



IT355 - WEB SISTEMI 2

Povezivanje mobilnih i veb Java tehnologija

Lekcija 14

PRIRUČNIK ZA STUDENTE

IT355 - WEB SISTEMI 2

Lekcija 14

POVEZIVANJE MOBILNIH I VEB JAVA TEHNOLOGIJA

- ✓ Povezivanje mobilnih i veb Java tehnologija
- ✓ Poglavlje 1: Spring i kompatibilne tehnologije i platforme
- ✓ Poglavlje 2: Alati za razvoj aplikacija različitih tehnologija
- ✓ Poglavlje 3: Razvoj backend strane primenom Spring okvira
- ✓ Poglavlje 4: Kreiranje veb klijenta za Spring backend
- ✓ Poglavlje 5: Kreiranje mobilnog klijenta za Spring backend
- ✓ Poglavlje 6: Pokazna vežba 14 (45 min)
- ✓ Poglavlje 7: Individualna vežba 14
- ✓ Poglavlje 8: Domaći zadatak 14
- ✓ Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

▼ Uvod

UVOD

Lekcija ima zadatak da ukaže na fleksibilnost Springa da se povezuje sa različitim tehnologijama.

Lekcija ima zadatak da kroz konkretan razvojni projekat ukaže na fleksibilnost *Spring* okvira da se povezuje sa različitim savremenim tehnologijama.

U ovom radu upoznaćemo se sa radom Spring okvira, koji je baziran na Java programskom jeziku, u nekom sasvim drugom svetlu. Pored Spring okvira koristiće se i ostale tehnologije kao što su *HTML*, *CSS*, *Bootstrap* i *JavaScript*, zajedno sa mobilnim tehnologijama. Što se tiče mobilne tehnologije biće napravljena *Android* aplikacija koja će komunicirati sa *Spring REST* servisima.

Aplikacija koja će biti kreirana predstavlja internu aplikaciju proizvodne kompanije koja će olakšati vođenje magacina. Administrator će moći da vidi koji zaposleni na kojoj mašini pravi određeni proizvod i u kom je statusu izrade taj proizvod. Admin korisnik može da vrši *CRUD* (*create*, *read*, *update* i *delete*) operacije nad korisnicima sistema odnosno zaposlenima, kupcima, proizvodima, porudžbinama, kategorijama proizvoda, mašinama i proizvodnjom. Obični korisnik može da upravlja samo svojom proizvodnjom.

Android aplikacija imaće mogućnost pretrage proizvodnje kao i opciju skeniranja *QR* koda koji će učitati definisani proizvod.

Savladavanjem ove lekcije studenti će zaokružiti primenu svih Java tehnologija izučavanih tokom studiranja.

Lekcija je rađena kao konkretan Spring - JS - Android projekat i na taj način će teći izlaganje. Na kraju lekcije možete preuzeti i kompletno urađen primer!!!

▼ Poglavlje 1

Spring i kompatibilne tehnologije i platforme

SPRING OKVIR - MODULI KOJI SE KORISTE U PROJEKTU

Spring okvirom je kreirana centralna komponenta projekta.

Spring okvir (Spring Framework) predstavlja Java platformu koja obezbeđuje obimnu infrastrukturnu podršku za razvoj širokog spektra Java aplikacija. Celokupnom infrastrukturom upravlja Spring pa tim za razvoj može da se fokusira isključivo na razvoj konkretnih programa. Spring omogućava kreiranje aplikacija iz POJO (Plain old Java objects) objekata i primenu složenih (enterprise) servisa na POJO objekte. Navedene funkcionalnosti Spring okvira primenjuju se u potpunosti na Java SE (Java Standard Edition) model programiranja i u potpunosti, ili parcijalno, na Java EE (Java Enterprise Edition) model programiranja. Najveću popularnost i primenu Spring je našao kao okruženje za razvoj veb aplikacija. Može se reći da za Java platformu on predstavlja nezvanični standard i potpuno rešenje za njihov razvoj, jer se lako integriše sa Java servletima a same aplikacije razvijaju se na neverovatno standardan i udoban način. Udobnost u programiranju koju nudi Spring pre svega se ogleda u mogućnostima korišćenja već predefinisane hijerarhijske organizacije aplikacije uz pomoć modula Spring MVC, deskriptivnom opisivanju komunikacije sa bazama podataka preko skupa modula Spring Data pri čemu u većini slučajeva nije potrebno opisivanje konkretnog koda za dohvaćanje podataka, relativno jednostavnom načinu osiguravanja aplikacija preko modula Spring Security itd.

Programeri Spring okvira prilično nezadovoljni postojećim načinima organizovanja aplikacija na Java platformi uvideli su šansu u pravljenju okvira koji omogućava jednostavan način za modularizaciju poslovnih aplikacija. Spring omogućava uniforman način za modularizaciju aplikacija pri čemu je proširenje aplikacije jako jednostavno. Modularizacija u Spring okviru se omogućava razdvajanjem ključnih i međumodularnih funkcionalnosti.

Svaka Spring aplikacija mora da ubaci minimalno jezgro koje omogućava rad ostalim modulima Spring - a. Jezgro Spring - a omogućava razdvajanje ključnih funkcionalnosti aplikacije. Zasnovano je na principu inverzije kontrole i sadrži aplikativni sloj koji omogućava programiranje tzv. spring zrna i rad sa njima

Konačno, navedene Spring funkcionalnosti omogućavaju laganu integraciju Springa sa ostalim veb i mobilnim tehnologijama sa ciljem razvoja još kvalitetnijih i funkcionalnostima bogatijih aplikacija.

MVC ARHITEKTURA KORIŠĆENA U PROJEKTU

Model-View-Controller (MVC) je arhitekturni šablon koji se koristi u razvoju softvera.

Model - View - Controller (MVC) je arhitekturni šablon koji se koristi u razvoju softvera. U složenim aplikacijama koje prikazuju korisniku ogromne količine podataka programeri često žele da razdvoje kod koji se bavi podacima od onog koji se bavi interfejsom, tako da razvoj oba postane lakši i jednostavniji.

MVC rešava ovaj problem razdvajanjem podataka i biznis logike od njihovog prikaza i interakcije sa korisnikom, uz to uvodeći i komponentu zaduženu za koordinisanje prve dve.

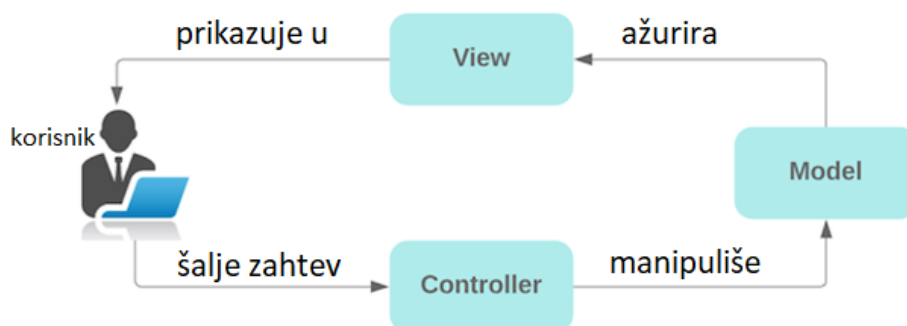
Pojam *MVC* danas ima mnogo značenja – toliko da je teško biti siguran koja je zvanična definicija MVC-a. Ono što definitivno znamo jeste da je to arhitekturni dizajn šablon koji opisuje jedan specifičan način građenja aplikacija / programa / sistema. Podrazumeva postojanje tri vrste komponenti (Model, View, Controller) koje se međusobno nalaze u posebno definisanom odnosu.

Iako ima mane (diskutovano u IT255) i iako su ga neki danas zamenili nekim drugim šablonima razvoja, MVC je kao ideja veoma važan u softverskoj zajednici, i manje više se očekuje da svako ko sebe naziva programerom zna šta ovaj pojam predstavlja i kako se implementira.

Danas, kada se u praksi koriste i MVC, ili njegovi "rođaci" MVP i MVVM, korisno je razumeti suštinu ideje o tome kako gradimo aplikacije sa grafičkim interfejsom kako bismo sa lakoćom prihvatili sve buduće varijacije na ovu temu.

Neki od najpopularnijih softverskih radnih okvira danas zasnivaju se upravo na MVC-u. **Zend, Laravel, Django i mnogi drugi oslanjaju se na ovaj način građenja aplikacija.** To pokazuje suštinu dizajn šablona — univerzalni recept za čest problem.

MVC deli sve ono što jedna aplikacija radi na tri dela. Ovakvo gledanje na funkcionalnost — takozvano razdvajanje zaduženja — smatra se i inače dobrom praksom.



Slika 1.1 MVC arhitektura od značaja za projekat [izvor: autor]

MAVEN

Maven je alat koji se koristi pri razvoju aplikacije u cilju lakše integracije sa postojećim bibliotekama klasa

Maven je alat koji se koristi pri razvoju aplikacije u cilju lakše integracije sa postojećim bibliotekama klasa. Maven se zasniva na plug-in arhitekturi, koja omogućava primenu dodataka (plugin - ova) za različite zadatke (compile, test, build, deploy, checkstyle, pmd, scp-transfer) u projektu, bez potrebe njihove direktne instalacije. Sve biblioteke, potrebne datom projektu, u daljem tekstu zavisnosti (eng. dependencies) deklarišu se u pom.xml fajlu. Maven kreira lokalni repozitorijum na disku sa potrebnim jar-ovima tako da prilikom pokretanja projekta, sve biblioteke navedene u pom.xml fajlu u početku traži lokalno, a zatim ako ih ne nađe preuzima ih sa centralnog maven repozitorijuma na internetu.

HTML I CSS

HTML i CSS sus tehnologije nezaobilazne u veb razvoju.

HTML (engl. Hyper Text Markup Language, jezik za označavanje hiperteksta) je opisni jezik specijalno namenjen opisu veb stranica. Pomoću njega se jednostavno mogu odvojiti elementi kao što su naslovi, paragrafi, citati i slično. Pored toga, u HTML standard su ugrađeni elementi koji detaljnije opisuju sam dokument kao što su kratak opis dokumenta, ključne reči, podaci o autoru i slično. Ovi podaci su opštepoznati kao meta podaci i jasno su odvojeni od sadržaja dokumenta.

CSS (engl. Cascading Style Sheets) je jezik formatiranja pomoću kog se definiše izgled elemenata veb-stranice. Prvobitno, HTML je služio da definiše kompletan izgled, strukturu i sadržaj veb-stranice, ali je od verzije 4.0 HTML-a uveden CSS koji bi definisao konkretan izgled, dok je HTML ostao u funkciji definisanja strukture i sadržaja.

JQUERY

Kod napisan u regularnom JavaScript jeziku upotrebom JQuery biblioteke je u znatno redukovanom obliku.

JQuery je popularna JavaScript biblioteka koncipirana tako da pojednostavi pisanje i izvršavanje JavaScript skripti. JQuery je projekat otvorenog koda, besplatan za preuzimanje i korišćenje, a pored osnovne biblioteke dostupan je i veliki broj besplatnih plugin-ova (jquery-mask-plugin, cropper, js-cookie, jquery-ui, itd..). Osnovna prednost jquery-a je u pojednostavljivanju operacija kao što su manipulacija DOM (eng. "document object model") elementima, rukovanje događajima, izrada animacije, kreiranje ajax poziva. Kompatibilan je sa svim verzijama popularnih veb pregledača. U pogledu kompleksnosti, JQuery je mnogo efikasniji u odnosu na standardni JavaScript. Kod napisan u regularnom JavaScript jeziku upotrebom JQuery biblioteke, može se napisati u znatno redukovanom obliku.

Postoje dve opcije korišćenja *jQuery*-a. Prva opcija podrazumeva preuzimanje biblioteke sa oficijelnog sajta, dok je druga uključivanje biblioteke u HTML stranicu preko CDN (eng. "Content Delivery Network") linka. Poslednja opcija je korištena u prototipskoj aplikaciji zato što se tim putem preuzima biblioteka sa najbližeg servera u zavisnosti od korisničke lokacije, što povoljno utiče na performanse i korisničko iskustvo. Neke od najvećih svetskih IT kompanija koriste jQuery na veb-u, kao što su: Google, Microsoft, IBM, Netflix, itd. U razvoju prototipske aplikacije jQuery je korišćen za tabelarni prikaz podataka na klijentskoj strani i izradu grafikona za prikaz cena akcija u realnom vremenu.

BOOTSTRAP

Bootstrap je kolekcija razvijenih CSS i JavaScript alata i biblioteka

Bootstrap je open-source JavaScript framework, odnosno kombinacija HTML-a, CSS-a i JavaScript-a, razvijen sa ciljem da omogući i olakša razvoj veb formi (interface-a, tj. layout-a) kao i razvoj naprednih veb komponenti. Zato ga sa te strane možemo zvati frontend framework. Razvijen je od strane Twitter-a.

Bootstrap je, dakle, kolekcija razvijenih CSS i JavaScript alata i biblioteka. Isto tako je modularnog tipa, što omogućava njegovu dalju lakšu „nadogranju“ i upotrebu sa različitim modulima koje izrađuju nezavisni developeri.

Osim što omogućava i olakšava integrisanje raznih vrsta komponenti (formi, dugmadi, akcija sa tekstom, i drugim), odlično „sarađuje“ sa JavaScript-om i sa jQuery bibliotekama. Omogućava razvoj različitih veb formi i interfejsa (layout-a): *grid*, *fixed*, *fluid* i *responsive*.

Najveća prednost Bootstrap-a je ta što u sebi ima ugrađen set alata i biblioteka za kreiranje fleksibilnih i respozivnih veb formi sa svim pripadajućim elementima.

JSP

JSP specifikacija je standardni deo Java Servlet API.

JSP stranica se može kreirati pomoću standardnih HTML/XML alata, ali treba naglasiti da je JSP specifikacija standardni deo Java Servlet API. Zbog ove činjenice u radu sa JSP stranicama iskustvo i znanje o servletima je veoma korisno. Ipak, postoje značajne razlike između ove dve tehnologije. Za razliku od servleta, gde je za programiranje potrebno značajno programersko znanje, JSP tehnologija je mnogo bliža i manje iskusnim programerima, ali i dizajnerima korisničkog interfejsa. Na ovaj način i dizajneri mogu da dobiju značajniju ulogu u toku razvoja same aplikacije. Još jedna prednost JSP tehnologije je efikasno korišćenje odvojenih nezavisnih programskih segmenata. Time do punog izražaja dolaze sve osobine objektno orijentisanog programiranja. Nezavisni programski segmenti se definišu u okviru komponenata zvanih *Java Beans*, a tehnologija koja ih koristi je *Enterprise Java Beans* tehnologija.

Da bi se pokretale JSP stranice i povezivale sa servletima potrebno ih je izvršavati na određenom veb server kao što je *Tomcat*, *GlassFish* ili slični.

Sa JSP stranicama statički deo se odvajaju od dinamičkog korišćenjem specijalnih tagova i skriptleta. Kada dizajner strane napravi bilo koju promenu, JSP stranica se automatski rekompajlira i ponovo učita na web server pomoću JSP mašine.

Piši jednom -izvršavaj bilo gde: JSP tehnologija donosi ovo pravilo i u interaktivne veb strane. JSP stranice se mogu prenositi sa jedne platforme na drugu ili promeniti veb server, ponašanje aplikacije će biti potpuno isto.

RESTFUL WEB SERVISI

Danas se RESTFull izdvojio kao dominantan mrežni servis.

Ovi servisi su jednostavnije integrisani sa HTTP-om od SOAP servisa, ne zahtevaju XML poruke ili WSDL opise servisa. **Danas se RESTFull izdvojio kao dominantan mrežni servis**, potisnuo je SOAP i WSDL jer je značajno jednostavniji za korišćenje.

Podaci se najčešće prebacuju u *JSON* formatu mada je dostupan i XML i YAML format. Zasniva se na REST arhitekturi, veoma je fleksibilan i jednostavan za razumevanje. Može biti izvršen na bilo kom klijentu ili serveru koji ima HTTP/HTTPS podršku. RESTful servisi treba da imaju sledeće osobine i karakteristike:

- Nepostojanje stanja (*Stateless*)
- Mogućnost keširanja (*Cacheable*)
- Uniformni interfejs (*Uniform interface URI*)
- Izričito korišćenje *HTTP* metoda
- Transfer *XML* i/ili *JSON*

Kod ovog tipa servisa, resursi (npr. statičke strane, fajlovi, podaci iz baze...) imaju sopstveni URL ili URI koji ih identifikuju. Pristup do resursa je definisan HTTP protokolom, gde svaki poziv čini jednu akciju (kreira, čita, menja ili briše podatke). Isti URL se koristi za sve operacije ali se menja HTTP metod koji definiše vrstu operacije. REST koristi "CRUD like" HTTP metode kao što su: *GET, POST, PUT, DELETE, OPTIONS*.

MYSQL BAZA PODATAKA

MySQL je najpopularniji i najkorišćeniji RDBMS sistem otvorenog koda.

MySQL je najpopularniji i najkorišćeniji RDBMS (eng.Relational Database Management System) sistem otvorenog koda za upravljanje relacionim bazama podataka. Programeri, administrator baza i DevOps inženjeri koriste *MySQL* da bi napravili savremene cloud-based veb aplikacije. MySQL zauzima centralni deo popularne LAMP platforme (Linux – Apache – MySQL – Perl/PHP/Python). Kao većina RDBMS rešenja MySQL je dostupan u nekoliko različitih verzija, koje se pokreću na Windows, OS X, Solaris FreeBSD i drugim varijantama Linux operativnog sistema. Mnogi od popularnih kompanija kao što su : Google, Facebook, Adobe, Zappos, YouTube koriste MySQL. Koriste ga i mnogi poznati open source projekti kao što su: WordPress, Joomla, Drupal.

Jedna od prednosti korišćenja MySQL-a je bogata API podrška (konektori) za različite programske jezike. Neki od najpoznatijih konektora su Connector/J za Java platform,

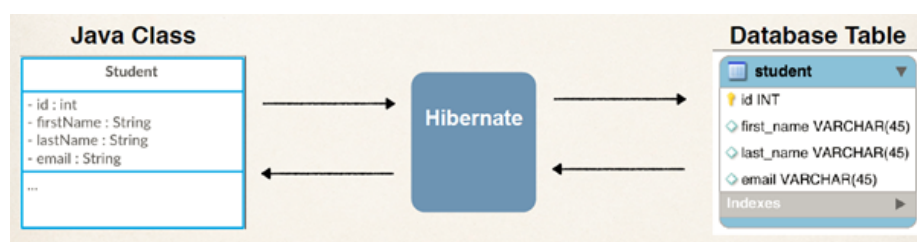
Connector/Net za .NET platformu, itd. MySQL podržava korišćenje stored procedura, okidača (trigger), kursora, funkcija, ugnježenih upita, keširanje upita. Počevši od verzije 8.0 MySQL podržava skladištenje dokumenta u vidu JSON fajlova. Osobine koje zajedno najbolje opisuju MySQL su: pouzdanost, visoka dostupnost, dobre performanse, jednostavnost korišćenja i sigurnost. Imajući u vidu gore navedene osobine MySQL baza podataka je izabrana kao tehnologija za čuvanje korisnika, njihovih kredencijala i privilegija u razvoju prototipske veb aplikacije.

HIBERNATE

Hibernate je alat za objektno-relaciono preslikavanje kod Java okruženja.

Hibernate je alat za objektno-relaciono preslikavanje kod Java okruženja. Predstavlja projekat otvorenog koda (engl. open source) koji služi za preslikavanje Java klasa u tabele relacione baze podataka i omogućava upite nad bazama podataka. Prva verzija Hibernate-a se pojavila 2001. godine. Potreba za uvođenjem Hibernate-a se javila jer postojeći alati nisu bili prilagodljivi složenim šemama podataka u aplikacijama. Njegovim autorom se smatra Gevin King.

Hibernate okvir je napisan u programskom jeziku Java, tako da se može izvršavati na svim operativnim sistemima na kojima se može izvršavati i Java (Windows, Unix, Linux i dr.). Podržava rad sa gotovo svim sistemima za upravljanje bazama podataka (DBMS) i ima ugrađene sve specifičnosti vezane za određeni sistem. Ukoliko se javi potreba za promenom DBMS-a, potrebno je samo promeniti odgovarajuće parametre u konfiguracionoj datoteci, obezbediti odgovarajući drajver i upotrebiti Hibernate za ponovno generisanje šeme baze podataka. Pri tome nije potrebno menjati aplikaciju. Uloga Hibernate-a je da bude posrednik između aplikacije i baze podataka.



Slika 1.2 Hibernate kao posrednik između aplikacije i baze podataka [izvor: autor]

Preslikavanje između objekata koji čine domenski model i tabela u bazi podataka u Hibernate-u se može definisati u XML dokumentima, programski u Java kodu ili preko anotacija. Zahvaljujući modularnosti arhitekture Hibernate-a, ovaj alat se može na različite načine koristiti prilikom razvoja softvera. Osnovna svrha Hibernate-a kao ORM alata je da se koristi za objektno-relaciono preslikavanje. Međutim, Hibernate se može koristiti i za transakcije i keširanje entiteta i upita.

ANDROID

Android je najprodavaniji operativni sistem na mobilnim uređajima.

Android je mobilni operativni sistem kompanije Gugl zasnovan na Linuks jezgri, prvenstveno dizajniran za mobilne uređaje sa ekranom osetljivim na dodir, kao što su pametni telefoni i tablet uređaji. Korisnički interfejs Androida je zasnovan na direktnoj manipulaciji objektima na ekranu, korišćenjem ulaza u vidu dodira koji odgovaraju pokretima u realnom svetu kao što su prevlačenje, pritiskanje zumiranje (engl. pinching) kao i unos teksta pomoću virtuelne tastature. Kao dodatak uređajima osetljivim na dodir, Gugl je razvio i Android TV za televizore, Android Auto za automobile i Wear OS za ručne satove, svaki od njih sa prilagođenim korisničkim interfejsom. Varijante Android operativnog sistema se koriste i na igračkim konzolama, digitalnim kamerama, personalnim računarima i drugim elektronskim uređajima.

Android je razvila istoimena kompanija (engl. Android, Inc.) koju je kompanija Gugl kupila 2005. godine. Android je predstavljen 2007. godine zajedno sa osnivanjem udruženja Open Handset Alliance (OHA) (engl. Open Handset Alliance, OHA), konzorcijuma hardverskih, softverskih i telekomunikacionih kompanija posvećenih razvoju otvorenih standarda za mobilne uređaje. Prvi Android telefon je prodat u septembru 2008. godine i od tada je predstavljeno više izdanja ovog operativnog sistema. Android aplikacije se mogu preuzeti sa Gugl Play prodavnice, na kojoj zaključno sa februarom 2017. godine, ima preko 2,7 miliona aplikacija. Android je najprodavaniji operativni sistem na mobilnim uređajima od 2013. godine, i pokreće se na većini pametnih telefona danas. Od maja 2017. godine, Android ima 2 milijarde aktivnih korisnika mesečno, i poseduje najveću bazu korisnika od svih operativnih sistema.

▼ Poglavlje 2

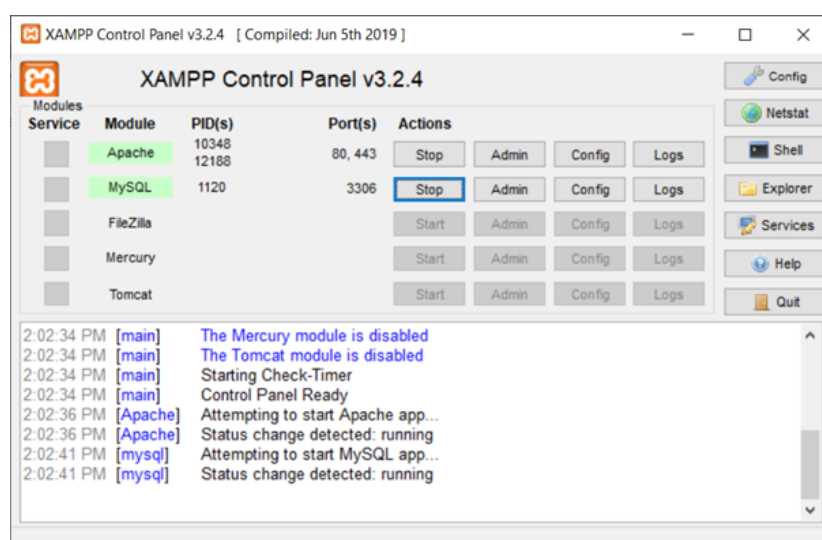
Alati za razvoj aplikacija različitih tehnologija

XAMPP

XAMPP je multiplatformska aplikacija čijom instalacijom pristup Apache veb serveru, MySQL-u itd.

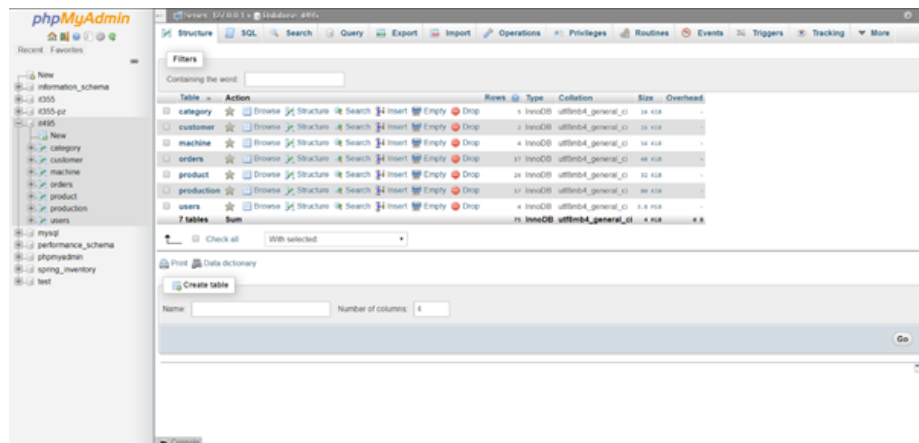
XAMPP je multiplatformska aplikacija čijom instalacijom, bez ikakvog potrebnog tehničkog predznanja, dobijate pristup Apache Web serveru, MySQL-u, PHP-u i Perl-u. Drugim rečima, u pitanju je jednostavno i brzo rešenje za brzo podešavanje minimalnog okruženja koje može da zadovolji sve potrebe po pitanju razvoja.

Kako je XAMPP multiplatformska aplikacija, može da se koristi na Linux i macOS operativnim sistemima, mada za svaki od tih sistema postoje i bolja rešenja kao što je MAMP za macOS



Slika 2.1 XAMPP Control Panel [izvor: autor]

Za čuvanje baze podataka korišćen je veb interfejs *phpMyAdmin* koji radi sa MySQL bazom podataka, alat dolazi u sklopu sa *XAMPP*-om.

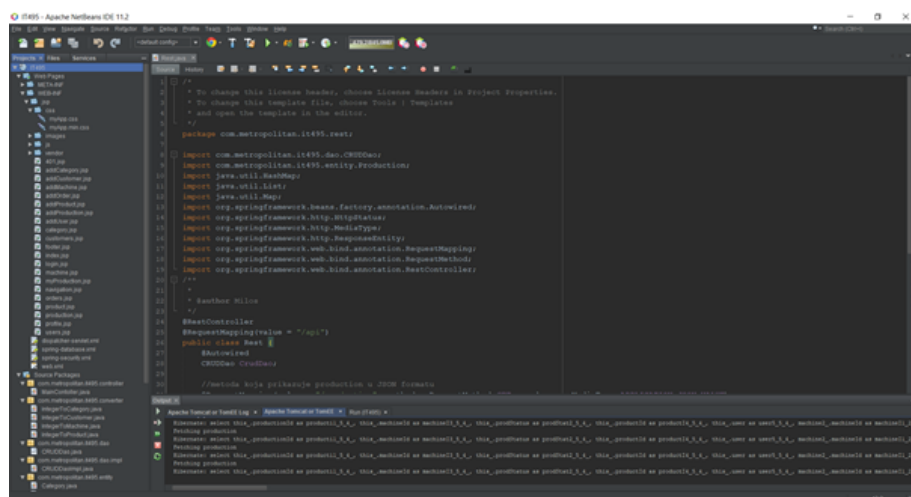


Slika 2.2 phpMyAdmin konzola [izvor: autor]

NETBEANS IDE I APACHE TOMCAT

Kao glavni server na kom se pokreće aplikacija korišćen je Apache Tomcat.

NetBeans je integrisano razvojno okruženje (IDE) prvenstveno namenjeno razvoju Java tehnologija, ali isto tako pruža dosta dodatnih mogućnosti koje mu omogućavaju da se jednako efikasno može koristiti za razvoj računarskih programa i u ostalim programskim jezicima kao što su C, C++, PHP, Fortran, Pajton, Rubi i drugi. NetBeans jednako dobro radi na različitim platformama kao što su Windows, Linux, BSD. Podržava različite tehnologije i alate koji poboljšavaju razvojni proces aplikacije.



Slika 2.3 NetBeans IDE [izvor: autor]

Kao glavni server na kom se pokreće aplikacija korišćen je Apache Tomcat.

Apache Tomcat, takođe poznat kao Tomcat Server, pokazao se popularnim izborom za veb programere koji grade i održavaju dinamične veb lokacije i aplikacije zasnovane na Java softverskoj platformi.

Razvijen od projekta Apache Software Foundation (ASF) kao open-source projekat, to je Java Servlet kontejner ili veb kontejner koji pruža proširenu funkcionalnost za interakciju sa Java

Servletima, istovremeno implementirajući nekoliko tehničkih specifikacija Java platforme: *Java Server Pages* (JSP), *Java Expression Language* (Java EL) i *WebSocket*.

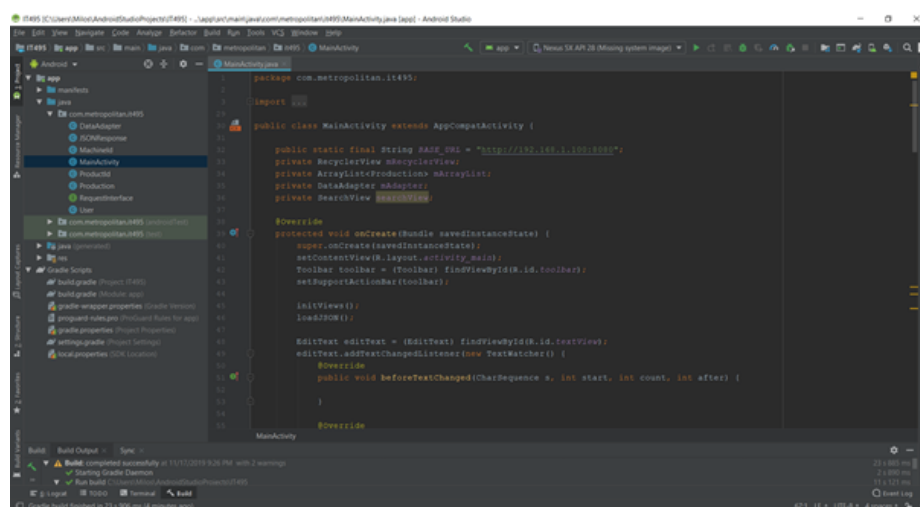
ANDROID STUDIO

Android Studio je zvanično integrisano razvojno okruženje za Google-ov Android operativni sistem.

Android Studio je zvanično integrisano razvojno okruženje (IDE) za Google-ov Android operativni sistem, izgrađen na *JetBrains*-ovom *Intelij IDEA* softveru i dizajniran posebno za Android razvoj. Dostupan je za preuzimanje na operativnim sistemima sa sistemima Windows, macOS i Linux. To je zamena za Eclipse Android Development Tools (ADT) kao primarni IDE za razvoj matičnih Android aplikacija.

Android Studio najavljen je 16. maja 2013. godine na Google I / O konferenciji. Bio je u ranoj fazi pregleda (preview) počevši od verzije 0.1 u maju 2013., a zatim je ušao u beta fazu počevši od verzije 0.8 koja je objavljena u junu 2014. Prva stabilna verzija objavljena je u decembru 2014., počevši od verzije 1.0.

Od 7. maja 2019. Kotlin je Google-ov preferirani jezik za razvoj aplikacija za Android. Ipak, drugi programski jezici su podržani, kao što su Java i C ++.



Slika 2.4 Android Studio IDE [izvor: autor]

▼ Poglavlje 3

Razvoj backend strane primenom Spring okvira

XML KONFIGURACIJA

Spring koristi Maven za preuzimanje zavisnosti tj. JAR fajlova.

Spring koristi Maven za preuzimanje zavisnosti tj. JAR fajlova. Ove zavisnosti se definišu unutar pom.xml fajla. Maven prvo definisane zavisnosti traži lokalno, ukoliko ih nema preuzima ih sa Interneta. Deo fajla je prikazan na listingu koji sledi. (Listing 1.)

```
5      <groupId>com.metropolitan</groupId>
6      <artifactId>IT495</artifactId>
7      <version>1.0-SNAPSHOT</version>
8      <packaging>war</packaging>
9
10     <name>IT495</name>
11
12     <properties>
13       <endorsed.dir>${project.build.directory}/endorsed</endorsed.dir>
14       <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15       <jstl.version>4.0.1</jstl.version>
16       <spring.version>4.3.4.RELEASE</spring.version>
17       <jackson.version>2.10.0</jackson.version>
18       <jacksonmapper.version>1.9.13</jacksonmapper.version>
19       <spring.security.version>4.2.0.RELEASE</spring.security.version>
20       <mysql.connector.version>8.0.18</mysql.connector.version>
21     </properties>
22
23     <dependencies>
24       <dependency>
25         <groupId>javax</groupId>
26         <artifactId>javaee-web-api</artifactId>
27         <version>7.0</version>
28         <scope>provided</scope>
29       </dependency>
30       <dependency>
31         <groupId>commons-codec</groupId>
32         <artifactId>commons-codec</artifactId>
33         <version>1.11</version>
34       </dependency>
35       <dependency>
36         <groupId>org.springframework</groupId>
37         <artifactId>spring-aop</artifactId>
38         <version>${spring.version}</version>
39       </dependency>
40     </dependencies>
```

Slika 3.1 Listing 1. pom.xml fajl [izvor: autor]

WEB.XML

web.xml datoteka je konfiguraciona datoteka veb aplikacija u Javi.

web.xml datoteka je konfiguraciona datoteka veb aplikacija u Javi. Upućuje kontejner servleta na klase koje treba učitati, koje parametre treba podesiti u kontekstu i kako presresti zahteve koji dolaze od pregledača. web.xml se naziva datotekom deskriptora razmeštanja. (Listing 2.)

```

24 <servlet-mapping>
25 <servlet-name>dispatcher</servlet-name>
26 <url-pattern>/</url-pattern>
27 </servlet-mapping>
28 <!-- Spring Security and File Upload -->
29 <filter>
30 <filter-name>MultipartFilter</filter-name>
31 <filter-class>org.springframework.web.multipart.support.MultipartFilter</filter-class>
32 </filter>
33 <filter>
34 <filter-name>springSecurityFilterChain</filter-name>
35 <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
36 </filter>
37 <filter-mapping>
38 <filter-name>MultipartFilter</filter-name>
39 <url-pattern>/*</url-pattern>
40 </filter-mapping>
41 <filter-mapping>
42 <filter-name>springSecurityFilterChain</filter-name>
43 <url-pattern>/*</url-pattern>
44 </filter-mapping>
45 <context-param>
46 <param-name>contextConfigLocation</param-name>
47 <param-value>
48 /WEB-INF/spring-security.xml,
49 /WEB-INF/spring-database.xml
50 </param-value>
51 </context-param>
52 <listener>
53 <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
54 </listener>

```

Slika 3.2 Listing 2. web.xml fajl [izvor: autor]

DISPATCHER-SERVLET.XML

U ovom fajlu vidimo da su mapirane putanje na kojima se nalaze slike, CSS i JavaScript fajlovi

Objekti koji čine okosnicu aplikacije i kojima upravlja Spring IoC kontejner nazivaju se zrna (engl. *beans*). Zrno je objekat koji se instancira, sastavlja i upravlja od strane Spring IoC kontejnera. Ova zrna su kreirana pomoću metapodataka konfiguracije koje definišemo kontejneru. Na primer, u obliku XML `<bean />` definicija kao što može da se vidi na sledećem listingu. (Listing 3.)

U ovom fajlu možemo da vidimo da su mapirane putanje na kojima se nalaze slike, CSS i JavaScript fajlovi. Kreirano je zrno za definisanje pogleda odnosno lokacija JSP strana. Konfigurisan je Hibernate. Kao i lokacija paketa u kom se nalaze entiteti klasa o kojima će kasnije biti govora.

```

14 <context:component-scan base-package="com.metropolitan.it495" />
15 <mvc:annotation-driven />
16 <mvc:default-servlet-handler />
17 <mvc:resources mapping="/images/*" location="/WEB-INF/jsp/images/" />
18 <mvc:resources mapping="/css/*" location="/WEB-INF/jsp/css/" />
19 <mvc:resources mapping="/js/*" location="/WEB-INF/jsp/js/" />
20 <mvc:resources mapping="/vendor/*" location="/WEB-INF/jsp/vendor/" />
21
22 <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
23     <property name="prefix" value="/WEB-INF/jsp/" />
24     <property name="suffix" value=".jsp" />
25 </bean>
26
27 <bean id="localeResolver" class="org.springframework.web.servlet.i18n.SessionLocaleResolver">
28     <property name="defaultLocale" value="sr" />
29 </bean>
30
31 <!--Kako bi Hibernate mogao da se poveže na bazu podataka potrebno je kreirati sesiju. -->
32 <bean id="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
33     <property name="configLocation">
34         <!-- Dodavanje path-a na kome se nalaze podešavanja za Hibernate -->
35         <value>classpath:hibernate.cfg.xml</value>
36     </property>
37     <!-- Dodavanje path-a paketa u kome se nalaze entity fajlovi na osnovu kojih
38     će Hibernate kreirati bazu i upisivati u nju-->
39     <property name="packagesToScan">
40         <list>
41             <value>com.metropolitan.it495.entity</value>
42         </list>
43     </property>
44 </bean>

```

Slika 3.3 Listing 3. dispatcher-servlet.xml [izvor: autor]

SPRING-SECURITY.XML

Sigurnost veb aplikacije podrazumeva definisanje posebnih sigurnosnih mehanizama.

Savremeni razvoj veb aplikacija ima mnogo izazova, a jedan od tih izazova je svakako sigurnost veb aplikacije, koja se često u korisničkim i razvojnim krugovima naglašava kao komponenta od esencijalnog značaja. Dobro dizajniran sigurnosni sistem trebao bi da obezbedi:

1. *Poverljivost* – dostupnost resursa samo ovlašćenim korisnicima,
2. *Integritet* – podrazumeva konzistentnost podataka, nemogućnost manipulacijom podataka od strane neovlašćenih lica,
3. *Dostupnost* – dostupnost resursa ovlašćenim korisnicima u svakom trenutku,
4. *Upotreba sistema isključivo od strane ovlašćenih korisnika.*

Sigurnost veb aplikacije podrazumeva definisanje posebnih sigurnosnih mehanizama kako same aplikacije, tako i njenog okruženja, da bi se obezbedile navedene stavke. Dva centralna sigurnosna koncepta, koje implementira sama veb aplikacija su *autentifikacija* i *autorizacija*. Autentifikacija je proces utvrđivanja identiteta korisnika, dok je autorizacija provera da li korisnik ima pristup za izvršavanje specifične akcije.

Spring Security obezbeđuje sveobuhvatno rešenje bezbednosti za Java EE poslovne aplikacije. Iako obezbeđuje odličnu podršku za aplikacije zasnovane na Spring-u, Spring Security može da se integriše i sa drugim radnim okvirima.

U ovom fajlu se definiše SQL upit za selektovanje korisničkog imena, lozinke i uloge (engl. role). Definiše se login stranica, stranica koja se otvara nakon uspešnog logina, stranica koja se prikazuje ukoliko korisnik nema dozvoljen pristup i logout stranica.

Takođe se definišu URL adrese svih stranica i dodeljuju se privilegije. Postoje dve vrste korisnika, a to su **USER** i **ADMIN**. U zavisnosti od uloge korisniku se dodeljuje vidljivost stranicama i omogućava pristup.

```

22 <intercept-url pattern="/product" access="hasRole('ROLE_ADMIN')" />
23 <intercept-url pattern="/addProduct" access="hasRole('ROLE_ADMIN')" />
24 <intercept-url pattern="/editProduct" access="hasRole('ROLE_ADMIN')" />
25 <intercept-url pattern="/deleteProduct/{productId}" access="hasRole('ROLE_ADMIN')" />
26 <intercept-url pattern="/orders" access="hasRole('ROLE_ADMIN')" />
27 <intercept-url pattern="/addOrder" access="hasRole('ROLE_ADMIN')" />
28 <intercept-url pattern="/deleteOrder/{orderId}" access="hasRole('ROLE_ADMIN')" />
29 <intercept-url pattern="/machine" access="hasRole('ROLE_ADMIN')" />
30 <intercept-url pattern="/addMachine" access="hasRole('ROLE_ADMIN')" />
31 <intercept-url pattern="/editMachine" access="hasRole('ROLE_ADMIN')" />
32 <intercept-url pattern="/deleteMachine/{machineId}" access="hasRole('ROLE_ADMIN')" />
33 <intercept-url pattern="/production" access="hasRole('ROLE_ADMIN')" />
34 <intercept-url pattern="/myProduction" access="hasRole('ROLE_USER')" />
35 <intercept-url pattern="/addProduction" access="hasRole('ROLE_USER')" />
36 <intercept-url pattern="/editProduction" access="hasRole('ROLE_USER')" />
37 <intercept-url pattern="/deleteProduction/{productionId}" access="hasAnyRole('ROLE_USER', 'ROLE_ADMIN')" />
38 <intercept-url pattern="/profile" access="hasAnyRole('ROLE_USER', 'ROLE_ADMIN')" />
39
40 <access-denied-handler error-page="/401" />
41 <form-login login-processing-url="/j_spring_security_check"
42 login-page="/"
43 default-target-url="/index"
44 authentication-failure-url="/error"
45 username-parameter="username"
46 password-parameter="password"/>
47 <logout logout-url="/j_spring_security_logout" logout-success-url="/logout" />
48
49 </http>
50 <authentication-manager>
51 <authentication-provider>
52 <password-encoder hash="sha-256"/>
53 <jdbc-user-service data-source-ref="dataSource"
54 users-by-username-query="select username, password, enabled from users where username=?"
55 authorities-by-username-query="select username, role from users where username=?" />
56 </authentication-provider>
57 </authentication-manager>
58 </beans>

```

Slika 3.4 Listing 5. spring-security.xml fajl [izvor: autor]

DODATNO O LOZINKAMA

U ovom konfiguracionom fajlu vidimo da se koristi enkripcija lozinke.

U ovom konfiguracionom fajlu vidimo da se koristi enkripcija lozinke. U nastavku je prikazan deo koda koji pre unosa u bazu podataka vrši enkripciju lozinke.

```

String sha256hex = DigestUtils.sha256Hex(p.getPassword());
p.setPassword(sha256hex);
crudDao.addUser(p);
model.addObject("successMsg", "Account successfully created.");

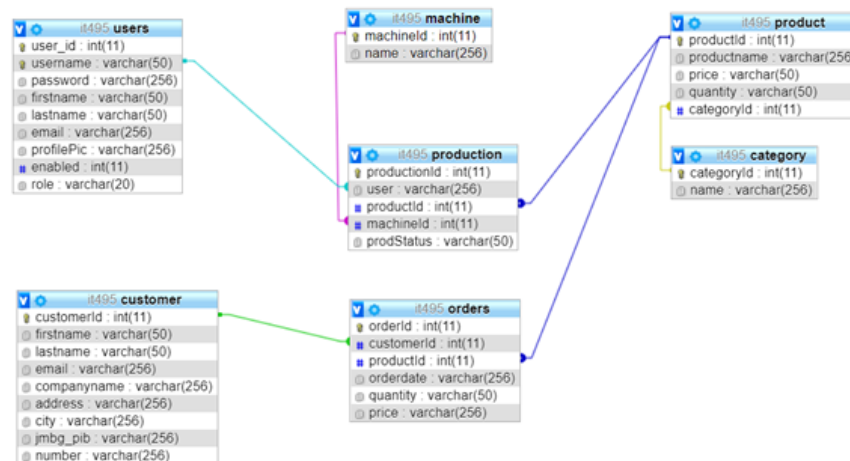
```

Slika 3.5 Listing 6. Enkripcija lozinke [izvor: autor]

HIBERNATE.CFG.XML

U Hibernate fajlu definiše se adresa baze podataka, username i password za loginovanje

U Hibernate fajlu definiše se adresa baze podataka, username i password za loginovanje. Takođe se podešava MySQL konektor.



Slika 3.6 Listing 7. hibernate.cfg.xml fajl [izvor: autor]

KLASE ENTITETI

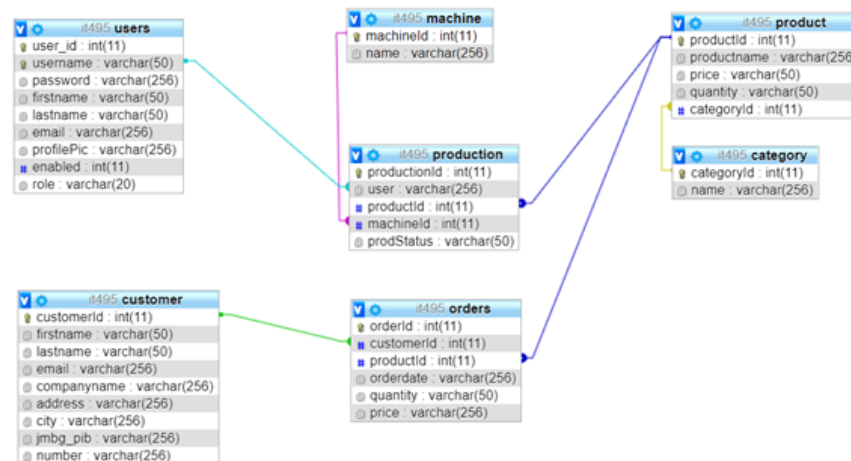
Entiteti klasa ili modeli su klase koje predstavljaju tabele iz baze podataka.

Entitetske klase ili modeli su klase koje predstavljaju tabele iz baze podataka. Mogu automatski da se generišu preko NetBeans-a.

Bitno je napomenuti sledeće:

1. Svi modeli moraju biti označeni sa `@Entity` anotacijom, koja služi da markira klasu kao persistent Java klasu, odnosno kao klasu koja se mapira u tabelu u bazi.
2. `@Table` anotacija se koristi za konfigurisanje detalja tabele u koju će biti mapirana označena java klasa.
3. `@Id` anotacija se koristi da definiše primarni ključ.
4. `@GeneratedValue` anotacija se koristi za konfiguraciju načina generisanja primarnog ključa. Odnosno prilikom kreiranja novog reda u tabeli, kolona ID će biti automatski popunjena različitim vrednostima za svaki novi red u tabeli.
5. `@Column` anotacija označava kolonu u tabeli, ovde se definiše naziv kolone.
6. `@ManyToOne` anotacija služi da se označi više prema jedan veza između entiteta.

Sve klase imaju getter, setter i toString() metode. U nastavku je prikazan dijagram baze podataka, za svaku tabelu iz dijagrama biće kreirana entitetska klasa.



Slika 3.7 Šema baze podataka za kreiranje entiteta [izvor: autor]

```

25 @Entity
26 @Table(name = "users")
27 public class Users implements Serializable {
28     private static final long serialVersionUID = 1L;
29     @Id
30     @GeneratedValue(strategy = GenerationType.IDENTITY)
31     @Column(name = "user_id")
32     private Integer userId;
33     @Column(name = "password")
34     private String password;
35     @Column(name = "first_name")
36     private String firstName;
37     @Column(name = "last_name")
38     private String lastName;
39     @Column(name = "email")
40     private String email;
41     @Column(name = "profile_pic")
42     private String profilePic;
43     @Column(name = "enabled")
44     private Boolean enabled;
45     @Column(name = "role")
46     private String role;
47     private static final long serialVersionUID = 1L;
48     @Id
49     @GeneratedValue(strategy = GenerationType.IDENTITY)
50     @Column(name = "user_id")
51     private Integer userId;
52     @Column(name = "password")
53     private String password;
54     @Column(name = "first_name")
55     private String firstName;
56     @Column(name = "last_name")
57     private String lastName;
58     @Column(name = "email")
59     private String email;
60     @Column(name = "profile_pic")
61     private String profilePic;
62     @Column(name = "enabled")
63     private Boolean enabled;
64     @Column(name = "role")
65     private String role;
66 }
  
```

Slika 3.8 Listing jednog entiteta [izvor: autor]

DAO (DATA ACCESS OBJECT)

DAO je strukturalni obrazac koji nam omogućava da izolujemo aplikacioni i poslovni sloj od perzistentnog sloja

DAO je strukturalni obrazac koji nam omogućava da izolujemo aplikacioni i poslovni sloj od perzistentnog sloja (obično je to relaciona baza podataka, ali to može biti bilo koji drugi mehanizam) koristeći apstraktni API.

Funkcionalnost ovog API-ja je da od aplikacije sakrije sve složenosti koje su uključene u obavljanje CRUD operacija u mehanizmu memorije. To dozvoljava da se oba sloja razvijaju odvojeno, a da ne znaju ništa jedan o drugom.

U projektu je kreiran interfejs *CRUDDao* koji definiše metode koje će se koristiti za rad nad bazom podataka. (sledeći listing)

```
22 public interface CRUDDao {
23
24     public Users findUserByUsername(String username);
25
26     public Users addUser(Users user);
27
28     public void deleteUser(Users user);
29
30     public void editUser(Users user);
31
32     public Object getUsers();
33
34     public Users getUserById(int userId);
35
36     public List<Customer> getCustomers();
37
38     public Customer getCustomerById(int id);
39
40     public void editCustomer(Customer customer);
41
42     public Customer addCustomer(Customer customer);
43
44     public void deleteCustomer(Customer customer);
45
46     public List<Category> getCategory();
47
48     public Category getCategoryById(int id);
49
50     public void editCategory(Category category);
51
52     public Category addCategory(Category category);
53
54     public void deleteCategory(Category category);
55
56     public List<Product> getProduct();
57
58     public Product getProductById(int id);
59 }
```

Slika 3.9 Interfejs CRUDDao [izvor: autor]

DAO (DATA ACCESS OBJECT) - IMLEMENTACIJA

Metode iz CRUDDao interfejsa su implementirane u klasi CRUDDaoImpl.

Metode iz [CRUDDao](#) interfejsa su implementirane u klasi [CRUDDaoImpl](#). U ovim metodama su definisane Hibernate transakcije.

[SessionFactory](#) je interfejs. [SessionFactory](#) može se kreirati pružanjem konfiguracionog objekta koji će sadržati sve detalje vezane za bazu podataka koji se izvlače ili iz datoteke hibernate.cfg.xml ili iz datoteke [hibernate.properties](#).

Možemo stvoriti jednu [SessionFactory](#) implementaciju po bazi podataka u bilo kojoj aplikaciji. Ako se aplikacija komunicira sa više baza podataka potrebno je kreirati po jedan [SessionFactory](#) za svaku bazu.

[SessionFactory](#) se obično stvara tokom pokretanja aplikacije i čuva za kasniju upotrebu. [SessionFactory](#) je bezbedan objekat koji se koristi u svim nitima aplikacije.

```
33 // instanciranje sesije
34 @Autowired
35 private SessionFactory sessionFactory;
36
37 // kreiranje setera
38 @Autowired
39 public void setSessionFactory(SessionFactory sessionFactory) {
40     this.sessionFactory = sessionFactory;
41 }
42 // kreiranje getera
43
44 public Session getSession() {
45     return sessionFactory.getCurrentSession();
46 }
```

Slika 3.10 SessionFactory konfiguracija [izvor: autor]

Jedna od ključnih tačaka anotacije `@Transactional` je da treba razmotriti dva odvojena koncepta, svaki sa sopstvenim dometom i životnim ciklusom:

- kontekst perzistencije
- transakcija baze podataka

Transakciona anotacija definiše opseg jedne transakcije baze podataka. Transakcija baze podataka događa se unutar konteksta perzistencije.

Kontekst perzistencije u JPA je EntityManager, implementiran interno koristeći *Hibernate* sesiju.

Kontekst perzistencije je samo sinhronizatorski objekat koji prati stanje ograničenog skupa Java objekata i osigurava da se promene na tim objektima na kraju zadrže u bazi podataka.

TRANSAKCIONE DAO METODE - IMPLEMENTACIJA

Kriterijumi se mapiraju preko entiteta uz dodavanje potrebne metode.

Kao što možemo da vidimo na listingu koji sledi ne koriste se klasični SQL upiti. Kod je dosta jednostavniji. Kriterijumi se mapiraju preko entiteta uz dodavanje potrebne metode mogu se izvršiti potrebne transakcije kao što su upis, brisanje, selektovanje i ažuriranje.

```

81  @Override
82  @Transactional
83  public Users getUserById(int userId) {
84      Users users = (Users) getSession().createCriteria(Users.class).add(Restrictions.eq("userId", userId)).uniqueResult();
85      return users;
86  }
87
88  @Override
89  @Transactional
90  public List<Customer> getCustomers() {
91      List<Customer> results = (List<Customer>) getSession().createCriteria(Customer.class).list();
92      return results;
93  }
94
95  @Override
96  @Transactional
97  public Customer getCustomerById(int id) {
98      Customer customer = (Customer) getSession().createCriteria(Customer.class).add(Restrictions.eq("customerId", id)).uniqueResult();
99      return customer;
100  }
101
102  @Override
103  @Transactional
104  public void editCustomer(Customer customer) {
105      getSession().saveOrUpdate(customer);
106  }
107
108  @Override
109  @Transactional
110  public Customer addCustomer(Customer customer) {
111      return (Customer) getSession().merge(customer);
112  }
113
114  @Transactional
115  @Override
116  public void deleteCustomer(Customer customer) {
117      getSession().delete(customer);

```

Slika 3.11 Transakcije DAO metode [izvor: autor]

KONVERTERI

Potrebno je izvršiti određene konverzije kako bi upis u bazu bio korektan

Pošto se u projektu koristi relaciona baza podataka koja ima vezane tabele stranim ključem potrebno je izvršiti određene konverzije kako bi upis u bazu bio korektan.

U entitetu *Order* možemo da vidimo spojene tabele sa objektima klasa.

```

49  @JoinColumn(name = "customerId", referencedColumnName = "customerId")
50  @ManyToOne(optional = false)
51  private Customer customerId;
52  @JoinColumn(name = "productId", referencedColumnName = "productId")
53  @ManyToOne(optional = false)
54  private Product productId;

```

Slika 3.12 Spojene tabele sa objektima klasa [izvor: autor]

U tabeli *order* se upisuje integer, a to nije isti tip kao npr. *Customer* objekat. Da bi tabela *customer* mogla pravilno da prepozna i poveže elemente iz definisanih tabela moramo da izvršimo konverziju.

```

17 public class IntegerToCategory implements Converter<String, Category> {
18     @Autowired
19     CRUDDao crudDao;
20
21     @Override
22     public Category convert(String s) {
23         if (s.isEmpty()) {
24             return null;
25         }
26         Integer value = Integer.valueOf(s);
27         System.out.println("Konvertujem u category");
28         Category c = crudDao.getCategoryById(value);
29         return c;
30     }
31 }

```

Slika 3.13 Konverzija [izvor: autor]

GLAVNI KONTROLER

U Spring okviru svi zahtevi koje šalje dispečerski servlet obično su usmereni na klasu kontrolera.

U Spring okviru svi zahtevi koje šalje dispečerski servlet obično su usmereni na klasu kontrolera. Ova klasa kontrolera preslikava te zahteve u procese i izvršava tražene unose. U projektu može biti više kontrolora definisanih za različite svrhe. Svi ovi kontroleri odnose se na isti dispečer servlet. `@RequestMapping` se koristi za mapiranje dispečer servleta sa klasom kontrolera.

U mapiranju kontrolera postoje dve vrste mapiranja, a to su `GET` i `POST`. Obično u kontroleru može biti mnogo `GET` metoda dok se koristi jedna `POST` metoda. `GET` metoda se koristi za traženje zahteva od korisnika da izvrše željene rezultate rada i vrši se prikaz na `JSP` stranice. `GET` zahtev je prikazan u nastavku.

```

68 @RequestMapping(value = "/", method = RequestMethod.GET)
69 public ModelAndView defaultPage(@RequestParam(value = "error", required = false) String error) {
70     ModelAndView model = new ModelAndView();
71     if (error != null) {
72         model.addObject("error", "Invalid credentials!");
73     }
74     model.setViewName("login");
75     return model;
76 }

```

Slika 3.14 GET zahtev [izvor: autor]

`POST` se koristi za preuzimanje podataka, obično su u pitanju forme.

```

132 @RequestMapping(value = "/addUser", method = RequestMethod.POST)
133 public ModelAndView addUser(@ModelAttribute("user") Users p, ModelAndView model) {
134     Users exist = crudDao.findUserByUsername(p.getUsername());
135     if (p.getFirstname().isEmpty()) {
136         model.addObject("error", "First name cannot be empty!");
137     } else if (p.getLastname().isEmpty()) {
138         model.addObject("error", "Last name cannot be empty!");
139     } else if (p.getEmail().isEmpty()) {
140         model.addObject("error", "Email cannot be empty!");
141     } else if (p.getUsername().isEmpty()) {
142         model.addObject("error", "Username cannot be empty!");
143     } else if (exist != null) {
144         model.addObject("error", "Username already exists!");
145     } else if (p.getPassword().isEmpty()) {
146         model.addObject("error", "Password cannot be empty!");
147     } else {
148         String sha256hex = DigestUtils.sha256Hex(p.getPassword());
149         p.setPassword(sha256hex);
150         crudDao.addUser(p);
151         model.addObject("successMsg", "Account successfully created.");
152     }
153     return model;
154 }

```

Slika 3.15 POST zahtev [izvor: autor]

Anotacija `@ControllerAdvice` se pokreće prilikom svake `@RequestMapping` metode. Konkretno u ovu anotaciju je implementiran kod za preuzimanje lokacije profilne slike korisnika. Kako bi se slika prikazivala na svakoj stranici u navigaciji koristi se ova anotacija.

```

53 @ControllerAdvice
54 public class ProfilePicControllerAdvice {
55     @ModelAttribute
56     public void addProfilePicToModel(Model model) {
57         Authentication auth = SecurityContextHolder.getContext().getAuthentication();
58
59         if (!"anonymousUser".equals(auth.getName())) {
60             String username = auth.getName();
61
62             Users u = crudDao.findUserByUsername(username);
63             model.addAttribute("profilePic", u.getProfilePic());
64         }
65     }
66 }

```

Slika 3.16 ControllerAdvice anotacija [izvor: autor]

REST KONTROLER

Aplikacija koristi REST servise uz pomoć kojih komunicira sa mobilnim klijentom

Aplikacija koristi REST servise uz pomoć kojih komunicira sa mobilnim klijentom. Za prikaz podataka koristi se JSON format.

`@RestController` je specijalizovana verzija kontrolera. Uključuje anotacije `@Controller` i `@ResponseBody` kao rezultat pojednostavljuje implementaciju kontrolera.

```

53 @ControllerAdvice
54 public class ProfilePicControllerAdvice {
55     @ModelAttribute
56     public void addProfilePicToModel(Model model) {
57         Authentication auth = SecurityContextHolder.getContext().getAuthentication();
58
59         if (!"anonymousUser".equals(auth.getName())) {
60             String username = auth.getName();
61
62             Users u = crudDao.findUserByUsername(username);
63             model.addAttribute("profilePic", u.getProfilePic());
64         }
65     }
66 }

```


Slika 3.17 REST kontroler [izvor: autor]

Kontroler je označen sa anotacijom `@RestController`, stoga `@ResponseBody` nije potreban.

Svaka metoda zahteva kontrolera automatski vraća objekte u `HttpResponse`.

▼ Poglavlje 4

Kreiranje veb klijenta za Spring backend

STRANICA ZA PRIJAVU KORISNIKA

Login stranica sadrži formu kojom se proveravaju korisnički podaci.

Za kreiranje frontend strane odnosno klijenta korišćene su JSP strane koje u sebi sadrže HTML, CSS, JavaScript i koriste svoj EL (Expression Language) jezik.

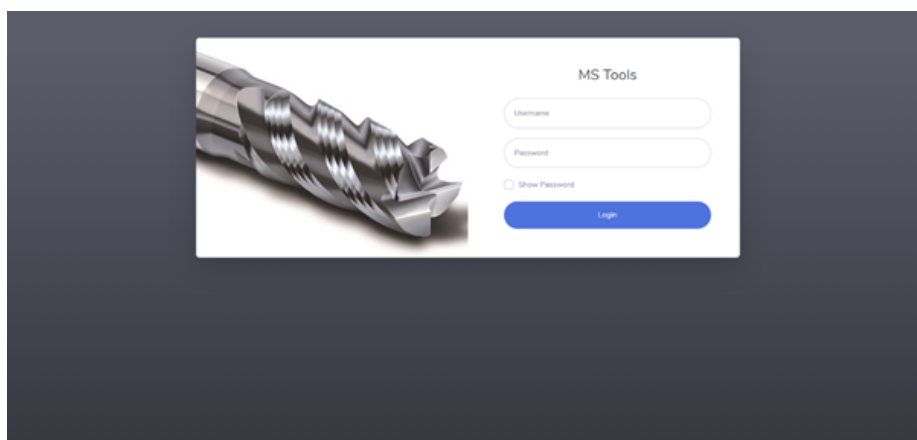
Veb aplikacija koristi moderan CSS šablon koji doprinosi celokupnom korisničkom iskustvu.

Login stranica sadrži formu kojom se proveravaju korisnički podaci. Dalji pristup veb aplikaciji nije moguć sve dok uneti podaci ne budu korektni.

Forma koristi Spring Security okvir za proveru podataka.

```
<form class="user" accept-charset="UTF-8" role="form" name='loginForm' action="{c:url value='/j_spring_security_check' />" method='POST'>
  <div class="form-group">
    <input type="text" class="form-control form-control-user" placeholder="Username" name="username">
  </div>
  <div class="form-group">
    <input type="password" class="form-control form-control-user" id="password" placeholder="Password" name="password" value="">
  </div>
  <div class="form-group">
    <div class="custom-control custom-checkbox small">
      <input type="checkbox" class="custom-control-input" id="customCheck" onclick="myFunction()" />
      <label class="custom-control-label" for="customCheck">Show Password</label>
    </div>
  </div>
  <input type="submit" name="submit" class="btn btn-primary btn-user btn-block" value="Login">
  <input type="hidden" name="{_csrf.parameterName}" value="{_csrf.token}" />
</form>
```

Slika 4.1 Forma za login - listing [izvor: Miloš Savić - Završni rad]



Slika 4.2 Forma za login - izgled [izvor: Miloš Savić - Završni rad]

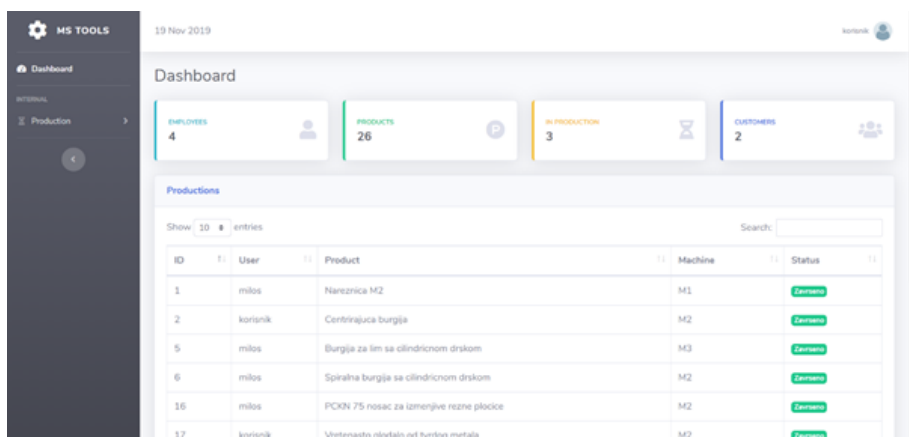
FUNKCIONALNOSTI OBIČNOG KORISNIKA

Funkcionalnosti običnog korisnika omogućene su kroz interakciju sa odgovarajućom stranicom.

Prvo ćemo se ulogovati kao običan korisnik i predstaviti njegove dostupne funkcionalnosti.

Na početnoj stranici vidimo lepo dizajniran prikaz koji nam daje informacije o trenutnom broju kupaca, proizvoda, zaposlenih i koliko je trenutno proizvoda u proizvodnji. Zatim u tabeli imamo spisak korisnika sa podacima koji proizvod prave na kojoj mašini i u kom je statusu izrade.

Ova početna strana je takođe ista za admin korisnika, ali je navigacija drugačija.

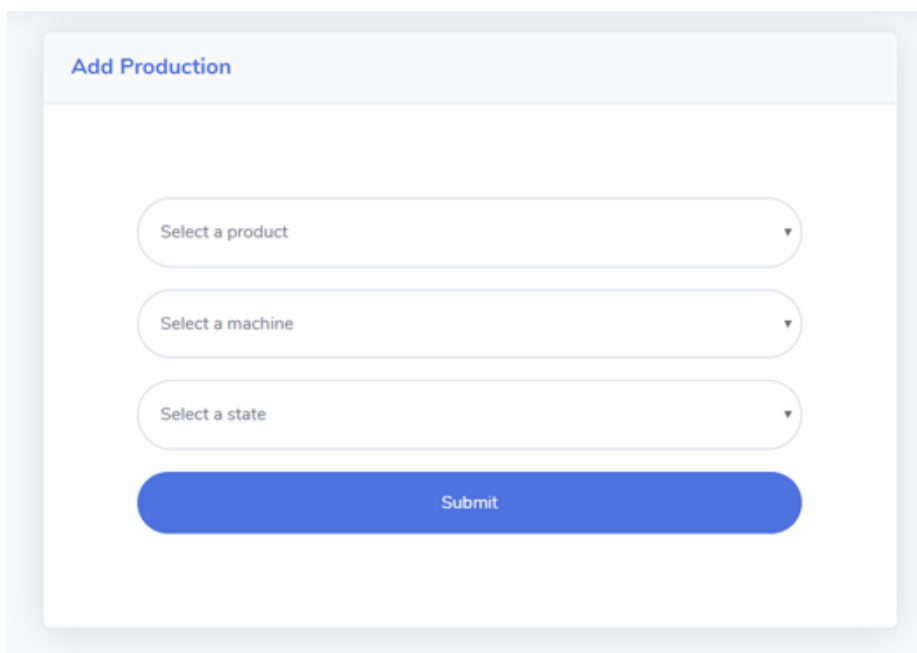


The screenshot shows a dashboard for 'MS TOOLS' dated 19 Nov 2019. It features four summary cards: EMPLOYEES (4), PRODUCTS (26), IN PRODUCTION (3), and CUSTOMERS (2). Below these is a 'Productions' section with a table listing production entries. The table has columns for ID, User, Product, Machine, and Status. The status for all entries is 'Završeno' (Completed).

| ID | User | Product | Machine | Status |
|----|----------|--|---------|----------|
| 1 | milos | Narznica M2 | M1 | Završeno |
| 2 | korisnik | Centrifuga burgija | M2 | Završeno |
| 5 | milos | Burgije za lim sa cilindricnom drskom | M3 | Završeno |
| 6 | milos | Spiralna burgija sa cilindricnom drskom | M2 | Završeno |
| 16 | milos | POKN 75 nosac za izmenjive rezne pločice | M2 | Završeno |
| 17 | korisnik | Vretenasto glodalo od tvrdog metala | M2 | Završeno |

Slika 4.3 Početna strana običnog korisnika [izvor: Miloš Savić - Završni rad]

Glavni zadatak zaposlenih je da unesu koji proizvod prave na kojoj mašini i u kom je statusu izrade. To obavljaju na stranici [Add Production](#) gde imaju formu za unos. (Slika 4.)



The 'Add Production' form contains three dropdown menus for selecting a product, a machine, and a state, followed by a blue 'Submit' button.

Slika 4.4 Funkcionalnosti običnog korisnika [izvor: Miloš Savić - Završni rad]

FORMA OBIČNOG KORISNIKA

Forma se sastoji iz padajućih lista koje prikazuju podatke iz baze podataka.

Forma se sastoji iz padajućih lista koje prikazuju podatke iz baze podataka. Upravo ovo je jedna korisna opcija EL jezika koji se koristi u *JSP* stranama. Koristi se *requestScope* koji će automatski učitati podatke prvobitno dobijene *Hibernate* transakcijom, a zatim prosledjene od strane kontrolera.

```
<form:form id="myform" class="user" method="POST" action="${post_url}" modelAttribute="production">
  <% String success = (String) request.getAttribute("successMsg");%>
  <%= (success != null) ? "<div class='alert alert-success'>" + success + "</div>" : ""%>
  <c:if test="${not empty error}">
    <div class="alert alert-danger">${error}</div>
  </c:if>

  <div class="form-group">
    <form:input type="hidden" id="productionId" class="form-control form-control-user" placeholder="Production ID" />
    <form:select id="productId" style="font-size:.8rem;border-radius:10rem;height:50px;" class="form-control">
      <form:option value="">Select a product</form:option>
      <form:options items="${requestScope.products}" itemValue="productId" itemLabel="product" />
    </form:select>
  </div>
  <div class="form-group">
    <form:select id="machineId" style="font-size:.8rem;border-radius:10rem;height:50px;" class="form-control">
      <form:option value="">Select a machine</form:option>
      <form:options items="${requestScope.machines}" itemValue="machineId" itemLabel="name" />
    </form:select>
  </div>
</form:form>
```

Slika 4.5 Listing forme [izvor: Miloš Savić - Završni rad]

Na stranici *My Productions* zaposleni može da vidi sve svoje proizvode u bilo kom statusu. Proizvodi koji nisu u statusu „završeno“ mogu da se ažuriraju odnosno može da im se menja status. Kada pređu u status „završeno“ automatski se gubi opcija ažuriranja i broj dostupnih proizvoda se povećava u bazi podataka. Kod iz kontrolera koji ovo obavlja može da se vidi na sledećem listingu.

```
540 @RequestMapping(value = "/editProduction", method = RequestMethod.POST)
541 public ModelAndView editProduction(@ModelAttribute("production") Production p,
542                                   model.addAttribute("production", p);
543                                   Principal principal = request.getUserPrincipal();
544                                   Users u = crudDao.findUserByUsername(principal.getName());
545                                   p.setUser(u);
546                                   crudDao.editProduction(p);
547
548                                   Product product = crudDao.getProductById(p.getProductId().getProductId());
549
550                                   if (p.getProdStatus().equals("Završeno")) {
551                                     System.out.println(productStatus);
552                                     int q = Integer.parseInt(product.getQuantity()) + 1;
553                                     product.setQuantity(Integer.toString(q));
554                                     crudDao.editProduct(product);
555                                   }
556
557                                   return new ModelAndView("redirect:/myProduction");
558 }
```

Slika 4.6 EditProduction POST metoda [izvor: Miloš Savić - Završni rad]

Productions

A list of all productions.

Productions

Show 10 entries Search:

| ID | User | Product | Machine | Status | Action |
|----|----------|-------------------------------------|---------|----------|-------------|
| 2 | korisnik | Centrirajuca burgija | M2 | Završeno | DELETE |
| 17 | korisnik | Vretenasto glodalo od tvrdog metala | M2 | Završeno | DELETE |
| 18 | korisnik | Metalni listovi za kružnu pil 20x3 | M2 | U toku | EDIT DELETE |
| ID | User | Product | Machine | Status | Action |

Showing 1 to 3 of 3 entries

Previous 1 Next

Slika 4.7 My Productions stranica [izvor: Miloš Savić - Završni rad]

Obični korisnik takođe može da vidi svoj profil i promeni profilnu sliku kao i *admin* korisnik što će biti predstavljeno u nastavku.

POČETNA STRANICA ADMIN KORISNIKA

Admin korisnik poseduje znatno više opcija od običnog korisnika

Kada se ulogujemo kao *admin* korisnik primetićemo da imamo znatno više opcija od običnog korisnika.

Administrator ima sledeće mogućnosti:

- Dodavanje, brisanje i editovanje korisnika
- Dodavanje, brisanje i editovanje kupaca
- Dodavanje, brisanje i editovanje porudžbina
- Dodavanje, brisanje i editovanje kategorija proizvoda
- Dodavanje, brisanje i editovanje proizvoda
- Dodavanje, brisanje i editovanje mašina
- Pregled svih proizvoda u izradi i brisanje samo onih koji su u statusu „završeno“
- Promena profilne slike

MS TOOLS

19 Nov 2019

Dashboard

EMPLOYEES 4 PRODUCTS 26 IN PRODUCTION 3 CUSTOMERS 2

Productions

Show 10 entries Search:

| ID | User | Product | Machine | Status |
|----|----------|--|---------|----------|
| 1 | milos | Narznica M2 | M1 | Završeno |
| 2 | korisnik | Centrirajuca burgija | M2 | Završeno |
| 5 | milos | Burgija za lim sa cilindricnom drskom | M3 | Završeno |
| 6 | milos | Spiralna burgija sa cilindricnom drskom | M2 | Završeno |
| 16 | milos | POKX 75 nosac za izmenjive rezne pločice | M2 | Završeno |
| 17 | korisnik | Vretenasto glodalo od tvrdog metala | M2 | Završeno |

Slika 4.8 Početna stranica administratora [izvor: Miloš Savić - Završni rad]

FUNKCIONALNOSTI ADMIN KORISNIKA ZA PREGLED I AŽURIRANJE KORISNIKA

Sve forme proveravaju unos podataka i daju povratnu poruku ukoliko neki podatak nedostaje.

Zbog velike obimnosti neće biti prikazane sve stranice kojima *admin* može da pristupi.

Treba naglasiti da sve forme proveravaju unos podataka i daju povratnu poruku ukoliko neki podatak nedostaje, takođe poruka se prikazuje i za uspešan unos.

Admin može da vidi listu svih korisnika, da edituje podatke i da briše korisnike. (Slika 9) Ova lista nije potpuna, ne može da se vidi lozinka zbog privatnosti podataka.

| ID | Username | First Name | Last Name | Email | Action |
|----|----------|------------|-----------|--------------------|---|
| 1 | admin | Milos | Savic | admin@gmail.com | <button>EDIT</button> <button>DELETE</button> |
| 2 | milos | Milos | Savic | milos@gmail.com | <button>EDIT</button> <button>DELETE</button> |
| 3 | korisnik | korisnik | korisnik | korisnik@gmail.com | <button>EDIT</button> <button>DELETE</button> |
| 4 | test | Test | Test | test@gmail.com | <button>EDIT</button> <button>DELETE</button> |

Slika 4.9 Tabela korisnika [izvor: Miloš Savić - Završni rad]

Edit user

Username

milos

First name

Milos

Last name

Savic

Email

milos@gmail.com

Cancel

Update

Slika 4.10 Izmena podataka korisnika [izvor: Miloš Savić - Završni rad]

Ažuriranje (**edit**) korisnika kao i svih ostalih objekata prikazuje se u modalnom prozoru. Kao što vidimo na slici 10 **admin** nema mogućnost promene korisničkog imena kao ni mogućnost promena lozinke.

Klikom na dugme edit poziva se modalni prozor uz pomoć komande `data-target="#EditModal${u.userId}"`.

U **Edit Modal** delu se nalazi forma koja koristi **POST** metodu za ažuriranje podataka. Ovaj princip se koristi na svim stranicama koje imaju opciju ažuriranje.

FUNKCIONALNOSTI ADMIN KORISNIKA ZA KREIRANJE PORUDŽBINE

Stranica za kreiranje porudžbine daje korisne podatke prilikom odabira.

```
<!-- Edit Modal -->
<div class="modal fade" id="EditModal${u.userId}" tabindex="-1" role="dialog" aria-labelledby="exampleModalLabel">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModalLabel">Edit user</h5>
        <button class="close" type="button" data-dismiss="modal" aria-label="Close">
          <span aria-hidden="true">×</span>
        </button>
      </div>
      <div class="modal-body">
        <:url var="post_url" value="/editUser" />
        <form id="myform" class="user" method="POST" action="${post_url}" modelAttribute="user">
          <div class="form-group">
            <form:input type="hidden" id="userId" class="form-control" value="${u.userId}" path="us
            <form:label path="username" style="font-size:.8rem;border-radius:10rem;">&nbsp;&nbsp;&nbsp;Us
            <form:input type="text" id="username" style="font-size:.8rem;border-radius:10rem;height
```

Slika 4.11 Edit Modal [izvor: Miloš Savić - Završni rad]

Stranica za kreiranje porudžbine daje korisne podatke prilikom odabira. Kada admin odabere proizvod za koji pravi porudžbinu automatski dobija prikaz broja dostupnih proizvoda. Nije moguće uneti veći broj od dostupnog broja. Takođe se prikazuje cene proizvoda koja se automatski množi sa brojem poručenih proizvoda. Za ovo automatsko množenje je zadužena jQuery skripta. (sledeći listing).

Kada se rezervacija uspešno kreira broj rezervisanih proizvoda se oduzima od broja dostupnih proizvoda.

```
// jQuery 'keyup' to trigger the computation as the user type
$('.calc').keyup(function () {

  var quantity = document.getElementById('quantity').value;
  var price = document.getElementById('price').value;

  var result = quantity * price;

  $('#totalPrice').val(result);
```

Slika 4.12 Skripta za automatsko množenje cene porudžbine [izvor: Miloš Savić - Završni rad]

KREIRANJE PORUDŽBINE

Prilikom kreiranja porudžbine datum i vreme kreiranja se automatski dodaju.

Create An Order

Milos

Nareznica M4

Available quantity: 23

2

Price of one: 18 €

36

Submit

Slika 4.13 Forma za kreiranje porudžbine [izvor: Miloš Savić - Završni rad]

Prilikom kreiranja porudžbine datum i vreme kreiranja se automatski dodaju.

Orders

A list of all orders.

Orders

Show 10 entries

Search:

| ID | Customer | Product | Order Date | Quantity | Price in € | Action |
|----|----------|--|---------------------|----------|------------|--------|
| 5 | Jovana | Mikro burgija | 08-11-2019 16:11 PM | 19 | 12 | DELETE |
| 6 | Milos | Spiralna burgija sa cilindricnom drskom | 08-11-2019 18:59 PM | 22 | 4 | DELETE |
| 7 | Milos | DCLN 95 nosac za izmenjive rezne plovice | 08-11-2019 20:21 PM | 82 | 11 | DELETE |

Slika 4.14 Tabela porudžbina [izvor: Miloš Savić - Završni rad]

BRISANJE IZ TABELA

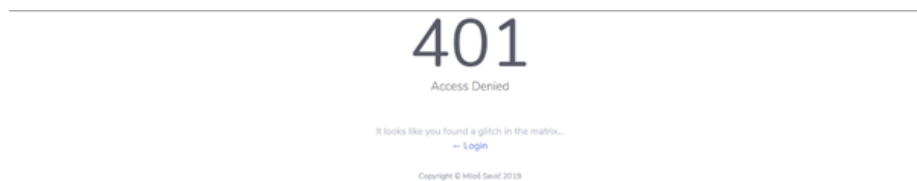
Brisanje iz tabela se vrši tako što se klikom na dugme GET metodi prosleđuje ID.

Brisanje iz tabela se vrši tako što se klikom na dugme **GET** metodi prosleđuje ID, odabrano polje se briše i stranica se ponovo osvežava.

```
169 @RequestMapping(value = "/deleteUser/{userId}", method = RequestMethod.GET)
170 public String deleteUser(@PathVariable("userId") int userId, HttpServletRequest request) {
171     System.out.println("Fetching & Deleting user with id " + userId);
172     Users user = crudDao.getUserById(userId);
173     if (user == null) {
174         System.out.println("Unable to delete. User with id " + userId + " not found");
175         String referer = request.getHeader("Referer");
176         return "redirect:" + referer;
177     }
178     crudDao.deleteUser(user);
179     String referer = request.getHeader("Referer");
180     return "redirect:" + referer;
181 }
```

Slika 4.15 Metoda za brisanje korisnika [izvor: Miloš Savić - Završni rad]

Ukoliko korisnik pokuša da pristupi stranici kojoj nema prava pristupa zahvaljujući funkcijama koje pruža Spring Security biće automatski preusmeren na stranicu *Access Denied* (sledeća slika), a to znači da brisanje ili editovanje preko linka nije moguće bez potrebnih privilegija.

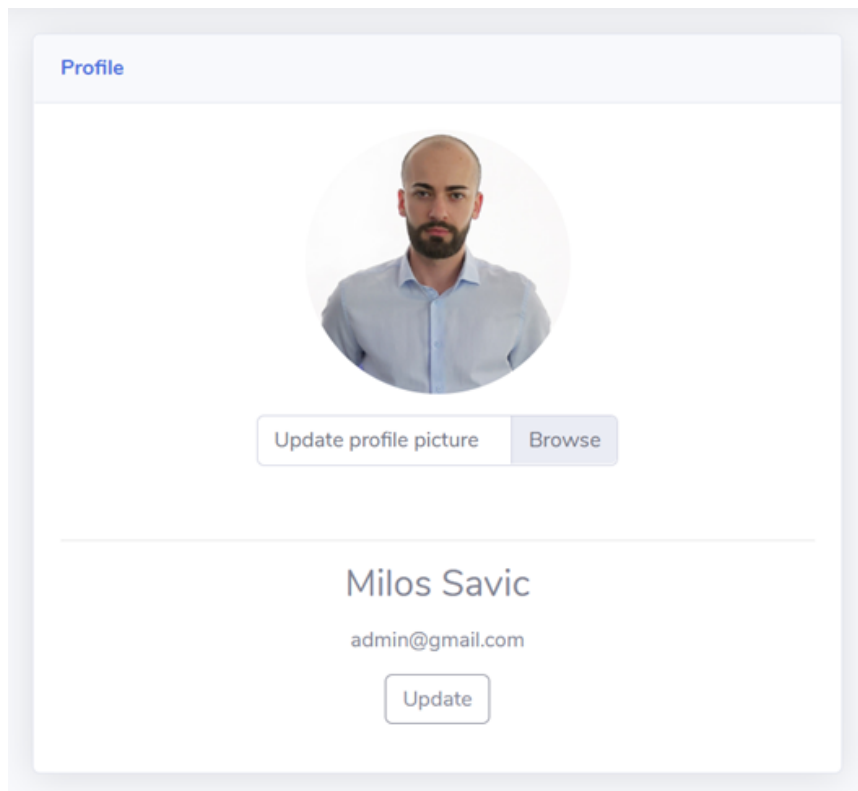


Slika 4.16 Access Denied stranica [izvor: Miloš Savić - Završni rad]

PROFIL ADMIN KORISNIKA

Admin korisnik kao i obični korisnik mogu da vide svoj profil.

Admin korisnik kao i obični korisnik mogu da vide svoj profil gde se prikazuje njihova profilna slika sa opcijom za izmenu. Pored toga prikazuje se ime, prezime i email adresa korisnika.



Slika 4.17 Profilna stranica [izvor: Miloš Savić - Završni rad]

Spring ima ugrađenu podršku za multipart za rukovanje datotekama u veb aplikacijama. Dizajn za multipart podršku je izveden uz pomoć [MultipartResolver](#)-a, definisanim u [org.springframework.web.multipart](#) paketu.

Svaki zahtev će biti pregledan da li sadrži multipart deo. Ako multipart nije pronađen, zahtev će se nastaviti kako se očekuje. Međutim, ako se u zahtevu nađe multipart, koristiće se [MultipartResolver](#) koji je deklarisan u kontekstu. Nakon toga, multipart atribut tretiraće se kao bilo koji drugi atribut.

```
@RequestMapping(value = "/profile", method = RequestMethod.POST)
public ModelAndView getProfile(@ModelAttribute("user") Users u, ModelAndView model, @RequestParam("file")
    MultipartFile file, HttpSession session) throws Exception {
    ServletContext context = session.getServletContext();
    String path = context.getInitParameter("upload.location");
    String filename = file.getOriginalFilename();

    byte[] bytes = file.getBytes();
    try (BufferedOutputStream stream = new BufferedOutputStream(new FileOutputStream(new File
        (path + File.separator + filename)))) {
        stream.write(bytes);
        stream.flush();
    }

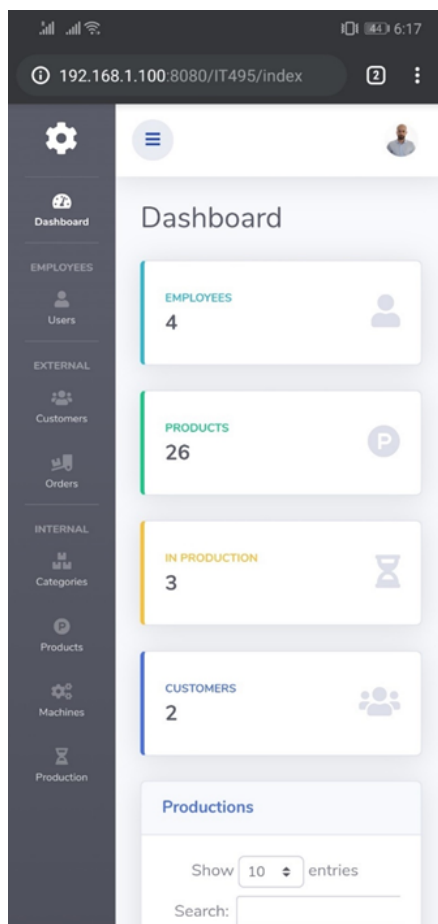
    u.setProfilePic("images/" + filename);
    model.addObject("object", u);
    crudDao.editUser(u);
    Thread.sleep(2000);
    model.setViewName("redirect:/profile");
    return model;
}
```

Slika 4.18 Deo kontrolera koji je zadužen za multipart [izvor: Miloš Savić - Završni rad]

RESPONZIVAN DIZAJN

Bitno je naglasiti da veb aplikacija ima responzivan dizajn, a sve zahvaljujući Bootstrap biblioteci.

Bitno je naglasiti da veb aplikacija ima responzivan dizajn, a sve zahvaljujući Bootstrap-u. Ovo znači da se sadržaj aplikacije automatski prilagođava rezoluciji displeja na kom se prikazuje (sledeća slika).



Slika 4.19 Veb aplikacija na mobilnom telefonu [izvor: Miloš Savić - Završni rad]

▼ Poglavlje 5

Kreiranje mobilnog klijenta za Spring backend

DODAVANJE ZAVISNOSTI U ANDROID PROJEKAT

Dodavanje zavisnosti u Android projekat je prvi korak ka kreiranju mobilnog klijenta.

Za razvoj Android aplikacije korišćen je Android Studio sa ugrađenim emulatorom za lakše testiranje aplikacije. Ciljana verzija operativnog sistema za koji je pisana ova aplikacije je API 25 odnosno Android 7.1.1 (Nougat) i predstavlja najnižu podržanu Android verziju.

U projekat je potrebno dodati zavisnosti koje su prikazane na sledećoj slici.

```
22 dependencies {
23     implementation fileTree(dir: 'libs', include: ['*.jar'])
24     androidTestCompile('com.android.support.test.espresso:espresso-core:3.0.2', {
25         exclude group: 'com.android.support', module: 'support-annotations'
26     })
27     implementation 'com.android.support:appcompat-v7:25.0.0'
28     implementation 'com.android.support:design:25.0.0'
29     implementation 'com.android.support:recyclerview-v7:25.0.0'
30     implementation 'com.android.support:cardview-v7:25.0.0'
31
32     implementation 'com.squareup.retrofit2:retrofit:2.6.2'
33     implementation 'com.squareup.retrofit2:converter-gson:2.6.2'
34
35     testImplementation 'junit:junit:4.12'
36 }
```

Slika 5.1 Neophodne zavisnosti [izvor: autor]

Kao što smo već spomenuli Spring uz pomoć REST kontrolera kreira JSON format koji treba da se parsira u aplikaciji. Za parsiranje JSON teksta koristiće se popularni alat Retrofit (<https://square.github.io/retrofit/>).

Retrofit je REST klijent za Javu i Android. Omogućava relativno lako preuzimanje i učitavanje JSON-a (ili drugih strukturiranih podataka) putem REST veb servisa. U Retrofit - u konfigurišemo koji konverter se koristi za serijalizaciju podataka. Obično za JSON koristimo Gson, ali mogu se dodati prilagođeni konverteri za obradu XML-a ili drugih protokola. Retrofit koristi biblioteku OkHttp za HTTP zahteve.

Retrofit takođe koristi jedan poznat koncept, a to je mapiranje klasa odnosno entiteta. Stoga su kreirane klase za potrebne objekte. Sledi listing delova koda.

Klasa Production je glavna klasa koja definiše sve JSON objekte.

```

1 package com.metropolitan.it495;
2
3 public class Production {
4     private String productionId;
5     private String prodStatus;
6     private ProductId productId;
7     private User user;
8     private MachineId machineId;
9
10
11     public String getProductionId() { return productionId; }
12
13
14     public String getProdStatus() { return prodStatus; }
15
16
17     public ProductId getProductId() { return productId; }
18
19
20     public User getUser() { return user; }
21
22
23     public MachineId getMachineId() { return machineId; }
24 }

```

Slika 5.2 Klasa Production [izvor: autor]

DODATNE KLASSE

Sve klase koje nisu tipa String, odnosno koje su objekti imaju svoju posebnu bazu u kojoj su definisani njihovi atributi.

Sve klase koje nisu tipa String, odnosno koje su objekti imaju svoju posebnu bazu u kojoj su definisani njihovi atributi.

```

1 package com.metropolitan.it495;
2
3 public class ProductId {
4     private String productname;
5
6     public String getProductname() { return productname; }
7 }

```

Slika 5.3 ProductId klasa [izvor: autor]

JSONResponse klasa definiše myArray niz u kom se nalaze *JSON* objekti.

```

1 package com.metropolitan.it495;
2
3 public class JSONResponse {
4     private Production[] myArray;
5
6     public Production[] getMyArray() { return myArray; }
7 }

```

Slika 5.4 JSONResponse klasa [izvor: autor]

U RequestInterface-u nalazi se GET metoda gde je definisan deo lokacije koji dolazi nakon glavnog URL-a.

```

1 package com.metropolitan.it495;
2
3 import ...
4
5
6 public interface RequestInterface {
7
8     @GET("IT495/api/production")
9     Call<JSONResponse> getJSON();
10 }

```

Slika 5.5 RequestInterface [izvor: autor]

U klasi *DataAdapter* nalazi se jedna bitna metoda, a to je metoda za filtriranje podataka. Ova metoda filtrira Array listu koja je prvobitno kreirana preuzimanjem JSON podataka.

```

41 @Override
42 public Filter getFilter() {
43     return new Filter() {
44         @Override
45         protected FilterResults performFiltering(CharSequence charSequence) {
46             String charString = charSequence.toString();
47             if (charString.isEmpty()) {
48                 mFilteredList = null;
49             } else {
50                 ArrayList<Production> filteredList = new ArrayList<>();
51                 for (Production production : mArrayList) {
52                     if (production.getMachineId().getName().toLowerCase().contains(charString)) {
53                         filteredList.add(production);
54                     }
55                 }
56                 mFilteredList = filteredList;
57             }
58             FilterResults filterResults = new FilterResults();
59             filterResults.values = mFilteredList;
60         }
61     };
62 }

```

Slika 5.6 Deo klase DataAdapter [izvor: autor]

JSON PARSIRANJE

Sledi prikaz metode za parsiranje JSON podataka uz pomoć Retrofit biblioteke.

Sledi prikaz metode za parsiranje *JSON* podataka uz pomoć *Retrofit* biblioteke. Možemo da da vidimo da *Retrofit* dosta pojednostavljuje kod.

```

private void loadJSON() {
    Retrofit retrofit = new Retrofit.Builder()
        .baseUrl(BASE_URL)
        .addConverterFactory(GsonConverterFactory.create())
        .build();
    RequestInterface request = retrofit.create(RequestInterface.class);
    Call<JSONResponse> call = request.getJSON();
    call.enqueue(new Callback<JSONResponse>() {
        @Override
        public void onResponse(Call<JSONResponse> call, Response<JSONResponse> response) {
            JSONResponse jsonResponse = response.body();
            mArrayList = new ArrayList<>(Arrays.asList(jsonResponse.getMyArray()));
            mAdapter = new DataAdapter(mArrayList);
            mRecyclerView.setAdapter(mAdapter);
        }

        @Override
        public void onFailure(Call<JSONResponse> call, Throwable t) {
            Log.d("Error", t.getMessage());
        }
    });
}

```

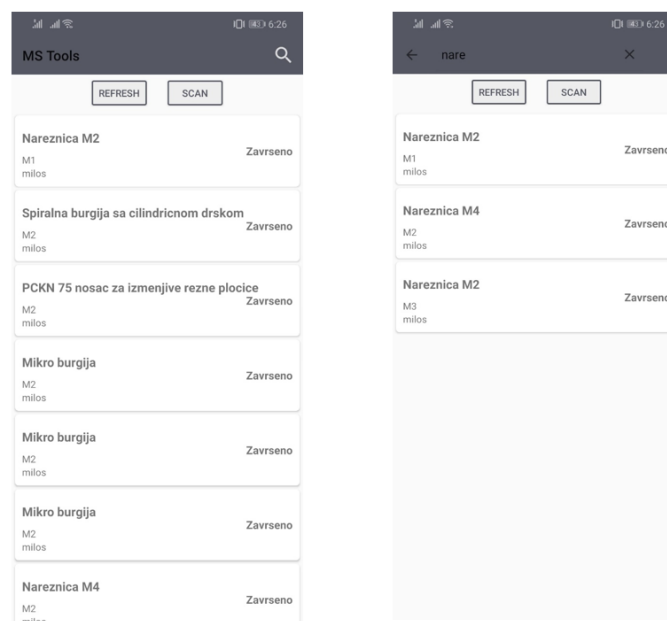
Slika 5.7 loadJSON metoda [izvor: autor]

DEMO

Aplikacija koristi RecyclerView unutar koga se prikazuje CardView

Aplikacija koristi *RecyclerView* unutar koga se prikazuje *CardView*. (Slika) Korisnik ima dva načina pretrage. Prvi način je ručno unošenje teksta u *SearchView* gde se automatski vrši filtriranje po svim poljima. A drugi način je skeniranjem QR koda koji je jedinstven za svaki proizvod.

Za skeniranje QR koda aplikacija se služi Intent funkcijom i poziva aplikaciju *Barcode Scanner+* koja nakon skeniranja vraća učitani podatak.

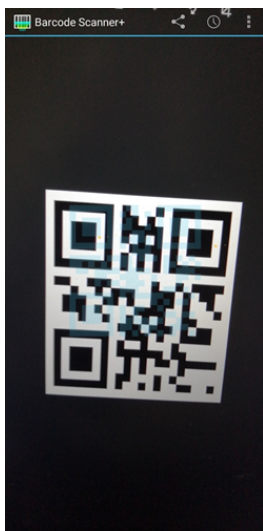


Slika 5.8 Početna strana aplikacije i prikaz filtriranog podatka [izvor: autor]

```
public void HandleClick(View view) {
    Intent intent = new Intent(action: "com.google.zxing.client.android.SCAN");
    intent.putExtra(name: "SCAN_MODE", value: "QR_CODE_MODE");
    startActivityForResult(intent, requestCode: 0);
}

public void onActivityResult(int requestCode, int resultCode, Intent intent) {
    if (requestCode == 0) {
        EditText tvStatus=(EditText)findViewById(R.id.textView);
        if (resultCode == RESULT_OK) {
            tvStatus.setText(intent.getStringExtra(name: "SCAN_RESULT"));
        } else if (resultCode == RESULT_CANCELED) {
            Toast.makeText(getApplicationContext(), text: "Scan cancelled.", Toast.LENGTH_SHORT).show();
        }
    }
}
```

Slika 5.9 Deo koda za pokretanje QR skenera i preuzimanje podataka



Slika 5.10 Skeniranje QR koda [izvor: autor]

▼ Poglavlje 6

Pokazna vežba 14 (45 min)

KREIRANJE ANDROID KLIJENTA ZA REST - INTERNET DOZVOLA

Demo kreiranja Android klijenta za konzumiranje REST Spring servisa.

Cilj pokazne vežbe je kreiranje mobilnog REST klijenta koji bi mogao da konzumira Spring REST servise po uzoru na sledeći video materijal.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

U prvom koraku neophodno je dodeliti aplikaciji dozvolu za upotrebu Interneta, a to se dešava u datoteci AndroidManifest.xml. Navedeno je prikazano sledećim listingom.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.abhishekpanwar.receivedatajson">

    <uses-permission android:name="android.permission.INTERNET"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

KREIRANJE KORISNIČKOG INTERFEJSA

U narednom koraku je potrebno kreirati i grafički korisnički interfejs aplikacije.

U narednom koraku je potrebno kreirati i grafički korisnički interfejs aplikacije. Sledi listing datoteke `activity_main.xml`. Iz GUI - a se inicira REST poziv i u njemu se prikazuju rezultati backend odgovora.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.abhishekpanwar.receivedatajson.MainActivity">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Me!"
        android:layout_marginBottom="10dp"
        android:id="@+id/button"/>

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@id/button">
        <TextView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:padding="5dp"
            android:textSize="24sp"
            android:id="@+id/fetcheddata"
            android:hint="Fetched Text Here!!!"/>
        </ScrollView>

    </RelativeLayout>
```

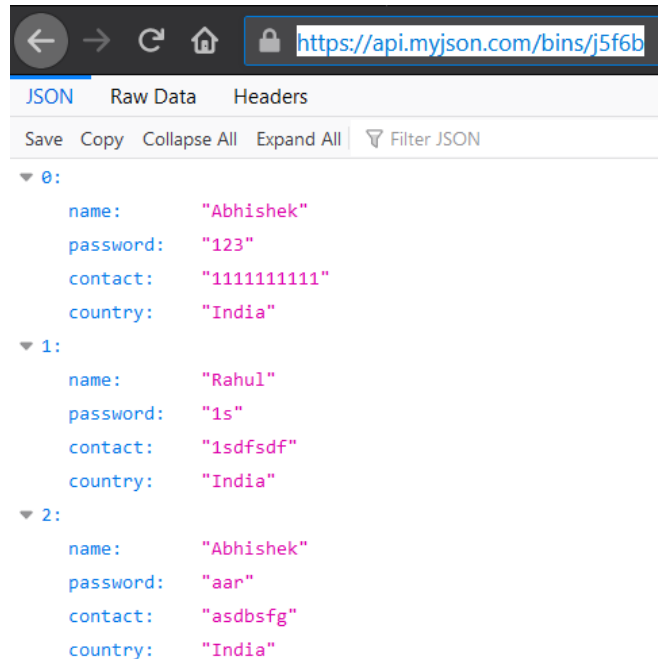
SIMULIRANI SERVER

Zbog vremenske ograničenosti individualnih vežbi koristićemo simulirani server.

Zbog vremenske ograničenosti individualnih vežbi koristićemo simulirani server. REST poziv će se vršiti na osnovu sledećeg linka:

<https://api.myjson.com/bins/j5f6b>

Na ovoj lokaciji se nalazi sledeća JSON reprezentacija:



Slika 6.1 Simulirani server - JSON reprezentacija [izvor: api.myjson.com]

GLAVNA KLASA AKTIVNOSTI

Glavna klasa aktivnosti ima zadatak da učitava GUI i da pokrene glavnu programsku logiku.

Glavna klasa aktivnosti ima zadatak da učitava GUI i da pokrene glavnu programsku logiku.

Sledećim listingom je dat kod klase *ActivityMain.java*.

```

public class MainActivity extends AppCompatActivity {
    Button click;
    public static TextView data;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        click = (Button) findViewById(R.id.button);
        data = (TextView) findViewById(R.id.fetchdata);

        click.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                fetchData process = new fetchData();
            }
        });
    }
}

```

```

        process.execute();
    }
});
}
}

```

Pored učitavanja GUI u `onCreate()` metodi klasa ima zadatak da omogući pokretanje pozadinskog (asinhronog) procesa nakon klika na kreirano dugme. Upravo u ovom asinhronom zadatku dešava se REST poziv, a to je definisano kroz kreiranje objekta tipa `fetchData` i njegovim izvršavanjem metode `execute()`.

ASINHRONI POZIV REST SERVISA

U asinhronom zadatku dešava se REST poziv.

Kao što je istaknuto u prethodnom izlaganju, u asinhronom zadatku dešava se REST poziv, a to je definisano kroz kreiranje objekta tipa `fetchData` i njegovim izvršavanjem metode `execute()` u glavnoj klasi aktivnosti. Sledi listing klase `fetchData`:

```

public class fetchData extends AsyncTask<Void,Void,Void> {
    String data = "";
    String dataParsed = "";
    String singleParsed = "";
    @Override
    protected Void doInBackground(Void... voids) {
        try {
            URL url = new URL("https://api.myjson.com/bins/j5f6b");
            HttpURLConnection httpURLConnection = (HttpURLConnection)
url.openConnection();
            InputStream inputStream = httpURLConnection.getInputStream();
            BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(inputStream));
            String line = "";
            while(line != null){
                line = bufferedReader.readLine();
                data = data + line;
            }

            JSONArray JA = new JSONArray(data);
            for(int i =0 ;i <JA.length(); i++){
                JSONObject JO = (JSONObject) JA.get(i);
                singleParsed = "Name:" + JO.get("name") + "\n"+
                    "Password:" + JO.get("password") + "\n"+
                    "Contact:" + JO.get("contact") + "\n"+
                    "Country:" + JO.get("country") + "\n";

                dataParsed = dataParsed + singleParsed + "\n" ;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```
    }

    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (JSONException e) {
        e.printStackTrace();
    }

    return null;
}

@Override
protected void onPostExecute(Void aVoid) {
    super.onPostExecute(aVoid);

    MainActivity.data.setText(this.dataParsed);
}
}
```

Upravo navedena metoda `execute()` je zadužena za izvršavanje REST poziva u asinhronoj niti, a to je definisano metodom `doInBackground()` klase `fetchData`. U metodi se dešava:

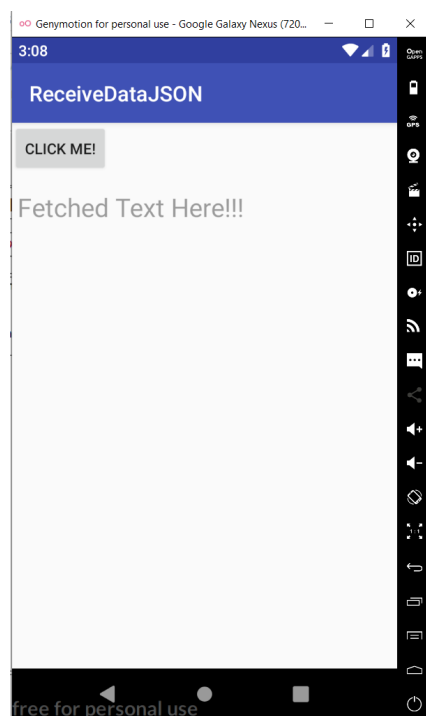
1. REST poziv ka navedenom linku - linije koda 8-16;
2. parsiranje JSON odgovora.

DEMO

Diskusija u vezi sa pokaznim primerom se završava kroz demonstraciju urađenog posla.

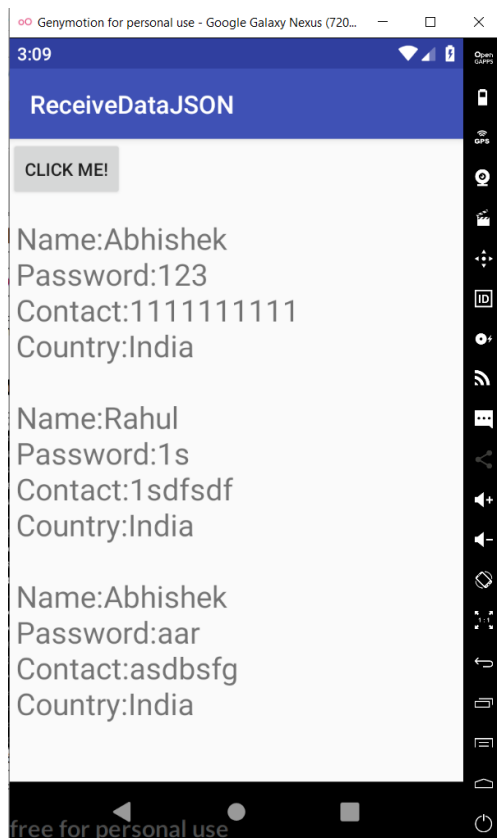
Diskusija u vezi sa pokaznim primerom se završava kroz demonstraciju urađenog posla.

Prva slika pokazuje učitano glavnu aktivnost sa pripadajućim korisničkim interfejsom.



Slika 6.2 Glavna aktivnost sa pripadajućim korisničkim interfejsom. [izvor: api.myjson.com]

Klikom na dugme obavlja se REST poziv, u asinhronoj niti, i dobija se odgovor kao na sledećoj slici.



Slika 6.3 Rezultat REST poziva u Android klijentu [izvor: api.myjson.com]

✓ Poglavlje 7

Individualna vežba 14

INDIVIDUALNA VEŽBA (90 MIN)

Pokušajte sami da kreirate

Nadograđivanje individualne vežbe iz Lekcije 9 - SPRING REST

ZADATAK:

1. Otvorite vaš zadatak individualne vežbe primenom razvojnog okruženja;
2. Angažujte urađeni zadatak na aplikativnom serveru;
3. Kreirajte Android mobilni klijent koji ima mogućnost poziva REST servisa definisanih u aplikaciji individualna vežba 9;
4. Detaljno dokumentovati DZ

▼ Poglavlje 8

Domaći zadatak 14

DOMAĆI ZADATAK BROJ 14 (120 MIN)

Cilj domaćeg zadatka je da student provežba naučeno na vežbama

Nadograđivanje domaćeg zadatka iz Lekcije 9 - SPRING REST

ZADATAK:

1. Otvorite vaš DZ9 primenom razvojnog okruženja;
2. Angažujte urađeni zadatak na aplikativnom serveru;
3. Kreirajte Android mobilni klijent koji ima mogućnost poziva REST servisa definisanih u aplikaciji DZ9;
4. Detaljno dokumentovati DZ

Nakon urađenog obaveznog zadatka studenti od predmetnog asistenta dobijaju dodatne različite zadatke putem emaila.

▼ Poglavlje 9

Zaključak

ZAKLJUČAK

U Lekciji 14 obrađeni su koncepti povezivanja mobilnih i veb Java tehnologija.

Lekcija je imala zadatak da kroz konkretan razvojni projekat ukaže na fleksibilnost *Spring* okvira da se povezuje sa različitim savremenim tehnologijama.

Posebno je istaknuto da osnovu diskusije čini Spring okvir, koji je baziran na Java programskom jeziku, u nekom sasvim drugom svetlu. Pored Spring okvira intenzivno su korišćene i ostale tehnologije kao što su *HTML*, *CSS*, *Bootstrap* i *JavaScript*, zajedno sa mobilnim tehnologijama. Što se tiče mobilne tehnologije kreirana je *Android* aplikacija koja komunicira sa *Spring* backend - om preko *REST* servisa.

Kreirana aplikacija predstavlja internu aplikaciju proizvodne kompanije koja će olakšati vođenje magacina. Slede neka zapažanja u vezi sa kreiranom aplikacijom. Administrator može da vidi koji zaposleni na kojoj mašini pravi određeni proizvod i u kom je statusu izrade taj proizvod. Admin korisnik može da vrši *CRUD* (*create*, *read*, *update* i *delete*) operacije nad korisnicima sistema odnosno zaposlenima, kupcima, proizvodima, porudžbinama, kategorijama proizvoda, mašinama i proizvodnjom. Obični korisnik može da upravlja samo svojom proizvodnjom.

Android aplikacija poseduje mogućnost pretrage proizvodnje kao i opciju skeniranja *QR* koda koji će učitati definisani proizvod.

Savladavanjem ove lekcije studenti su zaokružili primenu svih Java tehnologija izučavanih tokom studiranja.

DODATNI VIDEO MATERIJALI

Dodatni izvori za razumevanje sinergije mobilnih i Spring veb tehnologija.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

LITERATURA

U pripremanju Lekcije 14 korišćena je najaktuelnija pisana i web literatura.

Za pripremu lekcije korišćena je najnovija pisana i elektronska literatura:

1. Gary Mak, Josh Long, and Daniel Rubio, Spring Recipes Third Edition, Apress
2. Spring Framework Reference Documentation - <https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/>
4. Craig Walls, Spring Boot in Action, Manning
5. Delovi iz CS330 nastavnih materijala;
6. Delovi iz IT255 nastavnih materijala;

Dopunska literatura:

1. <http://www.javacodegeeks.com/tutorials/java-tutorials/enterprise-java-tutorials/spring-tutorials/>
2. <http://www.tutorialspoint.com/spring/>
3. <http://www.javatpoint.com/spring-tutorial>