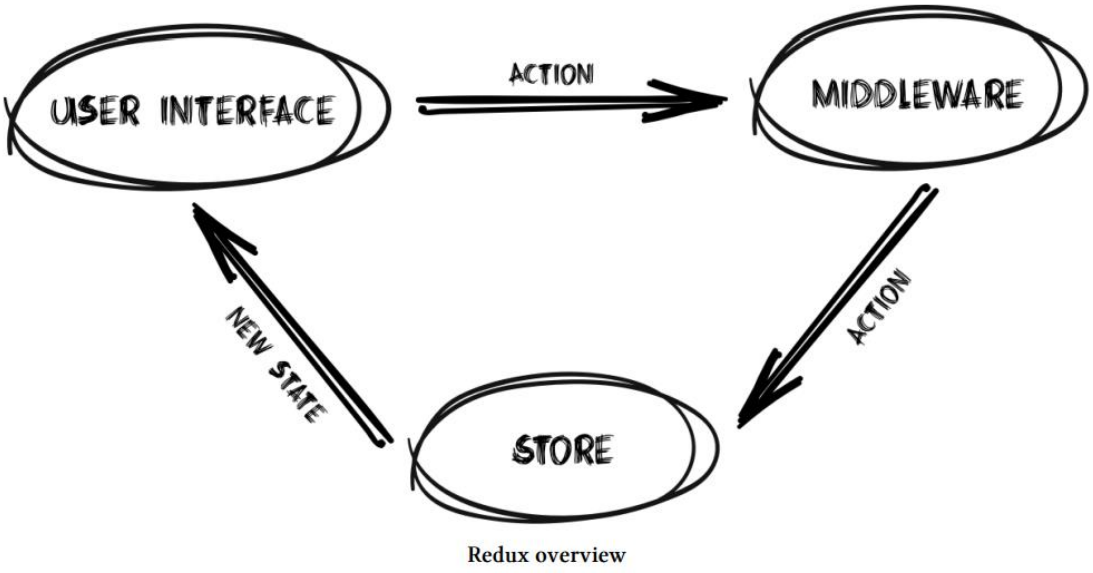
se kreira funkcija koja može generirati objekt temeljen na zadanim parametrima. Pošto te funkcije kreiraju *action* objekte, nazivaju se *action creators*. U primjeru programskog isječka (Programski isječak 14) može se vidjeti način kreiranja funkcija koje predstavljaju akcije.

```
export const getUsers = () => async dispatch => {
  const res = await axios.get("/api/users/all");
  dispatch({
    type: GET_USERS,
    payload: res.data
  });
};
```

Programski isječak 14. Akcija u "`userActions`" klasi

Kada se akcija pošalje u Store, tada je Store zadužen za promjenu stanja, a to radi tako što pozove funkciju s trenutnim stanjem i primljenom akcijom. Takva funkcija naziva se *reducer*. *Reducer* nikada ne mijenja stanje, on uvijek kreira kopiju sa željenim modifikacijama. [43] Na slici (Slika 16) može se vidjeti proces i način na koji Redux funkcionira. Od trenutka kada korisnik pošalje zahtjev s korisničkog sučelja (engl. *User Interface*), do obrade tog zahtjeva unutar Redux aplikacije, akcija (engl. *Action*) prolazi kontrolu od strane `redux-thunk` biblioteke koja je *middleware.* Svrha *middleware-a* je pregledati svaku primljenu akciju koja prolazi kroz sistem i provjeriti radi li se o funkciji, ako se radi o funkciji, pozove tu funkciju. Nakon toga skladište (engl. *Store*) šalje novo stanje akcije (engl. *New State*) korisničkom sučelju.

[43] The Complete Redux Book; Ilya Gelman and Boris Dinkevich; str. 10 - 12



Slika 16. Redux procesiranje radnji [44]

[44] Ibidem; str. 9

5. Izrada aplikacije

Web aplikacija razvijena za potrebe završnog rada je aplikacija koja olakšava i objedinjuje poslovne procese jedne bojne u MORH-u. Aplikacija ima dvije ključne funkcionalnosti: manipulacija podacima vojnika te izrada dokumenata i zapovjedi od strane zapovjednika bojne. Izrada *backend* dijela web aplikacije radila se u Java programskom jeziku i koristio se Spring Boot programski okvir. Za skladištenje podataka koristila se MySQL baza podataka, dok se za prezentacijski sloj koristio React i Redux. Prilikom izrade aplikacije koristile su se najnovije tehnologije i najnoviji programski okviri.

5.1. Podatkovni sloj

Za bazu podataka odabran je MySQL server, iako se tijekom cijelog razvoja web aplikacije koristila H2 *in memory* baza podataka. Za pristup bazi podataka korišten je Hibernate ORM jer se radilo s POJO objektima u Javi, tako da su se kreirale obične klase koje su se mapirale na tablice u bazi podataka. Na tablici (Tablica 2) ukratko je opisana svaka klasa koja predstavlja tablicu u bazi podataka.

Tablica 2. Opis podatkovnih klasa

Naziv klase/tablice	Opis
`User`	`User` klasa predstavlja svakog zaposlenika bojne i njegove osobne informacije.
`Category`	Vrsta zadaće na kojoj je korisnik angažiran.
`Project`	Svaki projekt predstavlja skup zapovjedi ili planova.
`ProjectBoard`	`ProjectBoard` je registrator u kojeg se mogu dodavati zapovjedi ili planovi.
`ProjectTasks`	Jedan `ProjectBoard` sadrži jedan projekt, dok se ti projekti sastoje od više zadataka.

Konekcijska konfiguracija sprema se u datoteku "application.properties". Na programskom isječku (Programski isječak 15) može se vidjeti koje su sve postavke potrebne za ispravno spajanje na MySQL i mapiranje POJO klasa na tablice u bazi podataka.

```
spring.jpa.show-sql=true
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/caos
spring.datasource.username=user
spring.datasource.password=secret
spring.jpa.database-
platform=org.hibernate.dialect.MySQL5Dialect
spring.jpa.properties.hibernate.dialect =
org.hibernate.dialect.MySQL57Dialect
spring.jpa.hibernate.ddl-auto=update
```

Programski isječak 15. Konfiguracija za MySQL i Hibernate

5.2. Poslovni sloj

Poslovni sloj sastoji se od repozitorija, servisa i kontrolera. Repozitorij obavlja komunikaciju sa servisnim slojem na način da naslijedi "CrudRepository" s parametrima određenih klasa iz sloja modela. Pa tako "CategoryRepository" nasljeđuje CrudRepository<Category,Long> i samim time nasljeđuje sve metode iz "CrudRepository" sučelja. Nakon što se napišu metode koje su potrebne u projektu, moguće je napraviti instancu klase "CategoryRepository", u servisnom sloju u klasi "CategoryService", te time omogućiti pristup svim metodama sučelja "CrudRepository" i metodama iz sučelja "CategoryRepository". Bitna stavka u konfiguriranju repozitorija i servisa je anotirati sve repozitorije s @Repository, i sve servisne klase sa @Service. U servisnom sloju kreiraju se klase koje obrađuju poslovnu logiku. U spomenutoj klasi "CategoryService" koja je prikazana u programskom isječku (Programski isječak 16), kreiraju se potrebne funkcije koje koriste metode iz sučelja "CategoryRepository", pa to izgleda ovako:

```
@Service
public class CategoryService {
```

```
    private final UserRepository userRepository;

    private final CategoryRepository categoryRepository;


    @Autowired
    public CategoryService(UserRepository userRepository,
CategoryRepository categoryRepository) {
        this.userRepository = userRepository;
        this.categoryRepository = categoryRepository;
    }
    public Iterable<Category>getAllCategory(){
        return categoryRepository.findAll();
    }
    public Category getCategoryById(Long id){
        return categoryRepository.getById(id);
    }}
```

Programski isječak 16. "CategoryService" klasa

Obradu poslovne logike iz servisnih funkcija obavljaju kontroleri. Oni obrađuju podatke i onda te podatke šalju prezentacijskom sloju koji ih ispisuje. Kontroleri se anotiraju s `@RestController` i `@RequestMapping`. Nakon kreiranja instance klase "CategoryService", kreiraju se metode anotirane `@PostMapping` i `@GetMapping` anotacijama, ovisno o ulozi za koju su zadužene. Naredni programski isječak (Programski isječak 17) prikazuje primjer jedne metode anotirane s `@GetMapping`:

```
@GetMapping("{id}")
public ResponseEntity<?>getCategoryById(@PathVariable Long
id){
    Category category=categoryService.getCategoryById(id);
    return new ResponseEntity<>(category,HttpStatus.OK);
}
```

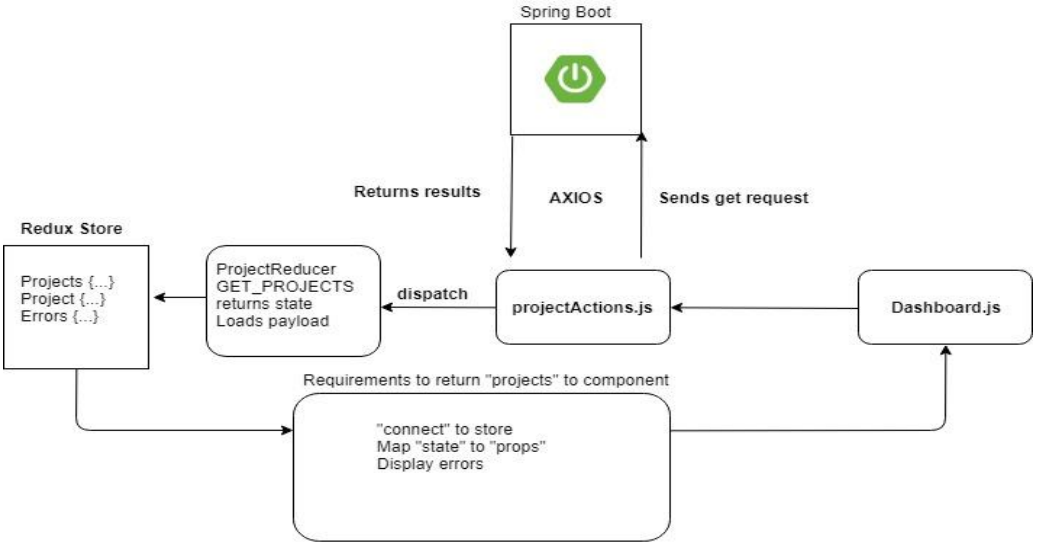
Programski isječak 17. `@GetMapping` metoda

Budući da je kontroler anotiran s `@RequestMapping("/api/category")`, a navedena metoda iznad s `@GetMapping("{id}")`, kreiran je REST API na putanji

http://localhost:8082/api/category/{id}. U fazi razvoja i testiranja poziva u *Postman-u*[45], zadavanjem parametra 1 na mjestu `{id}`, može se dohvatiti kategorija pod rednim brojem jedan. U fazi razvoja kada je riješen prezentacijski sloj koristi se navedena adresa za dohvat kategorija.

5.3. Prezentacijski sloj

Prezentacijski sloj potpuno je odvojeni projekt, zasebno razvijan koristeći React i Redux biblioteke u razvojnom alatu Visual Studio Code. Na slici (Slika 17) se može vidjeti detaljan proces od trenutka kada klijent na "`Dashboard.js`" pošalje zahtjev, pa do vraćanja odgovora i rezultata. Također je vidljivo koje sve korake je potrebno napraviti da bi se klijentu vratio rezultat.



Slika 17. Arhitektura prezentacijskog sloja[46]

U trenutku kada klijent želi dohvatiti listu projekata ili bilo koje druge podatke, bitnu ulogu odrađuje biblioteka `axios`, koja se instalira putem terminala u VS CODE-u s naredbom `npm install axios`. *Axios* je HTTP klijent za JavaScript koji se temelji na *Promise* [47] i koji se može koristiti na *frontend* aplikacijama. [48]

```
export const getProjects = () => async dispatch => {
```

[45] https://www.getpostman.com/; 27.01.2019.
[46] Izvor: Vlastiti rad autora: 28.01.2019.
[47] Promise je objekt koji može stvoriti jednu vrijednost u nekom trenutku u budućnosti
[48] https://medium.com/codingthesmartway-com-blog/getting-started-with-axios-166cb0035237; 27.01.2019.

```
  const res = await axios.get("/api/project/all");
  dispatch({
    type: GET_PROJECTS,
    payload: res.data
  });
};
```

Programski isječak 18. `getProjects` funkcija

Na prethodnom primjeru programskog isječka (Programski isječak 18) može se vidjeti korištenje biblioteke `axios` u `"projectActions.js"` klasi. Prije nego se počnu programirati funkcije koje predstavljaju akcije u `"projectActions.js"` klasi, potrebno je definirati konstante u zasebnoj klasi `"types.js"`, što izgleda ovako:

```
export const GET_PROJECTS = "GET_PROJECTS";
```

Kao što su u Javi odvojeni slojevi na modele, repozitorije, servis i kontrolere, tako se u React-u odvajaju slojevi na: *actions, reducers i components*. Idući korak je definirati *reducer-e.* Kreira se bazna klasa `"indeks.js"` prikazana u programskom isječku (Programski isječak 19) koja objedinjuje sve *reducer-e* koji će se koristiti u projektu i to izgleda ovako:

```
export default combineReducers({
errors: errorReducers,
user: userReducer,
category: categoryReducers,
project: projectReducer,
board: boardReducer
});
```

Programski isječak 19. `combineReducers` metoda

Kreirana klasa `"projectReducer.js"` predstavlja *reducer* za akcije vezane uz projekt, tako da se uvoze konstante iz `"types.js"` klase koje imaju veze s dohvatom jednog ili više projekata, ili pak brisanjem projekata. Prvotno se u svakom *reducer-u* napravi kreiranje konstante `initialState`, prikazano u programskom isječku (Programski isječak 20), gdje se postavlja State od jednog projekta na prazan objekt i State od više projekata na polja projekata:

```
const initialState = {
projects: [],
project: {}
};
```

Programski isječak 20. Primjer `initialState`

Nakon postavljanja `initialState` konstante definira se funkcija koja se označava s `export` zbog izvoza. U programskom isječku (Programski isječak 21) prikazana je funkcija koja prima State i akciju pa ovisno o tipu akcije vraća novi State i jedan ili više projekata.

```
export default function(state = initialState, action) {
  switch (action.type) {
    case GET_PROJECT:
      return {
        ...state,
        project: action.payload
      };
    case GET_PROJECTS: {
      return {
        ...state,
        projects: action.payload
      };}
    case DELETE_PROJECT:
      return {
        ...state,
        projects: state.projects.filter(
          project => project.officialNumber !==
action.payload
        )};
    default:
      return state;
  }}
```

Programski isječak 21. Primjer funkcije u "`projectReducer.js`" klasi

Nakon što su riješeni potrebni koraci prije kreiranja akcija u "`projectActions.js`" klasi, kreiraju se metode koje predstavljaju akcije. Ovisno o potrebama projekta i bazirano na API pozivima s *backend-a* na *frontend* dijelu, potrebno je kreirati iste metode. Pa se tako kreiraju metode `createNewProject`, `getProject`, `getProjects` i `deleteProject`. Primjer jedne od metoda koja dohvaća jedan projekt po identifikatoru se može vidjeti u programskom isječku (Programski isječak 22):

```
export const getProject = (id, history) => async dispatch =>
{
  try {
    const res = await axios.get(`/api/project/${id}`);
    dispatch({
      type: GET_PROJECT,
      payload: res.data
    });
  } catch (error) {
    history.push("/projectDashboard");
  }};
```

Programski isječak 22. `getProject` akcija

Akcija `getProject`, koja sadrži parametre `id` i `history`, asinkrono otprema rezultat koji je dohvaćen pomoću `axios` biblioteke i `get` poziva na `backend` servis u konstantu `res`. Ako je sve u redu, kao tip se dohvaća `GET_PROJECT`, a `payload` koji predstavlja rezultat vraća `res` varijablu s podacima. U slučaju greške, klijentu se ispisuje `error` te ga se preusmjerava na "`projectDashboard.js`" komponentu. Kao što se vidi na slici (Slika 17), nakon što "`projectReducer.js`" vrati `State` i napuni `payload` s podacima, inicijalizira se Redux State. Redux State se može gledati kao privremenu bazu podataka koja sprema sve podatke iz baze podataka. Međutim, ovisno o stranici na kojoj je klijent, pa tako u trenutku kada je klijent na "`projectDashboard.js`" stranici, *reducer* učitava samo potrebne akcije te puni `projects` i `project` iz `initialState` s objektima. Primjer spremanja podataka u Redux State se može vidjeti na slici (Slika 15). Zadnji korak koji je potreban da bi se klijentu vratio rezultat na zahtjev, spajanje je komponente na Redux State, mapiranje State-a na Props-e i prikazivanje grešaka. Spajanje na Redux State radi se pomoću komponente `connect`, koja se uvozi iz biblioteke `react-redux`. U programskom isječku (Programski isječak 23) prikazani su ključni koraci koji su potrebni u svakoj komponenti koja se želi spojiti na Redux State, mapirati State na Props-e i koristiti akcije koje su se kreirale.

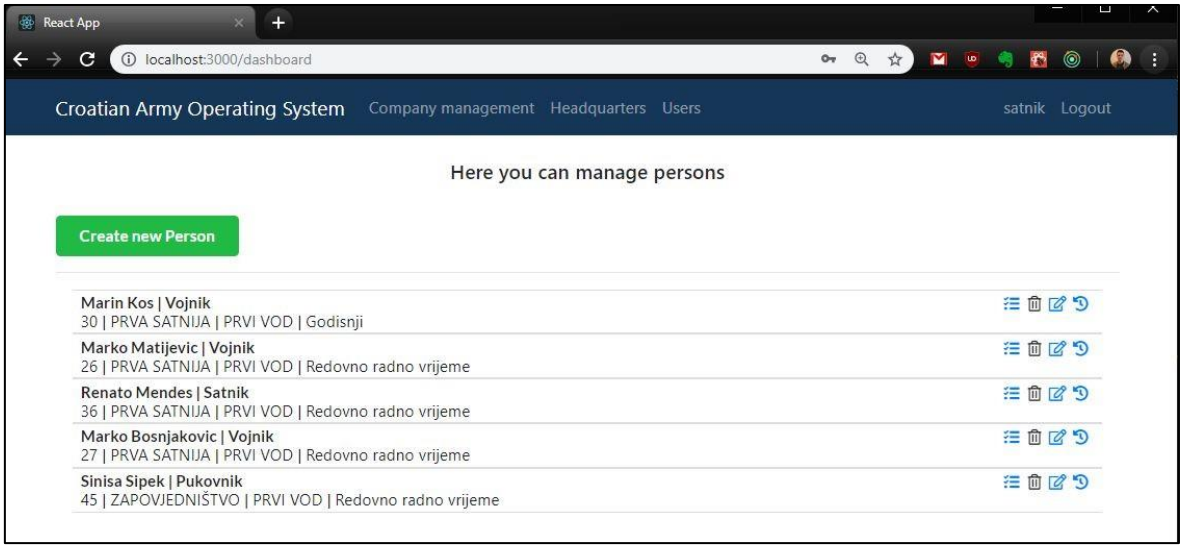
```
ProjectDashboard.propTypes = {
  project: PropTypes.object.isRequired,
  getProjects: PropTypes.func.isRequired
};
const mapStateToProps = state => ({
  project: state.project
});
```

```
export default connect(
  mapStateToProps,
  { getProjects }
)(ProjectDashboard);
```

Programski isječak 23. Spajanje na Redux State

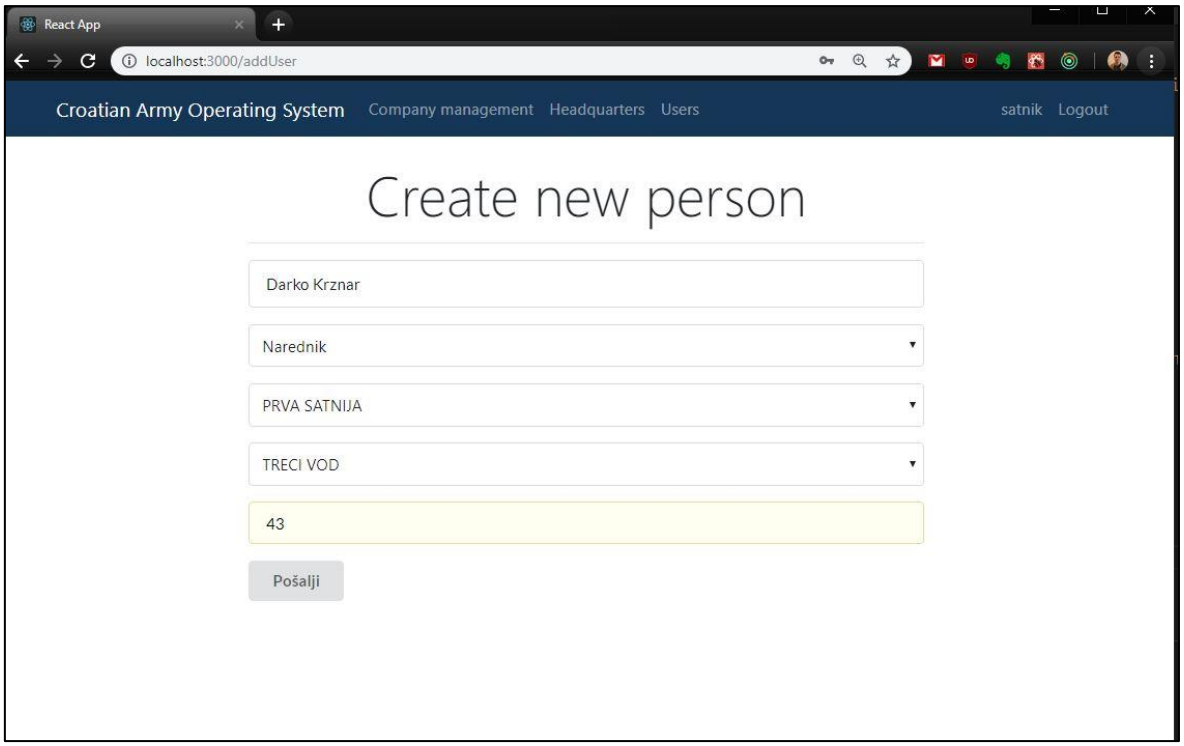
6. Korištenje aplikacije

Nakon prijave s ispravnim korisničkim imenom i lozinkom, dolazi se na početnu stranicu web aplikacije na kojoj se može vidjeti popis vojnika. Popis vojnika se sastoji od imena i prezimena, satnije u kojoj je vojnik raspoređen, vodu unutar satnije, činu i dobi određenog vojnika. Uz informacije o vojnicima navedena je i zadaća na kojoj je vojnik raspoređen. Na slici (Slika 18) se može vidjeti primjer početne stranice.

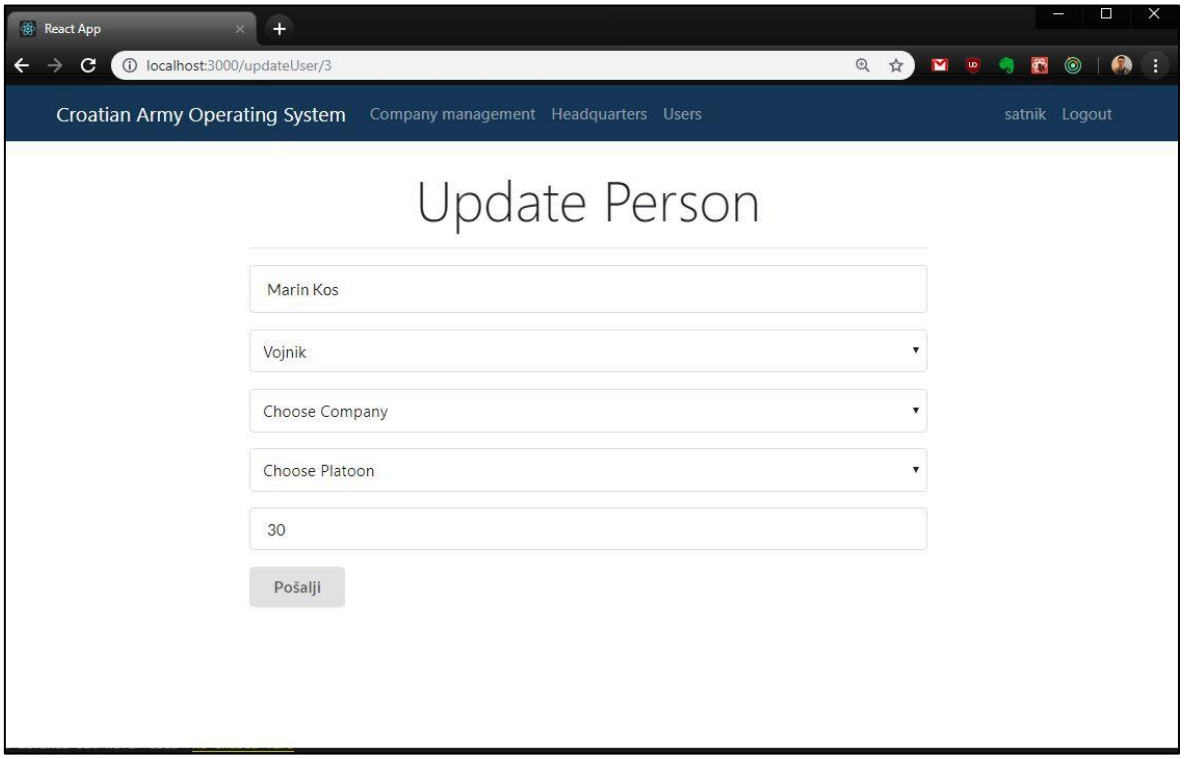


Slika 18. Početna stranica

Dodatne opcije na početnoj stranici su dodavanje novih vojnika, izmjena informacija o postojećim vojnicima i određivanje na kojoj se zadaći vojnik nalazi, te u kojem vremenskom intervalu. Primjer dodavanja novog vojnika se može vidjeti na slici (Slika 19), dok se izmjena informacija o vojniku može vidjeti na slici (Slika 20).

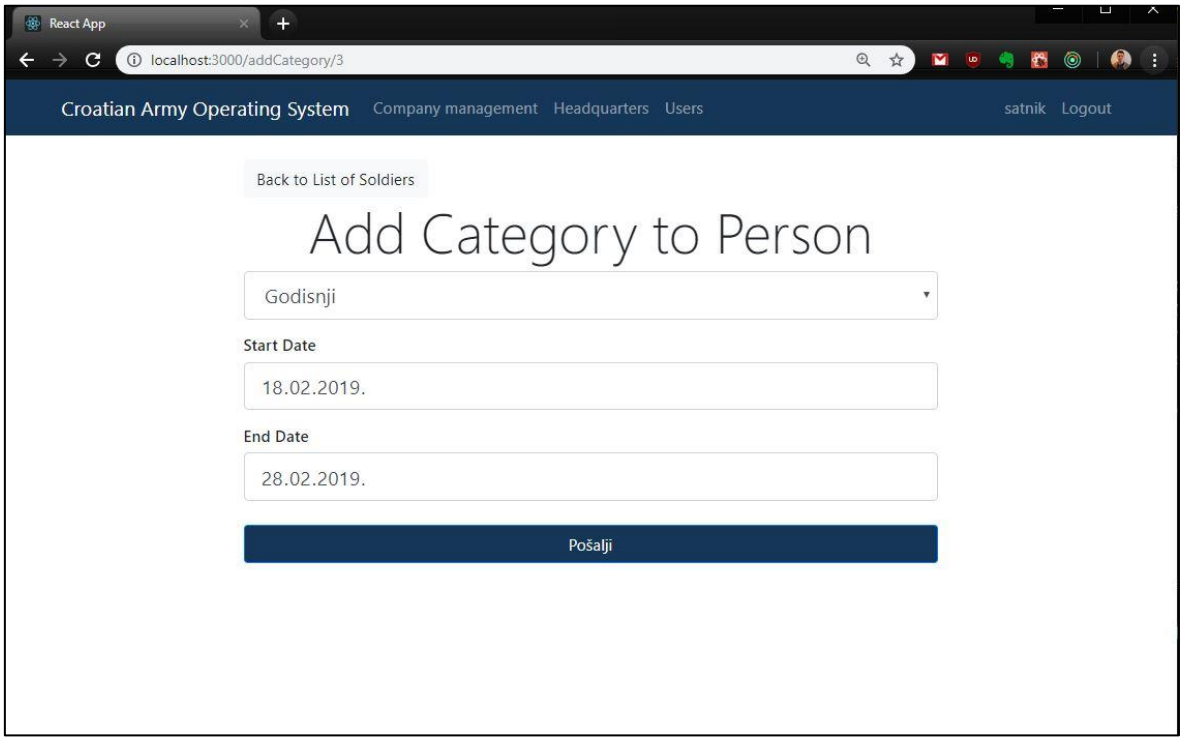


Slika 19. Dodavanje novih vojnika



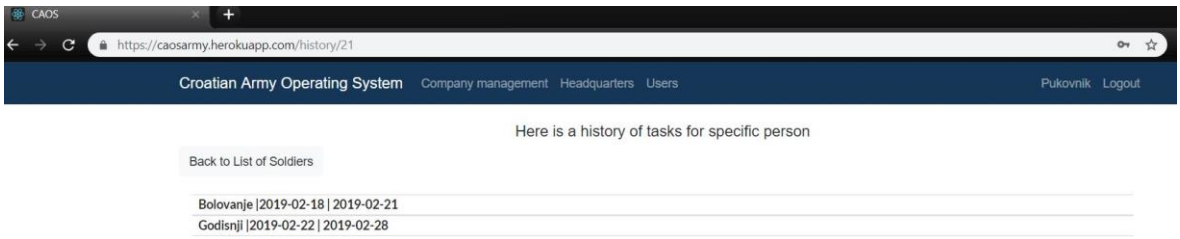
Slika 20. Izmjena postojećih vojnika

Prilikom određivanja datuma zadaće, sustav pamti interval zadane zadaće i automatski prikazuje trenutnu angažiranost vojnika. Dodavanje kategorije i vremenski interval za određenog vojnika je prikazan na slici (Slika 21).



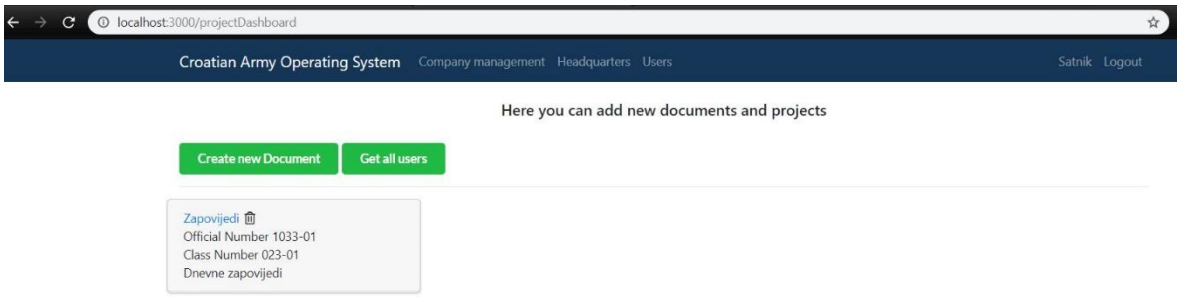
Slika 21. Određivanje zadaća za vojnike

Na slici (Slika 22) je prikazana lista angažiranosti određenog vojnika, na kojoj se može vidjeti gdje je vojnik bio angažiran u kojem razdoblju, te gdje će biti angažiran u narednom periodu. Na početnom ekranu gdje je prikazana lista vojnika i njihove informacije, kao što se može vidjeti na slici (Slika 18), zadnji gumb nas vodi na prikaz angažiranosti određenog vojnika.



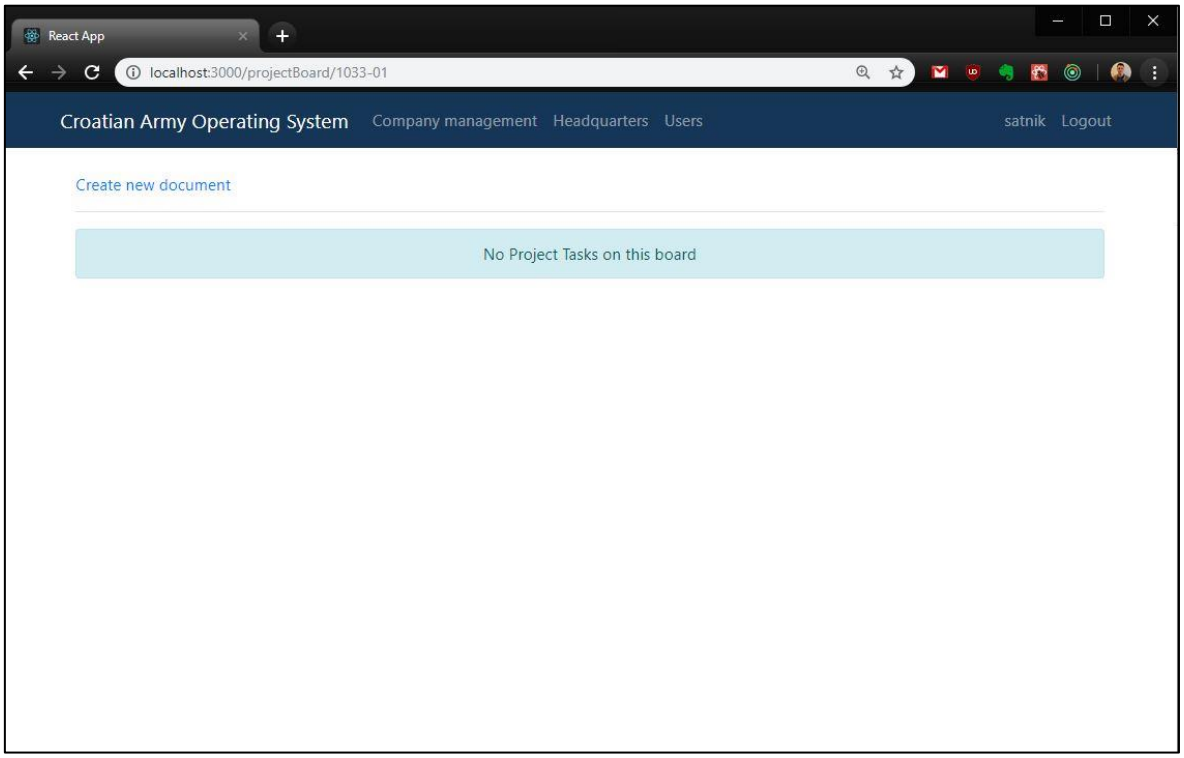
Slika 22. Povijest angažiranosti određenog vojnika

Na slici (Slika 23) je prikazana stranica na kojoj zapovjedništvo može dohvatiti popis vojnika i kreirati dokumente koji su zamišljeni kao registratori jer je jedan od važnijih poslovnih procesa unutar MORH-a kreiranje novih dokumenata, zapovijedi i planova koji se skladište u registratore. Dokument "Zapovijedi" je definiran s klasom i urudžbenim brojem, te je pri dnu kartice opisano da je riječ o dnevnim zapovijedima. Odabirom dokumenta "Zapovijedi" odlazi se na stranicu koja je prikazana na slici (Slika 24) gdje trenutno nema kreiranih dokumenata.



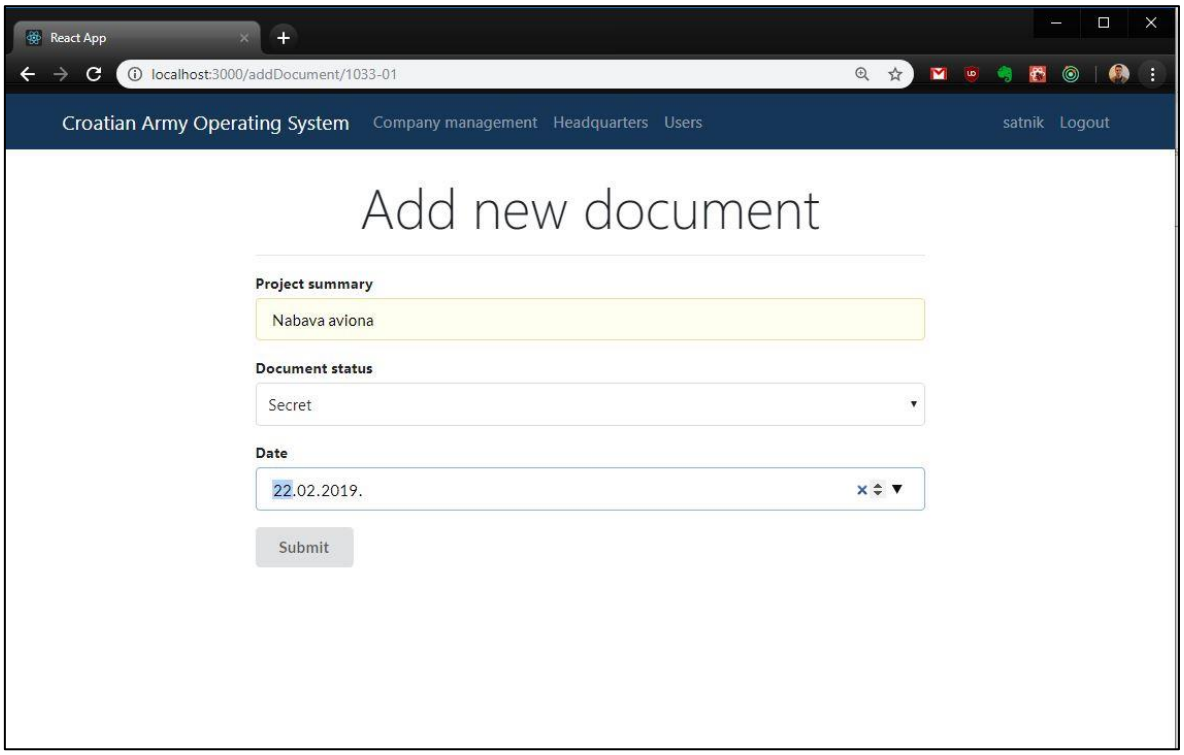
Slika 23. Popis zapovjedi i planova

Odabirom "*Create new document*" na slici (Slika 24) dohvaća se stranica za kreiranje novih dokumenata koja je prikazana na slici (Slika 25).

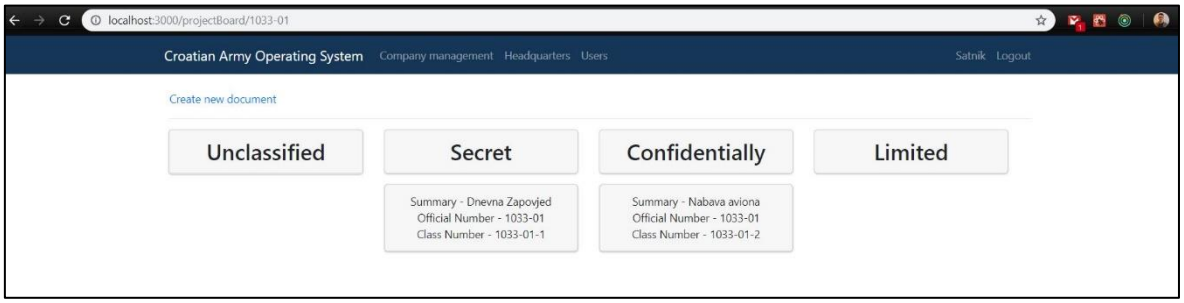


Slika 24. Prazan registrator

Stranica na kojoj se kreira novi dokument omogućuje kratki opis projekta, odabir statusa projekta i datum kreiranja projekta. Za status projekta je moguće odabrati neklasificirano (engl. *Unclassified*), tajno (engl. *Secret),* povjerljivo (engl. *Confidentially*) i ograničeno (engl. *Limited*). Sukladno odabiru statusa projekta, dokument se generira i postavlja se ispod odabranog stupnja važnosti. Primjer kreiranih dokumenata je prikazan na slici (Slika 26).



Slika 25. Dodavanje novih dokumenata



Slika 26. Popis dokumenata po stupnju važnosti

Zaključak

Tijekom pisanja završnog rada, izradom web aplikacije za unapređenje poslovanja u Hrvatskoj vojsci, pokazalo se na koji način je moguće razvijati poslovne aplikacije. Za razliku od razvijanja web aplikacija u starijim verzijama Spring programskog okvira, Spring Boot programski okvir uveo je višestruka poboljšanja. Prije se na samu konfiguraciju, bez da se napiše linija koda, trošilo dosta vremena, a Spring Boot je uvođenjem auto konfiguracije te implementiranjem početnih modula, pri samom startu ubrzao proces razvoja aplikacija. Uz Spring Boot, koji je pokrio poslovnu logiku, klijentska strana razvijala se izvan Jave. Nisu se koristili JSP i Thymeleaf koji su godinama bili, a i danas su, neizostavan dio razvoja web aplikacija u Java programskom jeziku. Nastojalo se potpuno razdvojiti slojeve razvoja radi fleksibilnosti i neovisnosti o programskim jezicima te se zbog toga prezentacijski sloj razvijao odvojeno koristeći React i Redux biblioteke.

Sama web aplikacija, koja je izrađena za potrebe završnog rada, je poslovni informacijski sustav koji je namijenjen jednoj postrojbi Hrvatske vojske. Ona nudi skladištenje svih pripadnika sa svim podacima na jednom mjestu, ali i olakšano vođenje, pa tako i ubrzano kreiranje raznih dokumenata i zapovjedi.

Popis kratica

API	*Application programming interface*	aplikativno programsko sučelje
CORS	*Cross-Origin Resource Sharing*	dijeljenje izvornih resursa
CSRF	*Cross-site-request-forgery*	krivotvorenje zahtjeva za *cross site*
HKoV	*Hrvatska kopnena vojska*	
HQL	*Hibernate Query Language*	Hibernate upitni jezik
HTML	*HyperText Markup Language*	hipertekstualni označni jezik
JAR	*Java ARchive*	Java arhiva
JDBC	*Java Database Connectivity*	Java povezivost s bazom podataka
JPA	*Java Persistence API*	*Java Persistence API*
JSON	*JavaScript Object Notation*	notacija JavaScript objekata
JSP	*Java servlet Pages*	Java servlet stranice
JSX	*JavaScript eXtension*	proširenje JavaScripta
JWT	*JSON Web Tokens*	JSON *web token*
MORH	*Ministarstvo obrane republike Hrvatske*	
MVC	*Model-View-Controller*	model-pogled-upravitelj
ORM	*Object Relational Mapping*	objektno relacijsko mapiranje
POJO	*Plain Old Java Object*	običan stari Java objekt
REST	*Representational State Transfer*	reprezentativni prijenos stanja
URI	*Uniform Resource Identifier*	jedinstveni identifikator resursa
WAR	*Web Application Resource*	resurs web aplikacije
XML	*EXtensible Markup Language*	proširivi označni jezik
XSS	*Cross-site-scripting*	križanje web lokacija

Popis slika

Popis tablica

Popis programskih isječaka

Literatura

[1] https://www.morh.hr/images/stories/_morh_2017/images/sheme/shema_morh_2017-01-23-v.jpg, Struktura morh-a, 28.12.2018.

[2] Beginning Spring Boot 2; K. Siva Prasad Reddy, 2017.

[3] https://ordina-jworks.github.io/spring/2017/06/07/Spring-IO-2017-The-Spring-ecosystem.html; Prikaz moderne web aplikacije, 14.02. 2019.

[4] Spring-Boot-Cookbook; Alex Antonov, 2015.

[5] https://danielmiessler.com/study/mvc/, Izgled MVC arhitekture, 30.12. 2018.

[6] https://urn.nsk.hr/urn:nbn:hr:143:056520, Opis REST modela, 31.12. 2018.

[7] https://dzone.com/articles/understanding-the-basics-of-spring-vs-spring-boot, Prednosti Spring Boot programskog okvira, 01.01. 2019.

[8] https://bbvaopen4u.com/en/actualidad/rest-api-what-it-and-what-are-its-advantages-project-development, Prednosti REST modela, 02.01. 2019.

[9] https://hr.wikipedia.org/wiki/MySQL, OPIS MYSQL, 03.01. 2019.

[10] https://www.mysql.com/common/images/products/MySQL_Workbench_Mainscreen_Mac.png, IZGLED MYSQL WORKBENCH-a, 03.01. 2019.

[11] https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html, POPIS METODA IZ JPAREPOSITORY-a, 04.01. 2019.

[12] https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html, POPIS METODA IZ CRUDREPOSITORY-a, 04.01. 2019.

[13] Spring-Persistence-with-Hibernate; Ahmad Seddighi, 2009

[14] http://javaprogramiranje.blogspot.com/2012/01/lista-hiberante-anotacija.html, Lista Hibernate anotacija, 06.01. 2019.

[15] https://thoughts-on-java.org/ultimate-guide-association-mappings-jpa-hibernate/, LISTA HIBERNATE ASOCIJACIJA, 06.01. 2019.

[16] Learning-Spring-Boot; Greg L. Turnquist, 2014.

[17] Hands-On Full Stack Development with Spring Boot 2.0 and React; Juha Hinkula, 2018.

[18] http://kelvinleong.github.io/authentication/2018/06/06/JWT-Authentication.html, Objašnjenje JWT autentifikacije, 13.01. 2019.

[19] https://en.wikipedia.org/wiki/JavaServer_Pages, Opis JSP stranica, 20.01. 2019.

[20] https://medium.com/@ITesic/uvod-u-react-ekosistem-8ccfad0a1030, REDUX OPIS, 23.01. 2019.

[21] The Complete Redux Book; Ilya Gelman and Boris Dinkevich, 2017.

[22] https://tighten.co/blog/react-101-using-redux, IZGLED REACT DEVTOOLS-a, 24.01. 2019.

[23] https://medium.com/codingthesmartway-com-blog/getting-started-with-axios-166cb0035237, OPIS AXIOS BIBLIOTEKE, 27.01. 2019.