Primjena Angular i Spring Boot tehnologija u izradi web stranice

Sertić, Ivan

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFROMACIJSKIH TEHNOLOGIJA

Sveučilišni studij računarstva

PRIMJENA ANGULAR I SPRING BOOT TEHNOLOGIJA U IZRADI WEB STRANICE

Završni rad

Ivan Sertić

Osijek, 2019.

Sadržaj

1. UVOD

Povećanje broja korisnika interneta dovelo je do veće potražnje za web-stranicama. Vremenom su korisnički zahtjevi postali sve složeniji, a samim time i vremenski zahtjevniji. Sve to potaknulo je razvoj novih programskih tehnologija za izradu web-stranica. Tako danas imamo mnogo različitih tehnologija koje su dostupne. Ovisno o postavljenim zahtjevima, odabiru se određene tehnologije za izradu. Svaka tehnologija nudi svoje pogodnosti koje olakšavaju proces izrade, no niti jedna tehnologija ne može sama poslužiti za izradu složenijih zadataka. Tako imamo tehnologije koje koristimo za izradu korisničkog sučelja, tehnologije koje koristimo za izradu pozadinske aplikacije(engl. *back-end*) koja upravlja logičkim koracima te tehnologiju koja omogućuje pohranu podataka i njezino čuvanje u bazama podataka.

Za izradu ovog završnog rada bilo je potrebno korištenje više različitih tehnologija od kojih su osnovne Angular i Spring Boot. Korisničko sučelje pisano je u Angularu koji zahtjeva poznavanje opisnog jezika HTML i programskog jezika TypeScript. Dizajn stranice napravljen je u okviru (engl. *framework*) Bootstrap. Pozadinska aplikacija pisana je u razvojnom okruženju IntelliJ, koje je omogućilo korištenje okvira Spring Boot koji je baziran na programskom jeziku Java. Postgres je omogućio korištenje sustava za relacijske baze podataka.

Uz uvod ovaj završni rad će prikazati u drugom poglavlju opis korištenih tehnologija te prednosti i nedostatke korištenja Angular i Spring Boota. U trećem poglavlju je opisan proces izrade web stranice od pozadinske aplikacije do korisničkog sučelja. Četvrto poglavlje prikazat će izgled korisničkog rješenja i sve njegove dijelove. Zadnje poglavlje je zaključak o naučenom.

1.1 Zadatak završnog rada

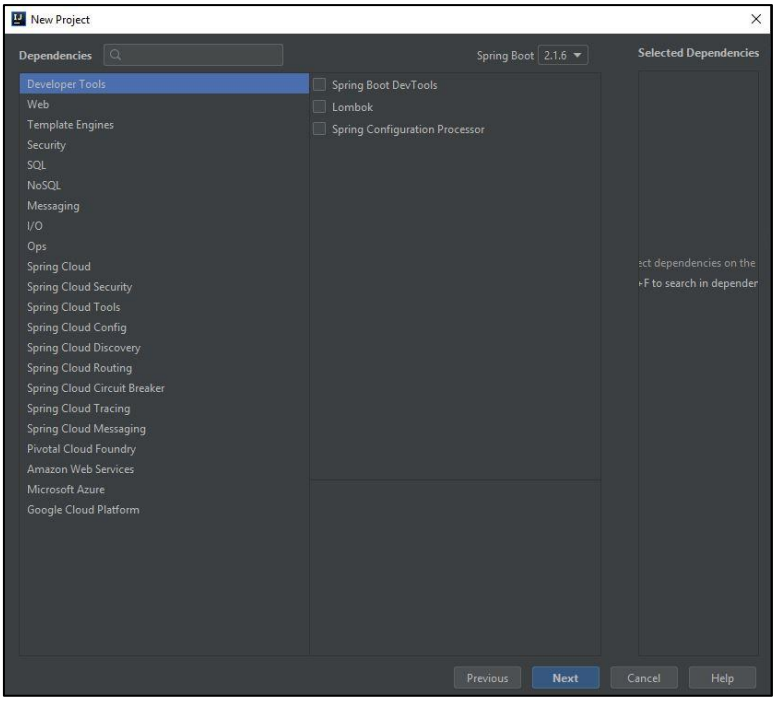
Zadatak ovog završnog rada je istražiti tehnologije za izradu web-stranice Angular i Springboot i navesti neke njihove prednosti i nedostatke. Prikazati stečena znanja o tehnologijama kroz proces izrade web-stranice i prikazati konačni rezultat u obliku funkcionalne web-strance.

2. KORIŠTENE TEHNOLGIJE

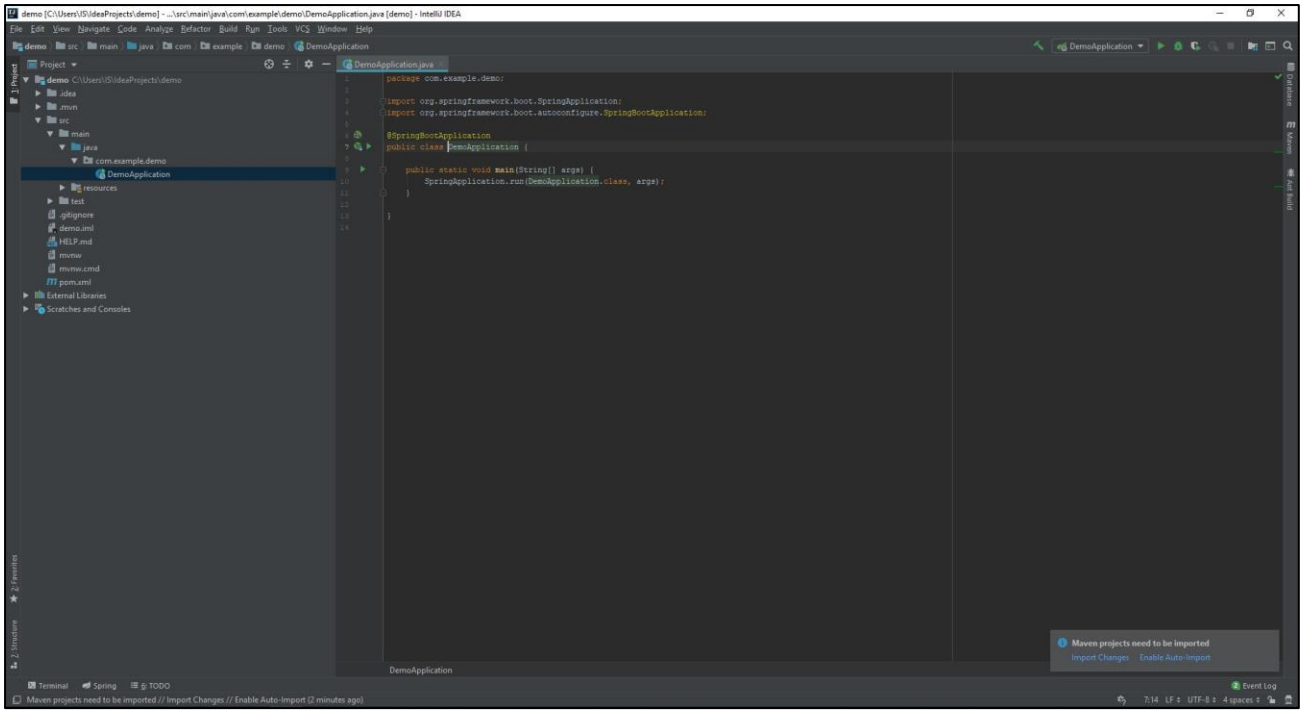
U ovom poglavlju opisane su tehnologije korištene pri izradi web-stranice, a to su: Angular, Bootstrap, Springboot i Postgres te programski jezici Java, TypeScript i opisni jezik HTML za koje su korišteno razvojno okruženje IntelliJ i Visual Studio Code program za uređivanje izvornog kôda.

2.1. Razvojno okruženje IntelliJ

IntelliJ je integrirano razvojno okruženje primarno namijenjeno Java programskom jeziku. Prema [1] razvila ga je firma IntelliJ, kasnije mijenja ime u JetBrains. Dvije dostupne verzije ovog okruženja, Community Edition i Ultimate Edition,, upotrebljavaju se za razvoj programa u komercijalne svrhe. Za razliku od besplatne verzije, Community Edition, Ultimate Edition nudi brojne pogodnosti vezene uz lakši razvoj aplikacija u Spring Bootu. Prema [2] JetBrains nudi besplatno Ultimate Edition verziju za sve studente, a potrebno se samo prijaviti putem službene elektroničke pošte fakulteta. Stvaranje novog projekta (Slika 2.1.) nudi sve potrebne pakete (engl. *dependecy*) za brzo i lako korištenje. Nakon odabira potrebnih paketa otvara se radna površina kruženja (Slika 2.2) .



Slika 2.1. Prikaz odabira paketa



Slika 2.2. Prikaz korisničkog sučelja IntelliJ-a

2.2. Programski jezik Java

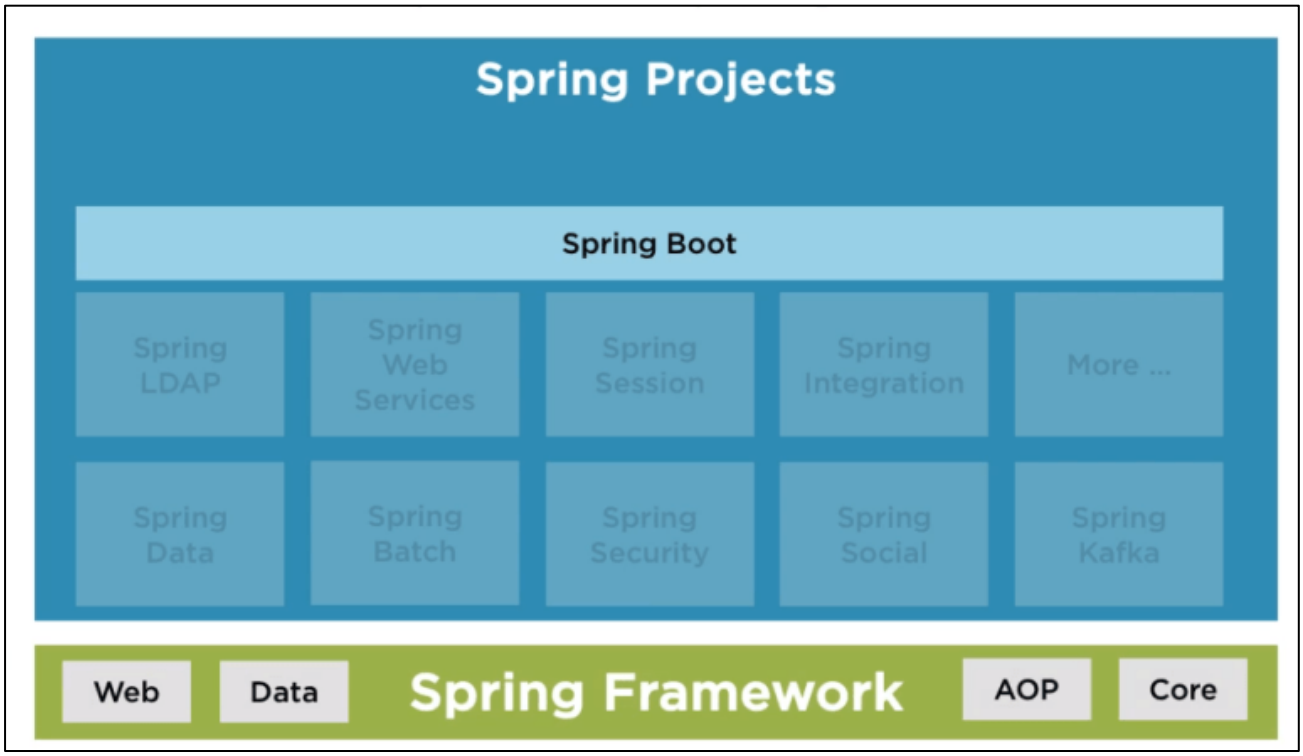
Programski jezik Java nastao je 1995. godine. Prema [3] razvio ga je James Gosling u tvrtci Sun Microsystems. Ovaj objektno orijentiran programski jezik nastao je s namjerom olakšanja posla razvojnih programera jer koristi mali broj implementacijskih ovisnosti (engl. implementation dependencies). Zamišljen je na način da se programski kôd jednom napiše, a aplikacija se može pokretati na bilo kojoj platformi koja podržava Javu. Java programski jezik ima sličnu sintaksu programskim jezicima C i C++ (Odsječak programskog kôda 2.1.), a danas svoju primjenu nalazi gotovo svugdje. Tako je Java programski jezik temelj razvijanja web-aplikacija pomoću Spring Boot tehnologije.

```
public static void main(String[] args) {
    System.out.println("Hello World");
}
```

Odsječak programskog kôda 2.1. Prikaz sintakse Java programskog Jezika

2.3. Spring Boot tehnologija

Spring Boot tehnologija omogućava bržu i lakšu izradu projekta u Spring okviru. U staroj Spring tehnologiji sve ovisnosti morale su se pisati ručno u XML opisnom jeziku. Za stvaranje novih projekata morale su se pisati ručno od početka bez mogućnosti automatskog generiranja kôda. Arhitekturu Spring Boot aplikacije možemo prikazati kao ovisnosti dodane na temelj Spring okvira (Slika 2.3.).



Slika 2.3. Prikaz arhitekture Spring Boot projekta

Prema [4] prednosti koje je unijela Spring Boot tehnologija su:

- Automatsko generiranje kôda ovisnosti (engl. dependency)
- Inteligentna auto-konfiguracija
- Ugrađen Tomcat
- Automatska konfiguracija Spring paketa gdje god je moguće
- Potpuno izbačena konfiguracija XML-om

Kako bi stvaranje Spring projekata bilo još lakše koristi se web-stranica Spring Initilizr. Na toj stranici se generiraju sve potrebne ovisnosti s obzirom na odabrane pakete. Ovakvo generiranje Spring projekata našlo je svoj put i u IntelliJ te korištenjem ovog razvojnog okruženja nije potrebno odlaziti na prethodno navedenu stranicu. Sve ovisnosti možemo naći u datoteci *pom.xml* (Slika 2.4.).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.6.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>demo</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

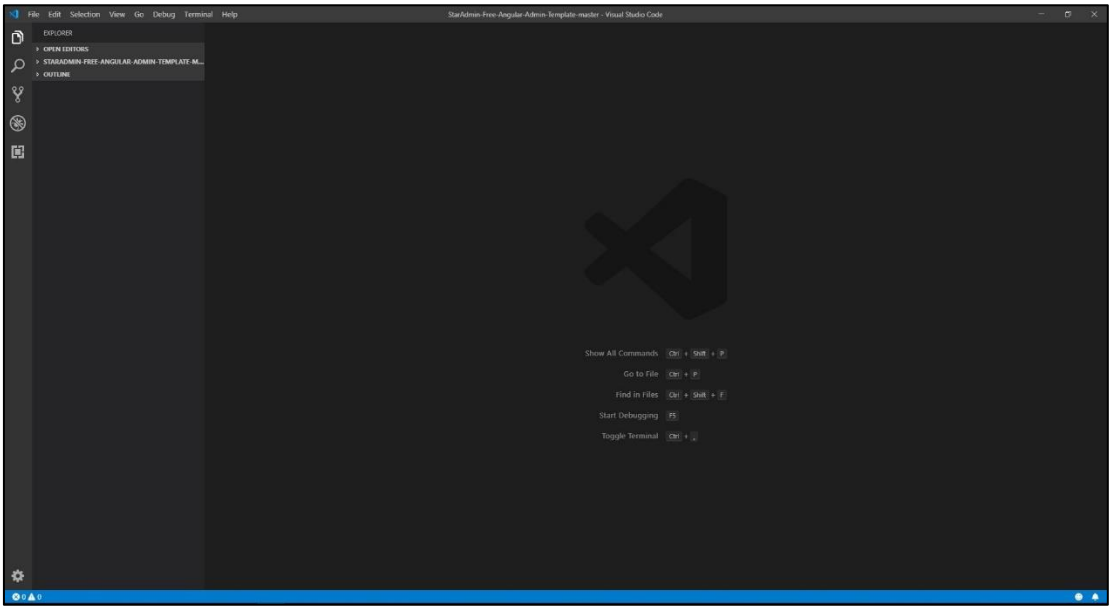
Slika 2.4. Prikaz sadržaja *pom.xml* datoteke

Korištenje ove tehnologije isto tako ima svoje nedostatke. Tako zbog korištenja automatski stvorenog kôd, Spring Boot često stvara ovisnosti koje nisu korištene u projektu i time usporava vrijeme potrebno za pokretanje i razvoj projekata. Isto tako ova tehnologija skriva detalje kako su ovisnosti stvorene i povezane te je za potpuno razumijevanje pozadinskog rada stvorenih ovisnosti

potrebno dodatno proučiti dokumentaciju te se zbog toga javlja problem ograničenja kontrole nad aplikacijom.

2.4. Visual Studio Code

Visual Studio Code je program za uređivanje izvornog kôda programa. Razvio ga je Microsoft za uporabu na Windows, Linux i macOS operacijskim sustavima. S obzirom na tip datoteke Visual Studio Code, za razliku od ostalih programa za uređivanje teksta, ima ugrađene različite boje kako bi kôd bio čitljiv.

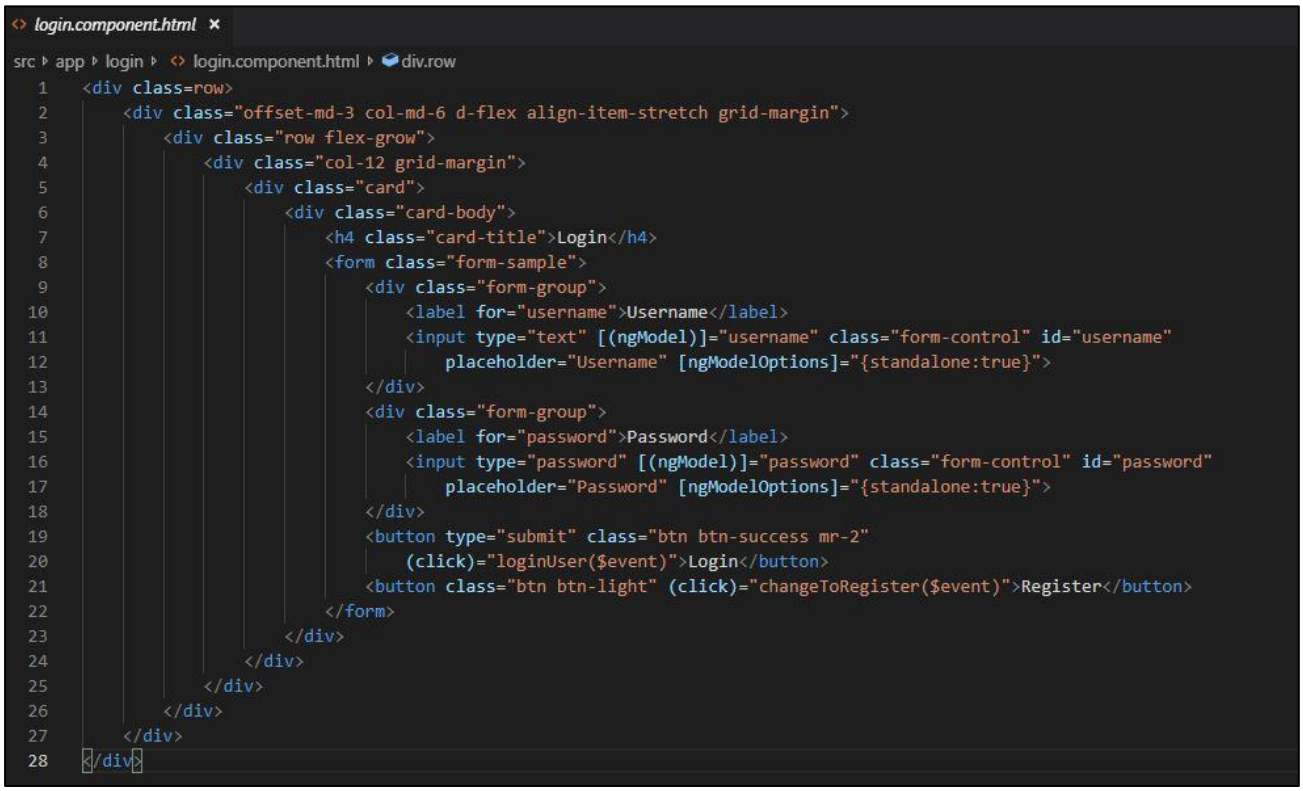


Slika 2.5. Korisničko sučelje programa Visual Studio Code

2.5. Angular tehnologija

Tehnologija Angular temeljena je na JavaScriptu. Ovaj web okvir (engl. web framework) koristi se za stvaranje korisničkog sučelja web-aplikacija. Za korištenje ove tehnologije potrebno je poznavati opisni jezik HTML i programski jezik TypeScript.

HTML ili Hypertext Markup Language je opisni jezik potreban za prikaz sadržaja web-stranica pomoću web preglednika. Pomoću sintakse ovog jezika (Odsječak programskog kôda 2.2.) opisuje se izgled stranice te se zatim taj kôd prikazuje kao web stranica. Zbog jednostavnosti izgleda web-stranica pisanih u ovom jeziku često se povezuje sa jezicima kao što su JavaScript i CSS.



Odsječak programskog kôda 2.2. Prikaz sintakse opisnog jezika HTML

TypeScript je programski jezik otvorenog kôda. Prema [5] ovaj jezik se još naziva i „nadskupom“ JavaScripte te su usko povezani. Tako se svi programi pisani u JavaScriptu mogu pokretati kao TypeScript programi. Ovaj programski jezik svoju primjenu nalazi u stvaranju JavaScript aplikacija i na klijentskoj i na serverskoj strani.

Korištenjem Angular tehnologije odmah dolazimo do nedostatka u obliku poznavanja dva različita jezika. Zbog toga nailazimo i na problem učenja samih tehnologija potrebnih za izradu aplikacija u Angularu. Isto tako kôd za Angular aplikacije piše se u programu za obrađivanje teksta,

a sama aplikacija se pokreće preko konzole te je potrebno znanje naredbi potrebnih za pokretanje i instaliranje potrebnih ovisnosti kako bi aplikacija radila.

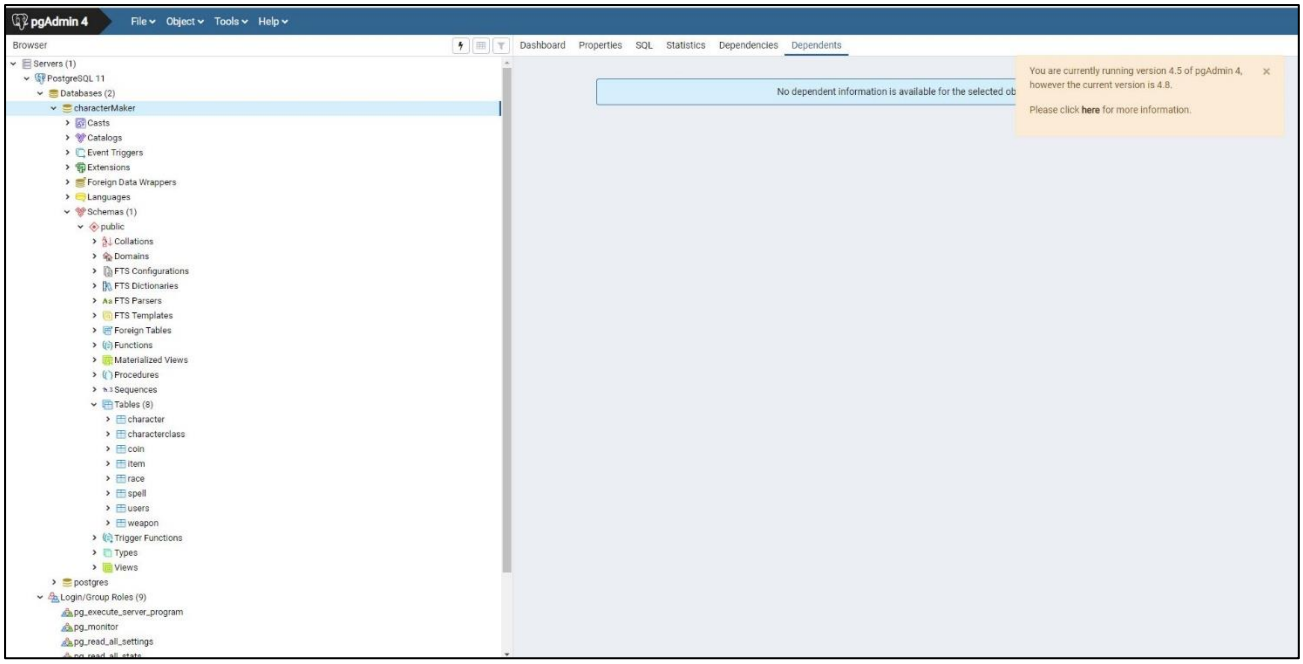
Prednosti koje nudi ova tehnologiju su lako čitljiv kôd te lako ponovno iskorištavanje istog kôda. Angular omogućuje lako jedinično testiranje(engl. unit testing) te održavanje aplikacija. Isto tako korištenje TypeScript programskog jezika nudi nam razne pogodnosti kao što su čišći kôd te programerima osigurava više alata pri izradi web-aplikacija.

2.6. Bootstrap

Bootsrap tehnologija je besplatna; CSS okvir otvorenog kôda korišten za stvaranje responzivnih korisničkih sučelja web-stranica i aplikacija. Sastoji se od CSS, a nekada i JavaScript, predložaka za dizajn web-stranica. Ovi predlošci najčešće sadrže dizajn za forme, dugmad i navigaciju unutar web-stranica.Prema [6] CSS Bootstrap sadrži predefiniran kôd za izgled elemenata stranice, veličinu, boju i položaj. Ova tehnologija ubrzava stvaranje web-stranica te je jedna od najraširenijih kada je u pitanju dizajn web-stranice.

2.7. Postgres

Tehnologija Postgres, poznata i kao PosgreSQL, je besplatan sustav upravljanja relacijskih baza podataka. Otvorenog je kôda, a svoju primjenu nalazi u jako širokom području, od lokalnih web-stranica do velikih skladišta. Ova tehnologija sadrži podršku za tri jezika a to su: SQL, jezik potreban za korištenje i stvaranje baza podataka, PostgreSQL koji ima sličnu sintaksu kao i Oracleov proceduralan jezik za SQL, te C.



Slika 2.6. Korisničko sučelje Postgresa

3. IZRADA WEB-STRANICE

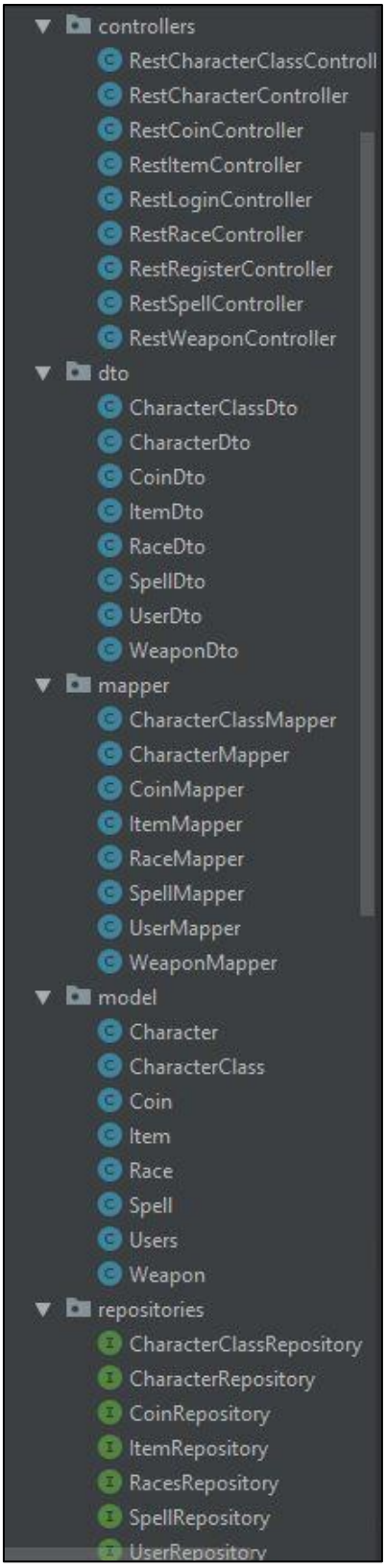
Proces izrade web-stranice počinje od izrade pozadinskog dijela aplikacije korištenjem Spring Boot tehnologije i Java programskog jezika. Prilikom stvaranja projekta odabiru se potrebne ovisnosti koje su implementirane u projekt te se njihov automatski generiran kôd nalazi u pom.xml datoteci (Odsječak programskog kôda 3.1.). Nakon toga stvaraju se klase koje predstavljaju model baze podataka, odnosno tablice unutar same baze. Sljedeći korak je stvoriti klase koje će predstavljati datoteke koje će se sa pozadinskog dijela slati na korisničko sučelje i obratno, a logiku iza tog procesa držat će tzv. *mapperi*. Komunikaciju s bazom podataka održavat će servisi i repozitoriji. Za svaki upit nad bazom stvoren je i *REST*[1] *point* kojim će upravljati određeni kontroleri. Sve to potrebno je odvojiti u nove pakete kako bi struktura projekta bila čitljiva (Slika 3.1.)

Nakon stvaranja pozadinske aplikacije stvoreno je korisničko sučelje te se nakon toga sučelje povezalo s pozadinskom aplikacijom.

[1] engl. Representational State Transfer (REST)- stil softverske arhitekture koji se sastoji od ograničenja koja služe za stvaranje web servisa

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.4.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.sertic</groupId>
    <artifactId>charactermaker</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>CharacterMaker</name>
    <description>Character Maker Spring Boot</description>

    <properties>
        <java.version>11</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

Odsječak programskog kôda 3.1. Prikaz sadržaja *pom.xml* datoteke



Slika 3.1. Prikaz sadržaja paketa projekta

3.1. Model baze podataka

Prije početka stvaranja modela potrebno je napisati postavke vezane uz samu bazu podataka. Korištenjem tehnologije Postgres mijenjaju se postavke projekta u datoteci *application.properties* (Odsječak programskog kôda 3.2.). Time je osigurano da će Spring Boot tehnologija pravilno pretvoriti model baze podataka pisane u Java programskom jeziku u kôd potreban za stvaranje baze podataka u tehnologiji Postgres.

```
hibernate.show_sql = true
spring.datasource.url= jdbc:postgresql://localhost:5432/characterMaker
spring.datasource.username=postgres
spring.datasource.password=ivan123
spring.jpa.properties.hibernate.temp.use_jdbc_metadata_defaults = false
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQL9Dialect
spring.jpa.hibernate.ddl-auto=create-drop
spring.datasource.platform=postgres
spring.datasource.initialization-mode=always
```

Odsječak programskog kôda 3.2. Prikaz sadržaja datoteke application.properties

Zatim je potrebno stvoriti klase koje će predstavljati tablice baze podataka. Te klase nalaze se u novostvorenom paketu model. Kako bi se podatci iz klase prenijeli u tablice baze podataka svaka klasa se anotira anotacijom „@Entity“ kako bi Spring Boot prepoznao te klase kao klase entiteta baze podataka. Isto tako u ove klase navode se svi stupci tablice, veze između tablica te metode za dohvaćanje i postavljanje vrijednosti varijabli odnosno vrijednosti stupaca tablice.

```java
package com.sertic.charactermaker.model;

import org.hibernate.annotations.GenericGenerator;

import javax.persistence.*;
import java.util.Objects;
import java.util.UUID;

@Entity
public class Item {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long Id;

    @GeneratedValue(generator = "uuid2")
    @GenericGenerator(name = "uuid2", strategy = "org.hibernate.id.UUIDGenerator")
    private UUID externalItemId;

    private String name;

    private Long amount;

    @ManyToOne(cascade = CascadeType.ALL)
    private Character character = new Character();

    public Long getAmount() {
        return amount;
    }

    public void setAmount(Long amount) {
        this.amount = amount;
    }

    public Character getCharacter() {
        return character;
    }

    public void setCharacter(Character character) {
        this.character = character;
    }

    public Long getId() {
        return Id;
    }

    public void setId(Long id) {
        Id = id;
    }
```

Odsječak programskog kôda 3.3. Primjer klase entiteta *Item*

3.2. Stvaranje datoteka za korisničko sučelje i njihovo upravljanje

Nakon stvaranja modela potrebno je stvoriti klase koje će predstavljati datoteke primljene odnosno poslane na korisničko sučelje. Ove klase nalaze se unutar paketa pod nazivom *dto*. Ove klase su identične klasama modela baze (Odsječak programskog kôda 3.4.) jer je njihova uloga da drže vrijednosti koje će se s korisničkog sučelja slati na pozadinsku aplikaciju i obrnuto.

```java
package com.sertic.charactermaker.dto;

import java.util.UUID;

public class SpellDto {

    private Long idDto;

    private UUID externalSpellIdDto;

    private String nameDto;

    private String typeDto;

    private Long spellLevelDto;

    private CharacterDto characterDto;


    public Long getIdDto() {
        return idDto;
    }

    public void setIdDto(Long idDto) {
        this.idDto = idDto;
    }

    public UUID getExternalSpellIdDto() {
        return externalSpellIdDto;
    }

    public void setExternalSpellIdDto(UUID externalSpellIdDto) {
        this.externalSpellIdDto = externalSpellIdDto;
    }

    public String getNameDto() {
        return nameDto;
    }

    public void setNameDto(String nameDto) {
        this.nameDto = nameDto;
    }

    public String getTypeDto() {
        return typeDto;
    }

    public void setTypeDto(String typeDto) {
        this.typeDto = typeDto;
    }
```

Odsječak programskog kôda 3.4. Prikaz klase paketa *dto*

Nakon stvaranja klasa paketa dto, potrebno je stvoriti klase koje će omogućiti pretvaranje sadržaja korisničkog sučelja, dto klasa, u sadržaj spreman za pohranu u bazu, klase paketa model. Stvaranjem klasa paketa *mapper* (Odsječak programskog kôda 3.5.) omogućuje se stvaranje modela iz korisničkog sučelja i obratno. Klase ovog paketa označavaju se anotacijom „@Component“.

```java
package com.sertic.charactermaker.mapper;

import com.sertic.charactermaker.dto.CoinDto;
import com.sertic.charactermaker.model.Coin;
import org.springframework.stereotype.Component;

import java.util.UUID;

@Component
public class CoinMapper {

    public void update(Coin entity, CoinDto dto){
        entity.setAmount(dto.getAmmountDto());
        entity.setCoinType(dto.getCoinTypeDto());
    }

    public CoinDto toDto(Coin entity){
        CoinDto dto = new CoinDto();
        dto.setAmmountDto(entity.getAmount());
        dto.setIdDto(entity.getId());
        dto.setExternalCoinIdDto(entity.getExternalId());
        dto.setCoinTypeDto(entity.getCoinType());

        return dto;
    }

    public Coin createEntity(CoinDto dto){
        Coin entity = new Coin();
        entity.setExternalId(UUID.randomUUID());
        update(entity,dto);

        return entity;
    }
}
```

Odsječak programskog kôda 3.5. Prikaz klase paketa mapper

3.3. Upravljanje bazom podataka

Prije stvaranja upita nad bazom te njihovih korištenja u samom kôdu, popunjene su dvije tablice: tablica *CharacterClass* i *Race* zbog stvaranja padajućeg izbornika koji sadrži stalne podatke.

SQL kôd za popunjavanje ovih baza nalazi se u datoteci data-postgres.sql (Odsječak programskog kôda 3.6.).

```
CREATE EXTENSION IF NOT EXISTS "uuid-ossp"

INSERT INTO race(id,external_race_id,name) VALUES('1',uuid_generate_v4(),'DWARF')
INSERT INTO race(id,external_race_id,name) Values('2',uuid_generate_v4(),'ELF')
INSERT INTO race(id,external_race_id,name) VALUES('3',uuid_generate_v4(),'HUMAN')
INSERT INTO race(id,external_race_id,name) VALUES('4',uuid_generate_v4(),'HALF-ORC')
INSERT INTO race(id,external_race_id,name) VALUES('5',uuid_generate_v4(),'GOBLIN')
INSERT INTO race(id,external_race_id,name) VALUES('6',uuid_generate_v4(),'HOBGOBLIN')
INSERT INTO race(id,external_race_id,name) VALUES('7',uuid_generate_v4(),'ORC')
INSERT INTO race(id,external_race_id,name) VALUES('8',uuid_generate_v4(),'LIZARDFOLK')
INSERT INTO race(id,external_race_id,name) VALUES('9',uuid_generate_v4(),'CENTAUR')
INSERT INTO race(id,external_race_id,name) VALUES('10',uuid_generate_v4(),'MINOTAUR')

INSERT INTO characterclass(id,external_character_class_id,name) VALUES ('1',uuid_generate_v4(),'FIGHTER')
INSERT INTO characterclass(id,external_character_class_id,name) VALUES ('2',uuid_generate_v4(),'PALADIN')
INSERT INTO characterclass(id,external_character_class_id,name) VALUES ('3',uuid_generate_v4(),'BARD')
INSERT INTO characterclass(id,external_character_class_id,name) VALUES ('4',uuid_generate_v4(),'SORCERER')
INSERT INTO characterclass(id,external_character_class_id,name) VALUES ('5',uuid_generate_v4(),'WARLOCK')
INSERT INTO characterclass(id,external_character_class_id,name) VALUES ('6',uuid_generate_v4(),'CLERIC')
INSERT INTO characterclass(id,external_character_class_id,name) VALUES ('7',uuid_generate_v4(),'RANGER')
INSERT INTO characterclass(id,external_character_class_id,name) VALUES ('8',uuid_generate_v4(),'ROGUE')
INSERT INTO characterclass(id,external_character_class_id,name) VALUES ('9',uuid_generate_v4(),'MONK')
INSERT INTO characterclass(id,external_character_class_id,name) VALUES ('10',uuid_generate_v4(),'WIZARD')
```

Odsječak programskog kôda 3.6. Prikaz sadržaja datoteke data-postgres.sql

Spring Boot upite na bazu podataka stvara pomoću sučelja na način da se upit stvara kôd prevođenja aplikacije s obzirom na ime metode. Upite koji nisu standardni potrebno je napisati unutar sučelja paketa *repository*. Svaka tablica ima svoje sučelje te se u to sučelje pišu deklaracije metoda koje predstavljaju upite. Kada se piše korisnički upit, metoda se anotira sa „@Query“ te se u zagradu piše upit. Sučelje je anotirano s „@Repository“.

```java
package com.sertic.charactermaker.repositories;

import com.sertic.charactermaker.model.Item;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import java.util.List;
import java.util.UUID;

@Repository
public interface ItemRepository extends CrudRepository<Item, Long> {

    Item findByExternalItemId(UUID externalItemId);

    @Query("SELECT i FROM Item i WHERE i.character.externalCharacterId = ?1")
    List<Item> getByExternalCharacterId(UUID externalCharacterId);

    @Query("SELECT i FROM Item i WHERE i.character.user.externalUserId = ?1 and " +
            "i.character.externalCharacterId = ?2 and i.externalItemId = ?3")
    Item getItemByExternalUserAndCharacterId(UUID externalUserId, UUID externalCharacterId,
                                             UUID externalItemId);

    @Modifying
    @Query("DELETE FROM Item i WHERE i.externalItemId = ?1")
    void deleteByExternalWeaponId(UUID externalWeaponId);
}
```

Odsječak programskog kôda 3.7. Primjer pisanja korisničkih upita

Stvaranje servisa i njihovih implementacija osigurava se korištenje prethodno deklariranih upita. Svaki servis ima svoje sučelje u kojem su definirane metode te klasu implementacije u kojoj se pomoću prethodno stvorenih repozitorija metode servisa povezuju s metodama odgovarajućih repozitorija. Svaka klasa servisa anotirana je s „@Service“ .

```java
package com.sertic.charactermaker.services;

import com.sertic.charactermaker.model.Coin;
import com.sertic.charactermaker.repositories.CoinRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import javax.transaction.Transactional;
import java.util.List;
import java.util.UUID;

@Service
@Transactional
public class CoinServiceImpl implements CoinService{

    @Autowired
    private CoinRepository coinRepo;

    @Override
    public Coin getCoinByExternalCoinId(UUID externalCoinId) {
        return coinRepo.findByExternalId(externalCoinId);
    }

    @Override
    public List<Coin> getAllCoinsOfACharacter(UUID externalCharacterId) {
        return coinRepo.getCoinByExternalCharacterId(externalCharacterId);
    }

    @Override
    public Coin getCoinFromUserAndCharacter(UUID externalUserId, UUID externalCharacterId, UUID externalCoinId) {
        return coinRepo.getCoinByExternalUserAndCharacterId(externalUserId,externalCharacterId,externalCoinId);
    }

    @Override
    public void update(UUID externalCoinId, Coin coin) {
        coinRepo.save(coin);
    }

    @Override
    public void deleteByExternalId(UUID externalCoinId) {
        coinRepo.deleteByExternalCoinId(externalCoinId);
    }

    @Override
    public void create(Coin coin) {
        coinRepo.save(coin);
    }
}
```

Odsječak programskog kôda 3.8. Prikaz implementacije servisa

3.4 Stvaranje REST Pointa

Za svaki upit nad bazom potrebno je stvoriti *REST point*. To su zapravo krajnje točke web-stranice na kojima se odvijaju upiti na bazi podataka. Svaki *REST point* ima svoju putanju koja je definirana u kontrolerima. Kada se putanja pozove kontroleri pozivaju servise i *mappere* te preko servisa komuniciraju s bazom podataka te dohvaćaju, postavljaju, mijenjaju i brišu podatke iz nje. Svaki kontroler anotiran je s „@RestController“, a svaka putanja s „@RequestMapping“. Za stvaranje korisnika stvoreni su posebni *REST pointi* koji služe posebno za prijavljivanje, a posebno za registraciju na web-stranicu (Odsječak programskog kôda 3.10.).

```java
@RestController
@RequestMapping("/user")
public class RestItemController {

    @Autowired
    private CharacterService characterService;

    @Autowired
    private ItemService itemService;

    @Autowired
    private ItemMapper itemMapper;

    public static final Logger logger = LoggerFactory.getLogger(RestCharacterController.class);

    //Create Item
    @CrossOrigin(origins = "http://localhost:4200")
    @RequestMapping(value = "/{externalUserId}/character/{externalCharacterId}/item", method = RequestMethod.POST)
    public ResponseEntity<String> createItem(@PathVariable("externalUserId") UUID externalUserId,
                                             @PathVariable("externalCharacterId")UUID externalCharacterId,
                                             @RequestBody ItemDto dto){
        final Character character = characterService.GetCharacterFromUser(externalCharacterId,externalUserId);

        if(character == null){
            return ResponseEntity.notFound().build();
        }else{
            final Item item = itemMapper.createEntity(dto);

            item.setCharacter(character);
            character.getItems().add(item);

            itemService.create(item);

            return ResponseEntity.ok().build();
        }
    }

    //Update Item
    @CrossOrigin(origins = "http://localhost:4200")
    @RequestMapping(value = "/{externalUserId}/character/{externalCharacterId}/item/{externalItemId}", method = RequestMethod.PUT)
    public ResponseEntity<String> updateItem(@PathVariable("externalUserId")UUID externalUserId,
                                             @PathVariable("externalCharacterId")UUID externalCharacterId,
                                             @PathVariable("externalItemId")UUID externalItemId,
                                             @RequestBody ItemDto dto){
```

Odsječak programskog kôda 3.9. Prikaz kontrolera za klasu Item

```java
package com.sertic.charactermaker.controllers;

import com.sertic.charactermaker.dto.UserDto;
import com.sertic.charactermaker.mapper.UserMapper;
import com.sertic.charactermaker.model.Users;
import com.sertic.charactermaker.services.UserService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.security.NoSuchAlgorithmException;
import java.util.Map;

@RestController
@RequestMapping("/register")
public class RestRegisterController {

    @Autowired
    private UserMapper userMapper;

    @Autowired
    private UserService userService;

    public static final Logger logger = LoggerFactory.getLogger(RestCharacterController.class);


    @CrossOrigin(origins = "http://localhost:4200")
    @RequestMapping(value = "", method = RequestMethod.POST)
    public ResponseEntity<Map<String,String>> register(@Valid @RequestBody UserDto dto){
        try{
            Users user = userMapper.createEntity(dto);
            userService.createUser(user);

            return ResponseEntity.ok().build();
        }catch (NoSuchAlgorithmException e){
            e.printStackTrace();
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(Map.of( k1: "error", v1: "Hashing went wrong"));
        }
    }
}
```

Odsječak programskog kôda 3.10. Prikaz kontrolera za registraciju

3.5. Stvaranje korisničkog sučelja

Korisničko sučelje stvoreno je tehnologijom Angular. Proces stvaranja sučelja ostvaren je na način da se za svaki *REST point* stvori web-stranica te se pomoću Boostrap tehnologije doda izgled. Pomoću Angulara upravlja se kretnjama između stranica.

Svaka komponenta stranice predstavljena je odgovarajućim HTML i TypeScript kôdom. Nakon stvaranja izgleda komponente u HTMLu, pomoću TypeScript datoteke omogućena je

funkcionalnost svih stranica. Isto tako svaki od prethodno navedenih *rest pointa* potrebno je implementirati u korisničkom sučelje pomoću njihovih putanja (Odsječak programskog kôda 3.11.), a svaku stvorenu komponentu potrebno je implementirati u sam projekt.

```
import { Injectable} from '@angular/core';
import { HttpClient, HttpHeaders} from '@angular/common/http';
import { Observable} from 'rxjs/Observable';


const httpOptions ={
    headers: new HttpHeaders({'Content-Type':'application/jason'})
};

@Injectable()
export class CharactermakerService{
    constructor(private http:HttpClient){}

    /* For Characters*/

    getClasses(){
        return this.http.get('http://localhost:8080/class');
    }

    getOneClass(id:any){
        return this.http.get('http://localhost:8080/class/'+id);
    }

    getRaces(){
        return this.http.get('http://localhost:8080/race');
    }

    getOneRace(id:any){
        return this.http.get('http://localhost:8080/race/'+id);
    }

    getAllCharacters(){
        return this.http.get('http://localhost:8080/user/'+localStorage.getItem("token")+'/character')
    }

    getOneCharacter(id:any){
        return this.http.get('http://localhost:8080/user/'+localStorage.getItem("token")+'/character/'+id);
    }

    createCharacter(body){
        return this.http.post('http://localhost:8080/user/'+localStorage.getItem("token")+'/character', body);
    }

    updateCharacter(id:any,body){
       return this.http.put('http://localhost:8080/user/'+localStorage.getItem("token")+'/character/'+id, body);
    }
```

Odsječak programskog kôda 3.11. Prikaz implementiranja rest pointa u korisničkom sučelju

Nakon implementiranja rest pointa korisničko sučelje povezano je s pozadinskom aplikacijom tako što se za svaku web-stranicu komponente u TypeScript datoteci poziva funkcija određenog rest pointa.

```
<div class="row">
    <div class="col-12 grid-margin stretch-card">
        <div class="card">
            <div class="card-body">
                <h4 class="card-title">Aromory</h4>
                <div class="table-responsive">
                    <table class="table table-striped">
                        <thead>
                            <tr>
                                <th>
                                    Name
                                </th>
                                <th>
                                    Attack Bonus
                                </th>
                                <th>
                                    Damage Type
                                </th>
                            </tr>
                        </thead>
                        <tbody>
                            <tr *ngFor="let weapon of listOfWeapons">
                                <td>
                                    {{weapon.name}}
                                </td>
                                <td>
                                    {{weapon.attackBonus}}
                                </td>
                                <td>
                                    {{weapon.damageType}}
                                </td>
                                <td>
                                    <button type="button" class="btn btn-primary btn-fw"
                                        (click)="changeToUpdateWeapon($event,weapon)">Update Weapon</button>
                                </td>
                                <td>
                                    <button type="button" class="btn btn-danger btn-fw" (click)="deleteWeapon($event,weapon)">Drop Weapon</button>
                                </td>
                            </tr>
                        </tbody>
                    </table>
                </div>
            </div>
        </div>
    </div>
    <div class="col-lg-6 grid-margin stretch-card">
        <button type="button" class="btn btn-primary btn-fw" (click)="changeToCreateWeapon($event)">Add Weapon</button>
    </div>
</div>
```

Odsječak programskog kôda 3.12. Prikaz HTML kôda komponente korisničkog sučelja

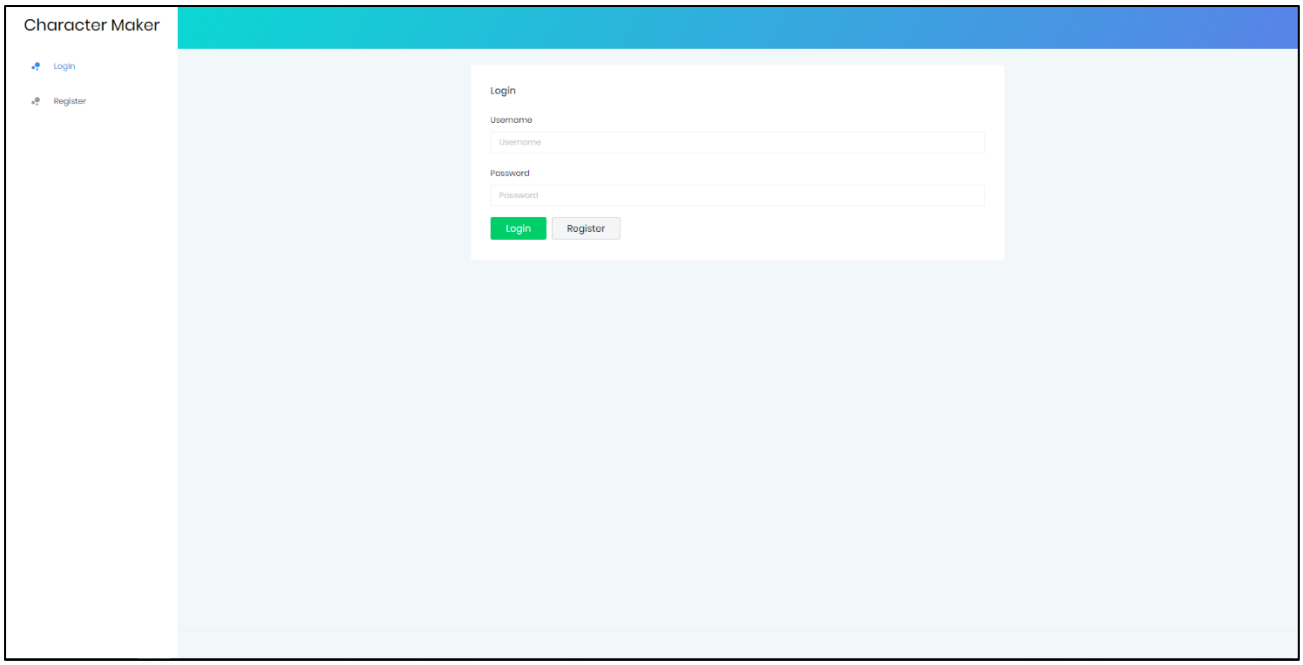
```
@Component(
    {
        selector: 'app-weapons',
        templateUrl: './weapons.component.html',
        styleUrls: ['./weapons.component.scss']
    }
)

export class WeaponsComponent implements OnInit{

    constructor(private router: Router, private service: CharactermakerService, private route: ActivatedRoute){}

    private characterId:String;
    public listOfWeapons: any[];
    ngOnInit(){
        this.listOfWeapons =[];
        this.characterId= this.route.snapshot.paramMap.get('characterId');
        this.getAllWeapons();
    }


    private getAllWeapons(){
        this.service.getAllWeapons(this.characterId).subscribe(
            (data: any[])=>{
                this.listOfWeapons = data;
            }
        );
    }

    public changeToUpdateWeapon(event, weapon:any){
        this.router.navigate(['/updateweapon',weapon['externalWeaponIdDto'],this.characterId]);
    }

    public changeToCreateWeapon(event){
        this.router.navigate(['/createweapon', this.characterId]);
    }

    public deleteWeapon(event, weapon:any){
        this.service.deleteWeapon(this.characterId, weapon['externalWeaponIdDto']).subscribe(
            data=>{
                this.getAllWeapons();
            }
        );
    }
```

Odsječak programskog kôda 3.13. Prikaz TypeScript datoteke komponente korisničkog sučelja

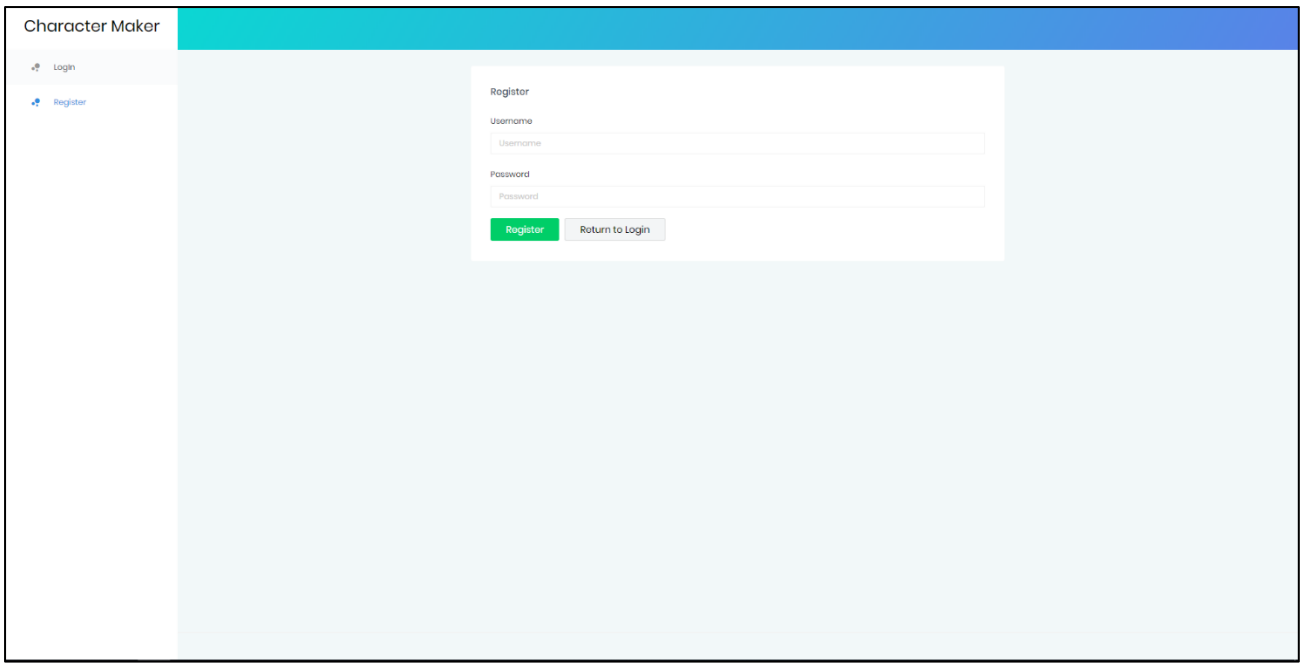
4. PRIKAZ KORISNIČKOG RJEŠENJA

U ovom poglavlju prikazan je rad aplikacije te je svaki korak popraćen sa snimkama ekrana koje prikazuju kako web-stranica izgleda.

Prilikom pokretanja web-stranice korisniku se prikazuje forma koja od njega traži prijavu (Slika 4.1.). Ukoliko korisnik ne postoji korisnik se mora registrirati tako što ispunjava formu registracije(Slika 4.2.), nakon čega je vraćen na čega je vraćen na stranicu za prijavu gdje popunjava formu sa svojim korisničkim imenom i zaporkom.

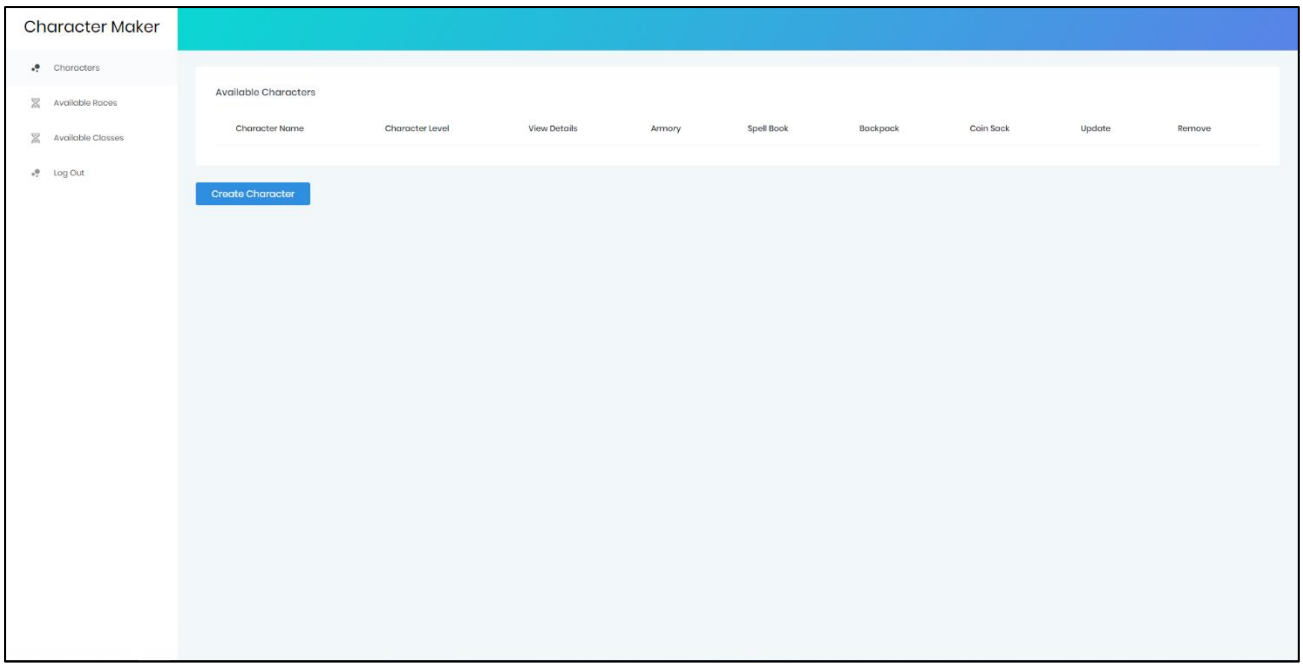


Slika 4.1. Prikaz stranice za prijavu

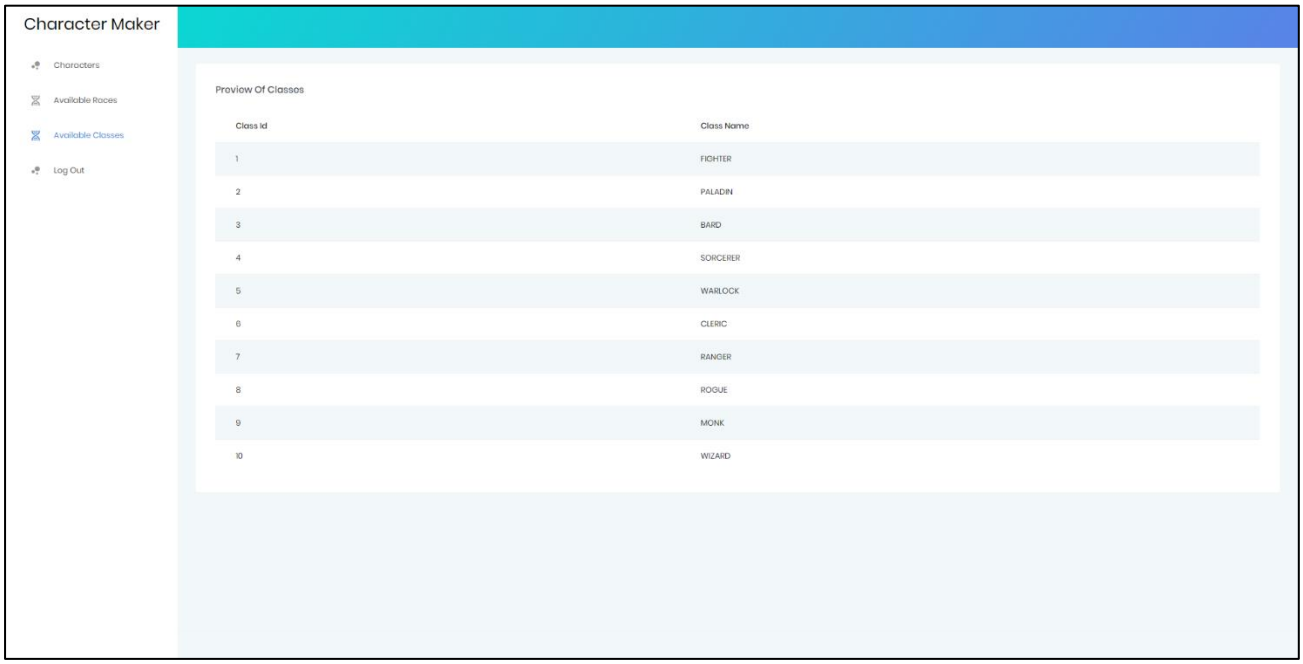


Slika 4.2. Prikaz stranice za registraciju

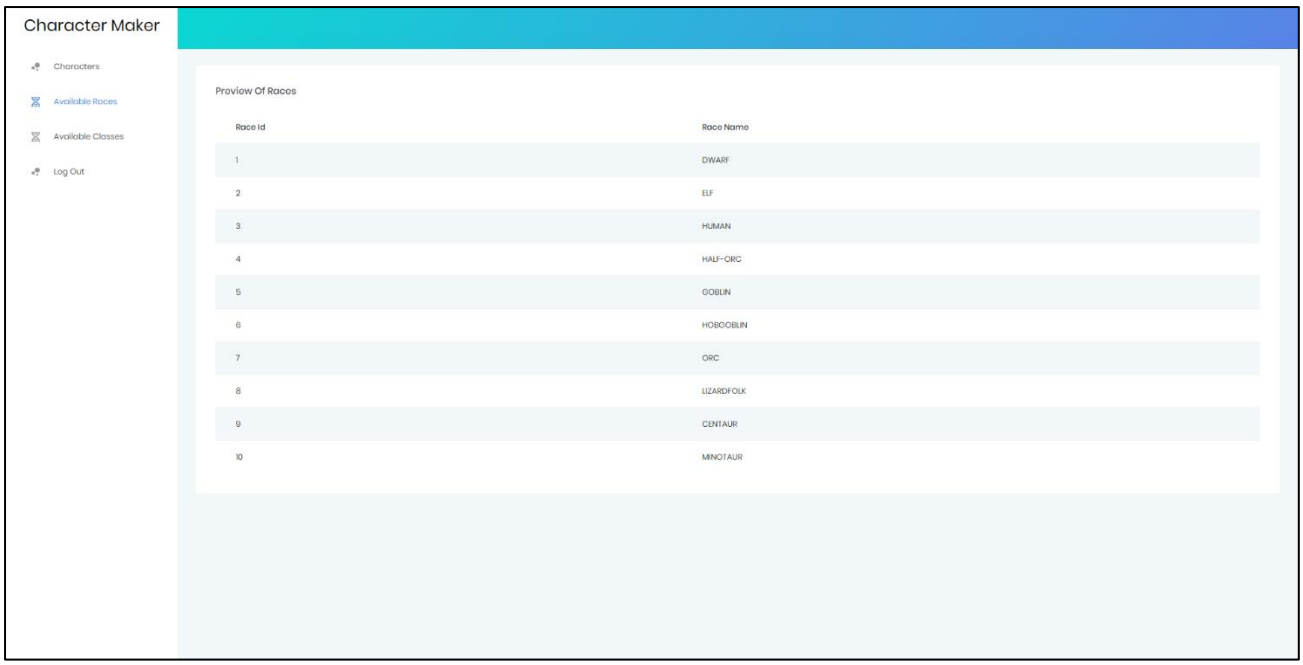
Ukoliko je prijava uspješna korisniku se otvara stranica sa njegovim avatarima, koja je prazna ukoliko je to njegova prva prijava (Slika 4.3.). Prije stvaranja svog lika, korisniku su pružene opcije pregleda raspoloživih klasa(Slika 4.4) i rasa (Slika 4.5). Isto tako korisnik ima opciju odjave sa stranice.



Slika 4.3. Prikaz stranice sa listom avatara

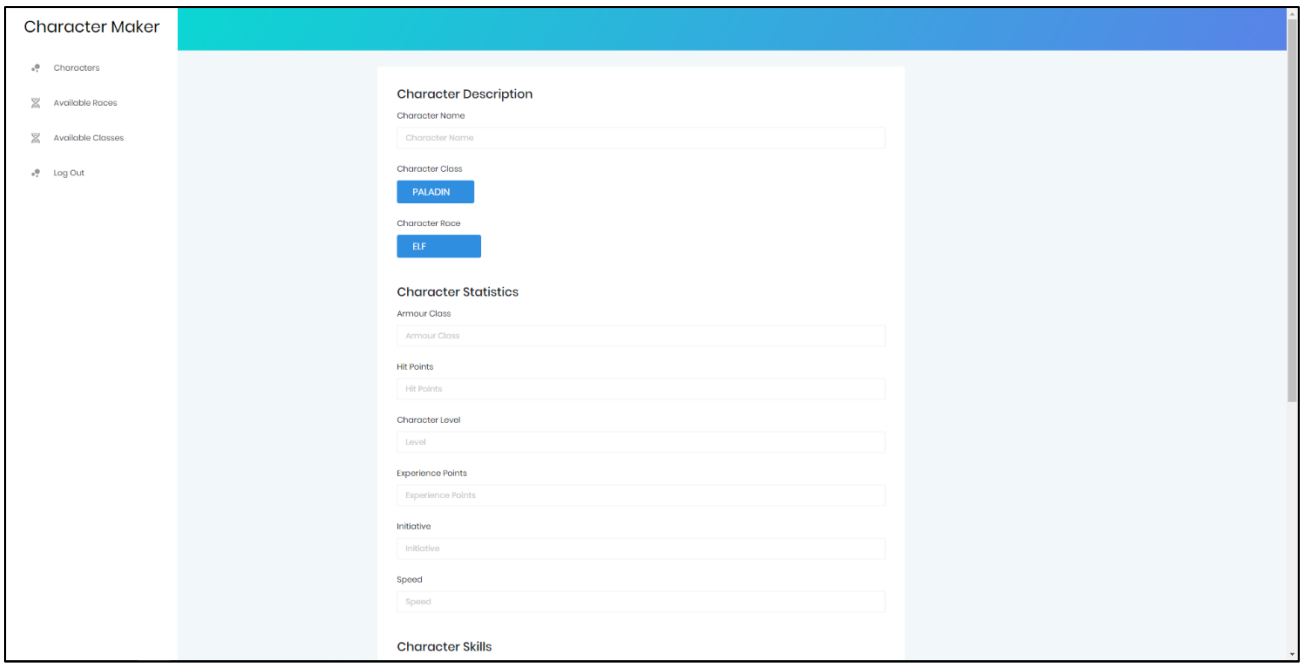


Slika 4.4. Prikaz raspoloživih klasa



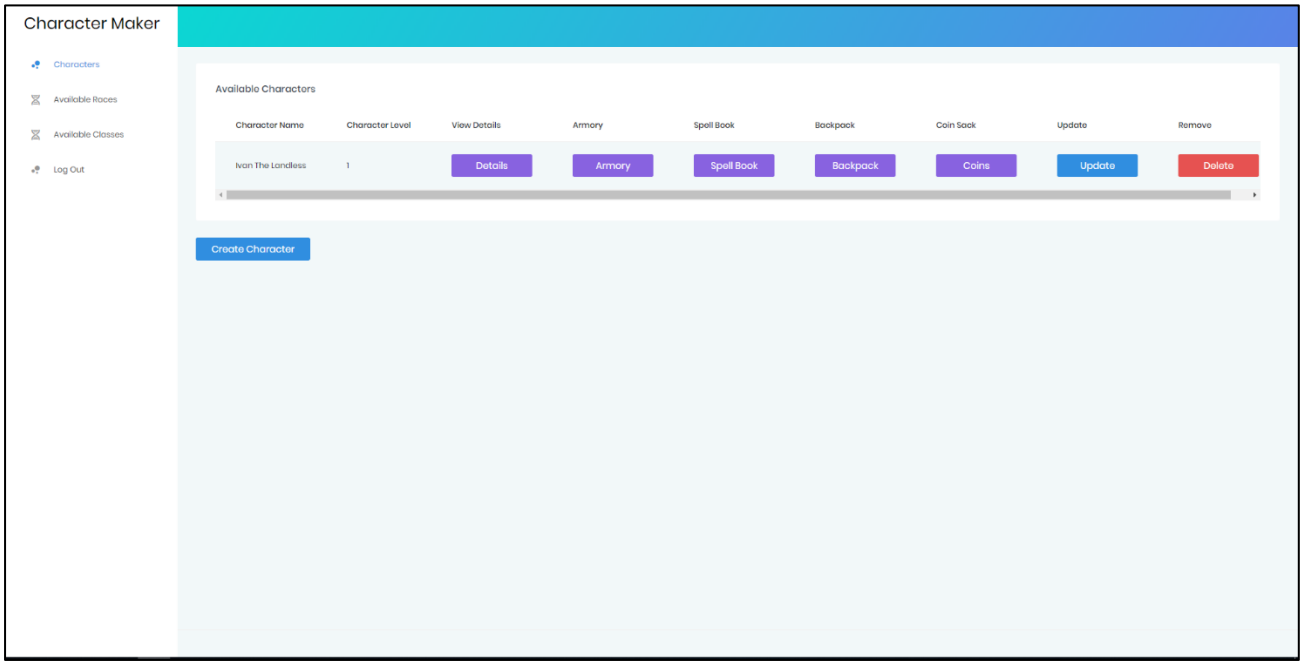
Slika 4.5. Prikaz stranice s raspoloživim rasama

Pritiskom na gumb *Create Character*, korisniku se otvara forma koju je potrebno popuniti kako bi se stvorio novi lik. Nakon popunjavanja forme pritiskom na gumb *submit* novi lik se sprema u bazu podataka. Korisnik može imati više od jednog lika. Prikaz detalja lika omogućen je pritiskom na gumb *Details*, nakon čega se na ekranu prikazuje stranica sa svim detaljima lika. Ukoliko korisnik želi izmijeniti lika to vrši pritiskom na gumb *Update Character* gdje mu se otvara forma identična formi za stvaranje lika, ali tu su sva polja popunjena prethodnim odabirima korisnika.



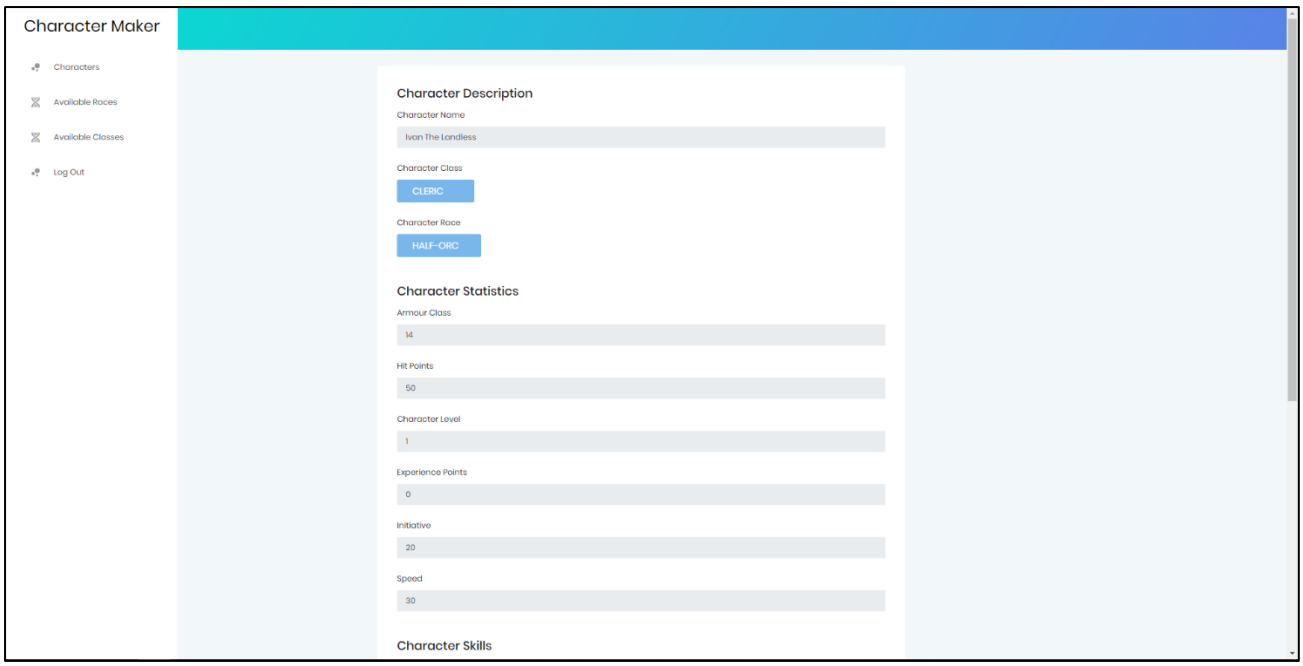
Slika 4.6. Prika forme za stvaranje avatara

Svaki avatar može imati svoju klasu, rasu, ime te različite prednosti i nedostatke koje korsinik popunjava prilikom stvaranja avatara (Slika 4.6.)



Slika 4.7. Prikaz liste avatara nakon stvaranja lika

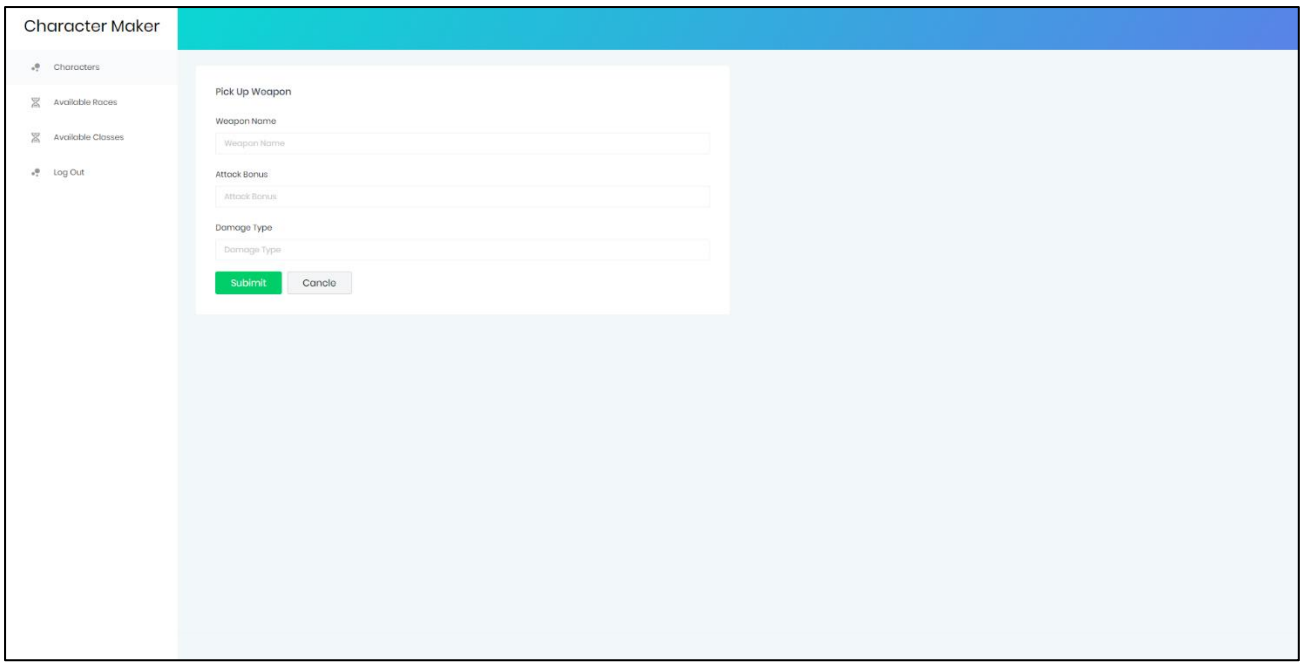
Nakon stvaranja avatara, korisniku se automatski prikazuju svi stvoreni avatari (Slika 4.7.)



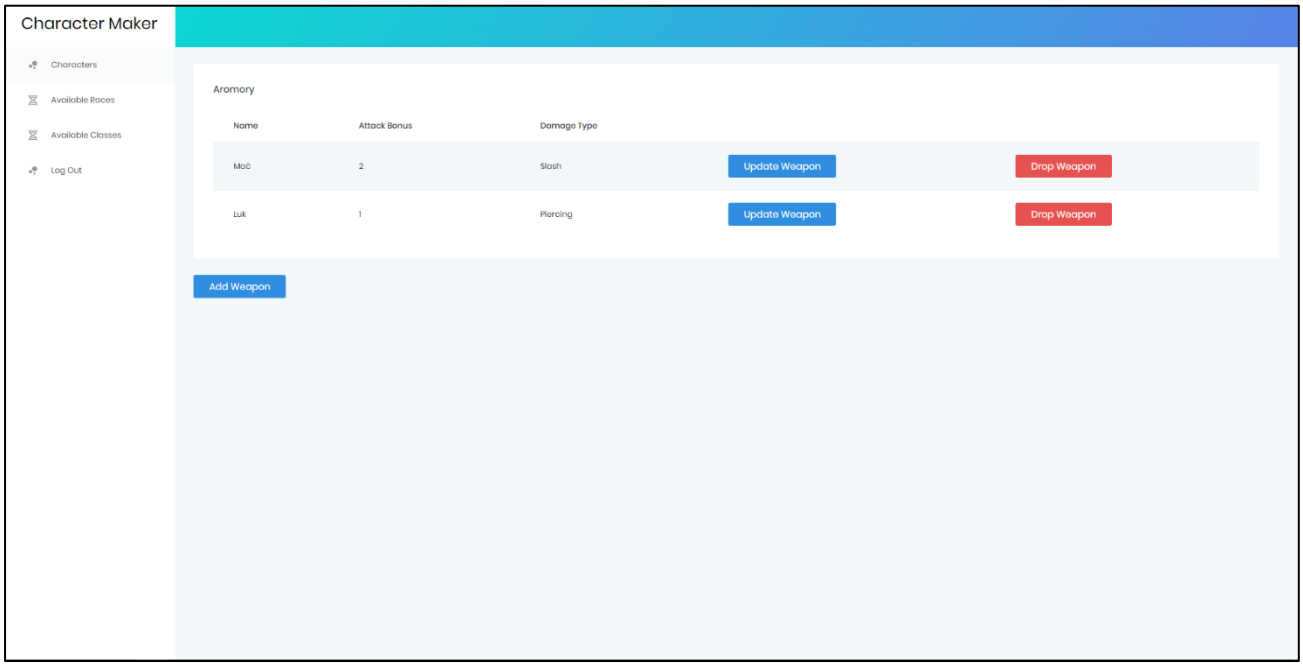
Slika 4.8. Prikaz detalja stvorenih avatara

Detalji svakog avatara prikazani su na posebnoj stranici kako bi korisnik mogao vidjeti sve elemente odabranog avatara (Slika 4.8.)

Svaki stvoreni lik ima svoju oružarnicu, ruksak, knjigu čarolija i vrećicu s novcem. Kako bi bilo moguće stvoriti oružje, novac, čaroliju i razne stvari potrebno je imati lika. Jedan lik može imati više različitih vrsta novca, oružja, čarolija i stvari. Prilikom stvaranja svake stavke potrebno je stisnuti na željeni gumb i popuniti formu koja sprema podatke u bazu podataka. Isto tako je moguće ukloniti stavke iz baze te ih promijeniti ukoliko je korisniku to potrebno.

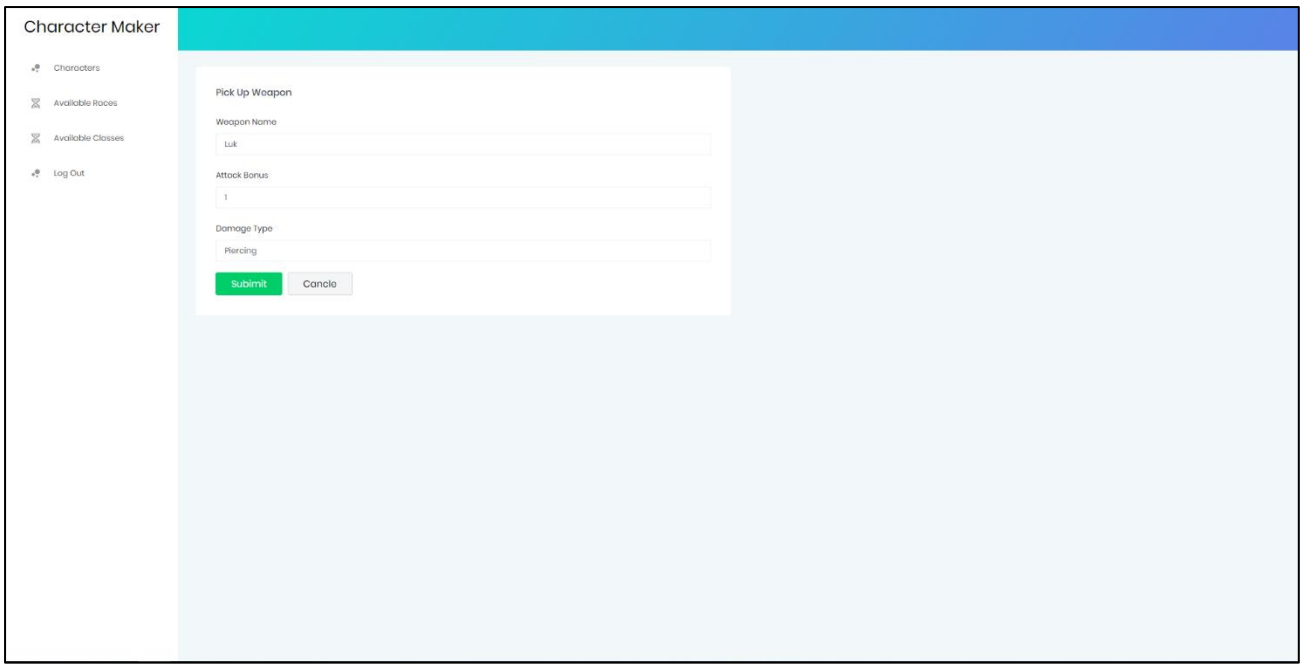


Slika 4.9 Prikaz forme za stvaranje oružja



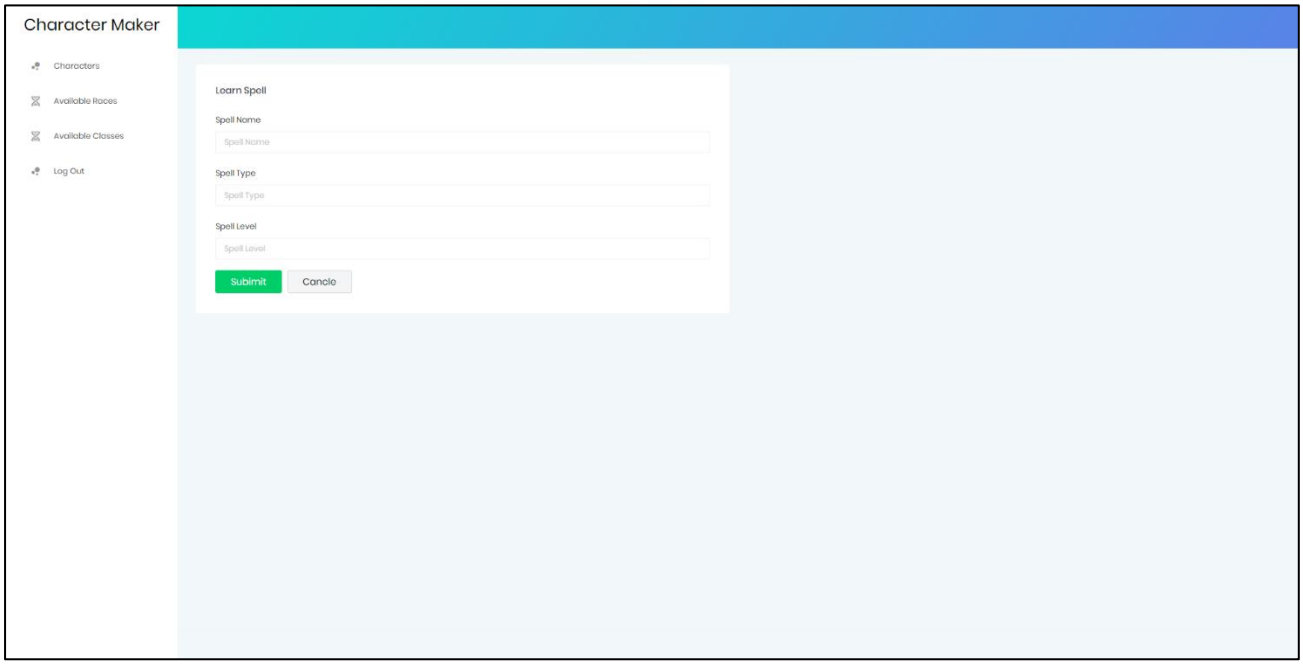
Slika 4.10. Prikaz stvorenih oružja

Popunjavanjem forme za stvaranje oružja (Slika 4.9.) korisnik avataru stvara željeno oružje sa odabranim parametrima. Nakon stvaranja oružja korisniku se prikazuju sva oružja odabranog avatara dostupna na korištenje (Slika 4.10.)



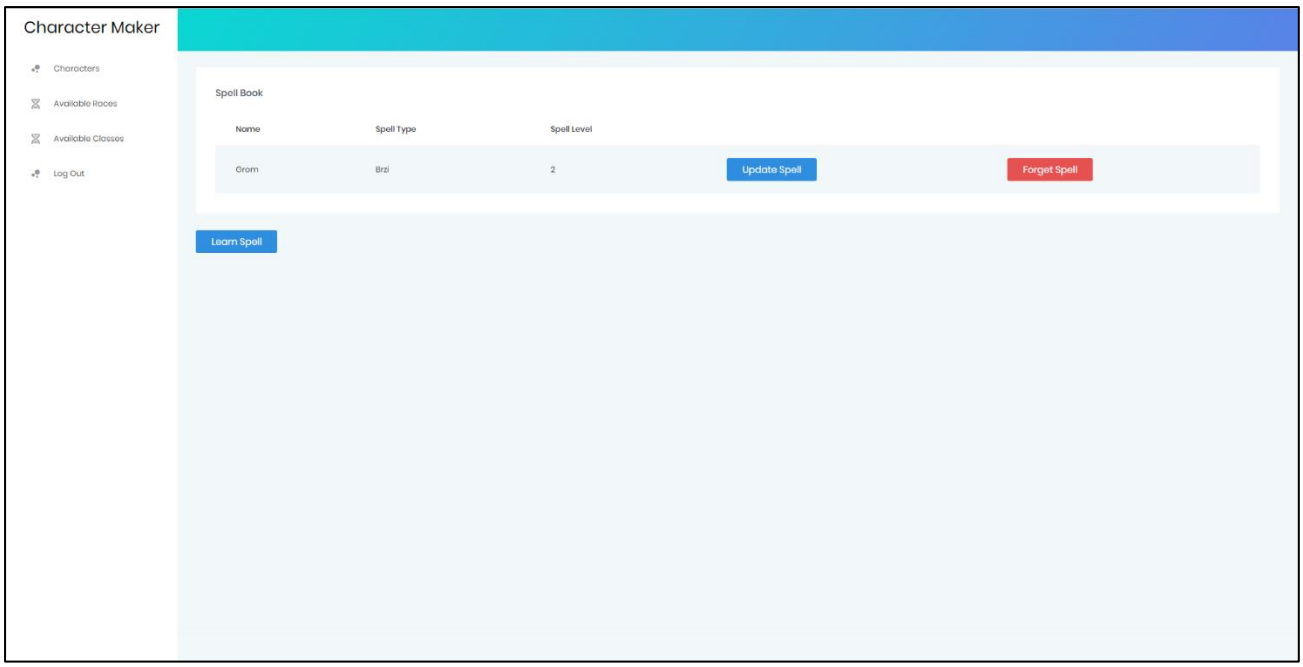
Slika 4.11. Prikaz forme za uređivanej oružja

Za uređivanje oružja ispunjava se forma koja kao početne parametre koje je korisnik unio prilikom stvaranja ili prethodnog uređivanja oružja



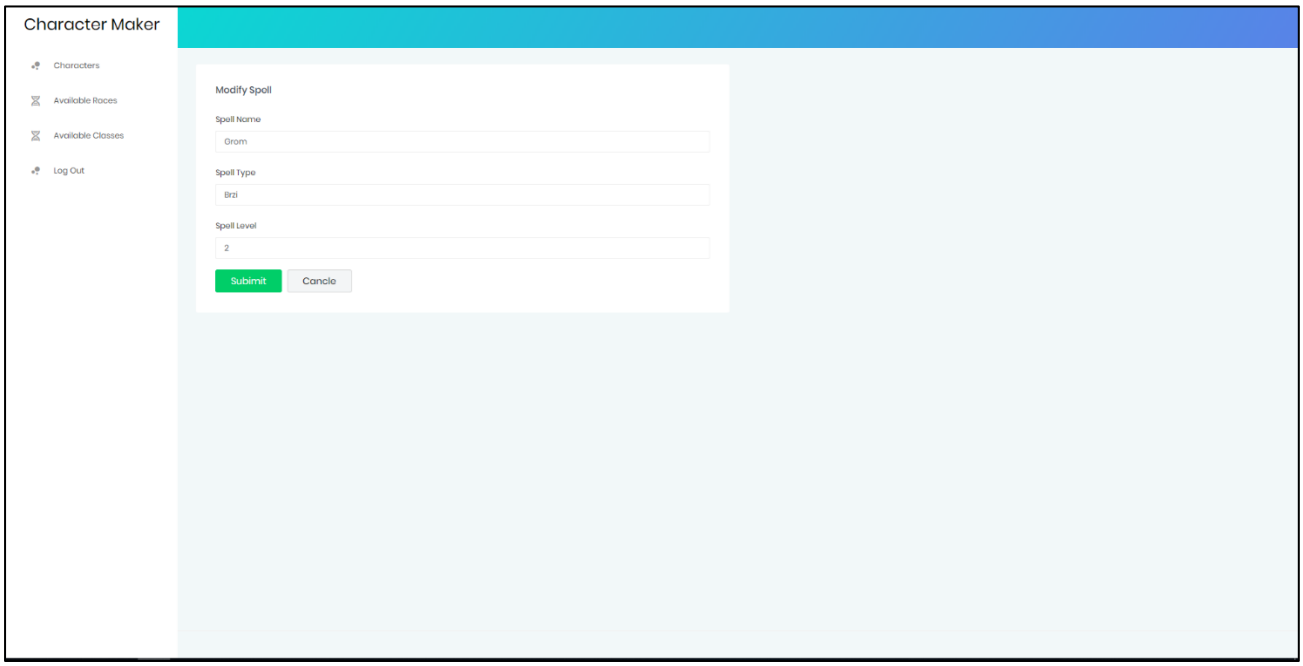
Slika 4.12. Prikaz forme za stvaranje čarolije

Forma za stvaranje čarolije (Slika 4.12.) omogućuje korisniku stvaranje čarolija svome avataru.



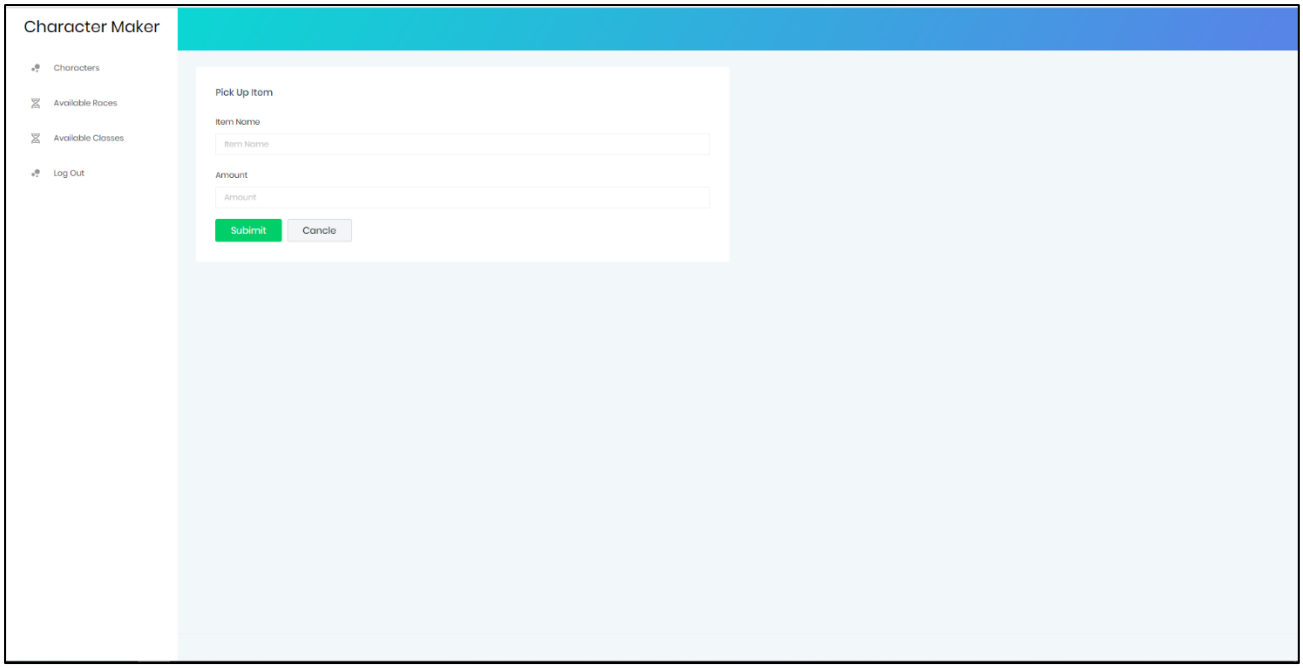
Slika 4.13. Prikaz svih stvorenih čarolija

Nakon stvaranja čarolija korisniku se otvara stranica sa listom svih čarolija koja sada sadrži sve čarolije koje je korisnik stvorio svome avataru (Slika 4.13.)



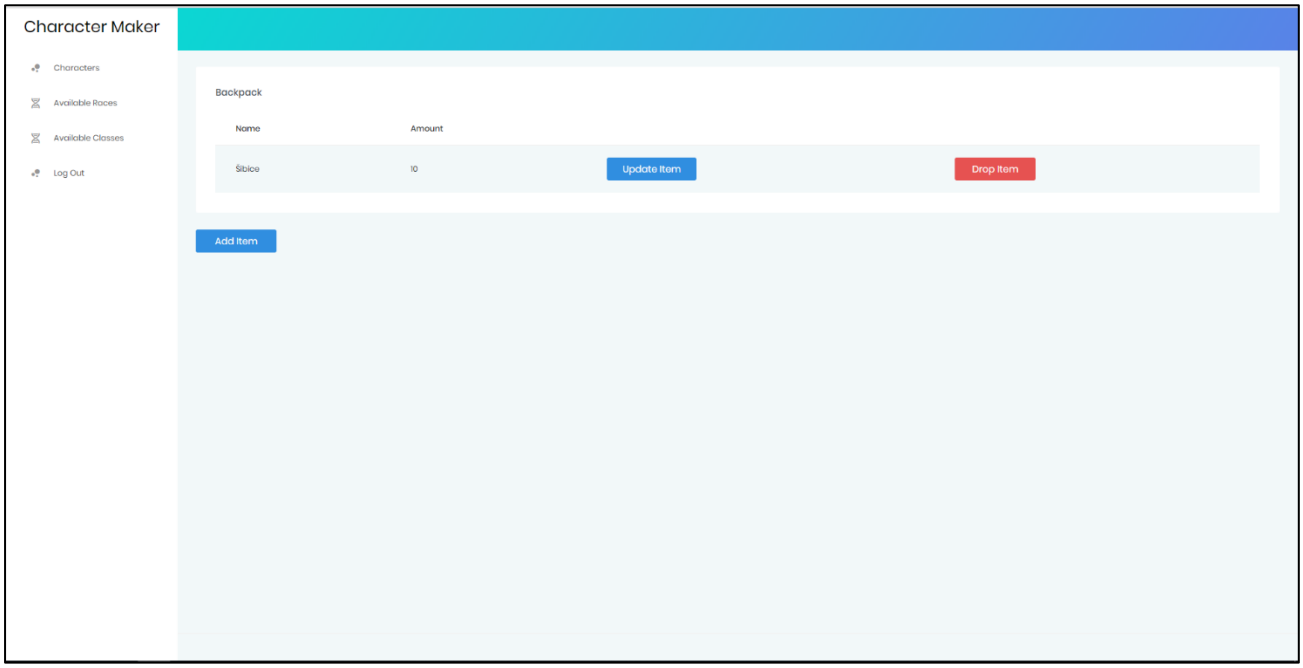
Slika 4.14. Prikaz forme za izmjenu čarolije

Svaka čarolija se može uređivati. Nakon popunjene forme za uređivanje čarolija (Slika 4.14.) izmjene su spremljene u bazu te prikazane na stranici sa listom čarolija.



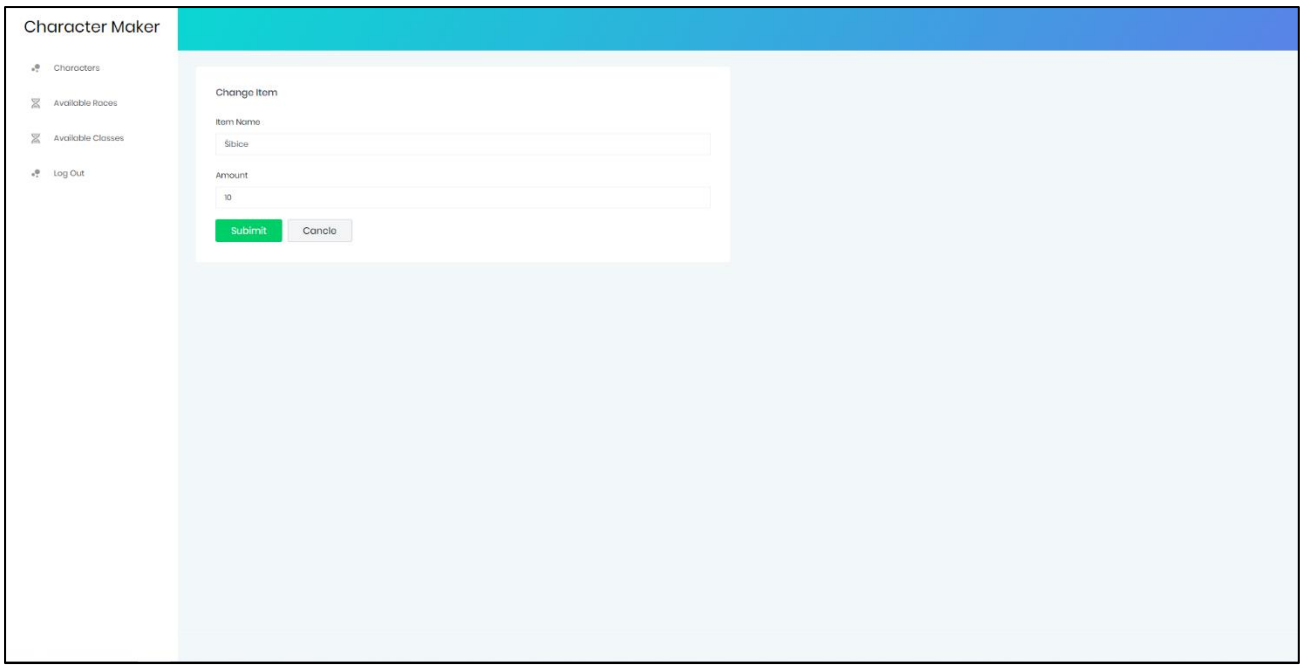
Slika 4.15. Prikaz forme za stvaranje stvari

Svaki avatar ima svoje stvari. Stvar se dodaje u bazu i nudi na raspolaganje korisniku nakon popunjavanja forme za stvaranje (Slika 4.15.)



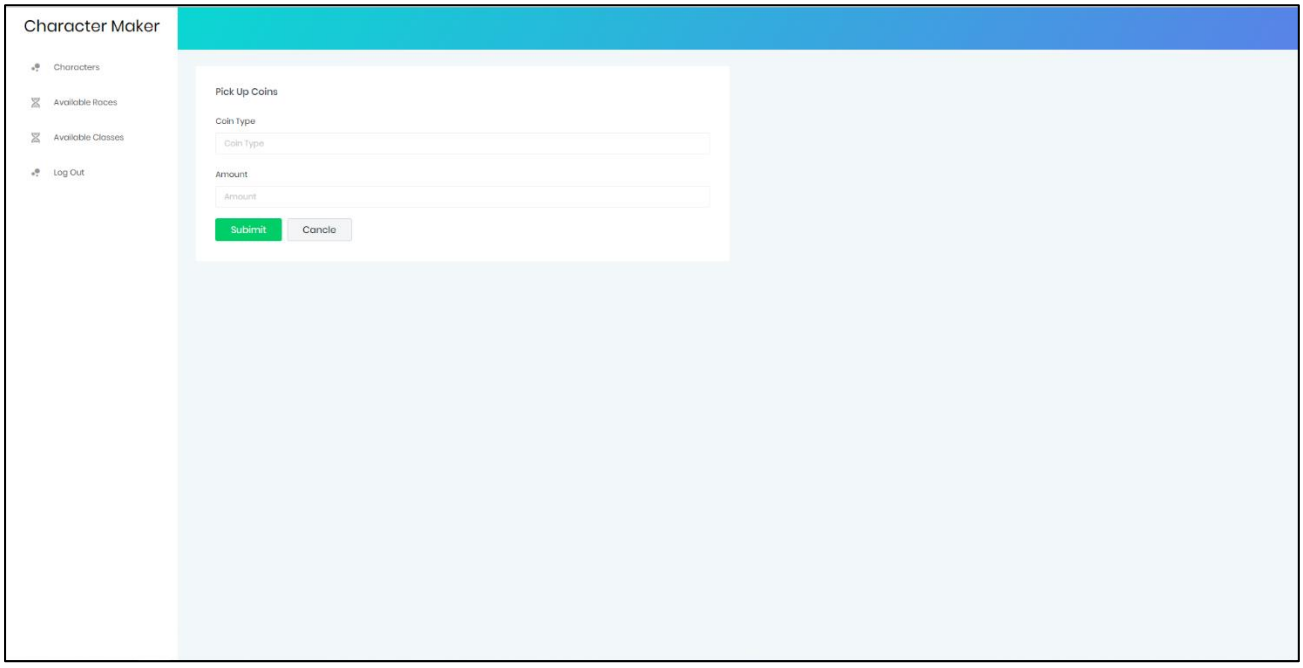
Slika 4.16. Prikaz stvorenih stvari

Prikaz svih stvorenih stvari korisniku nudi pregled stvari koje su dostupne odabranom avataru (Slika 4.16.)



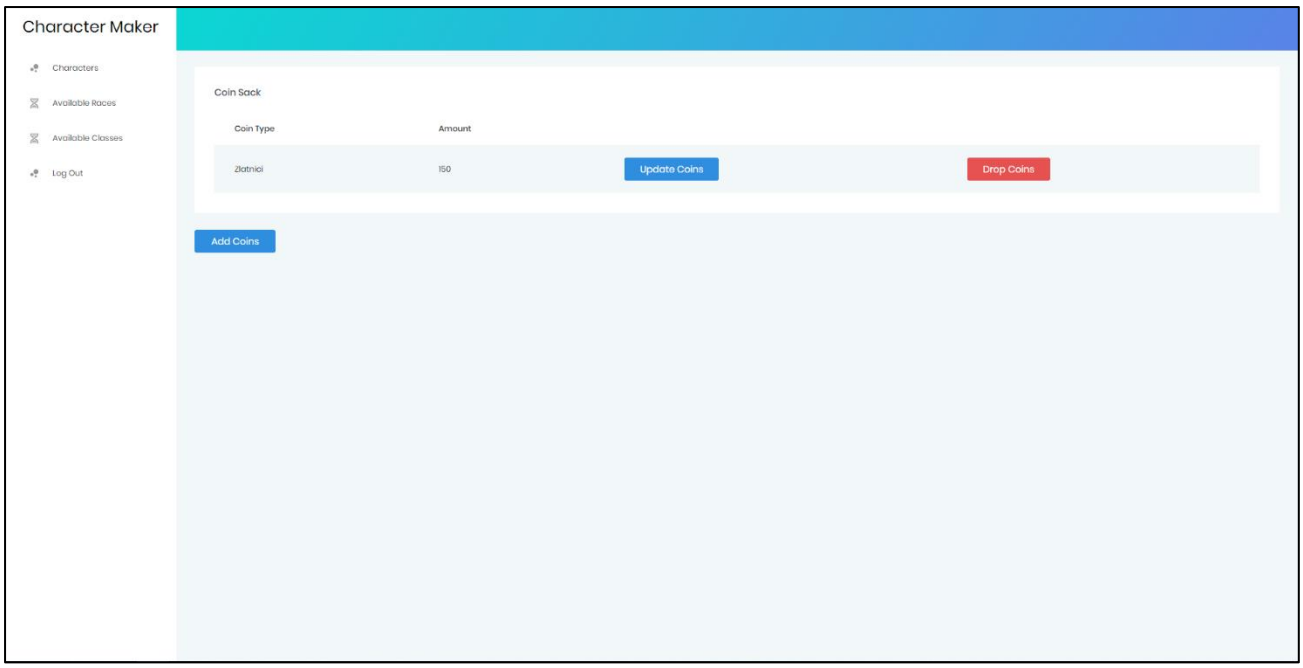
Slika 4.17. Prikaz forme za izmjenu stvari

Svaka stvar može se urediti na način da korisnik u formi za uređivanje stvari prethodno popunjene parametre zamjeni novima (Slika 4.17.)



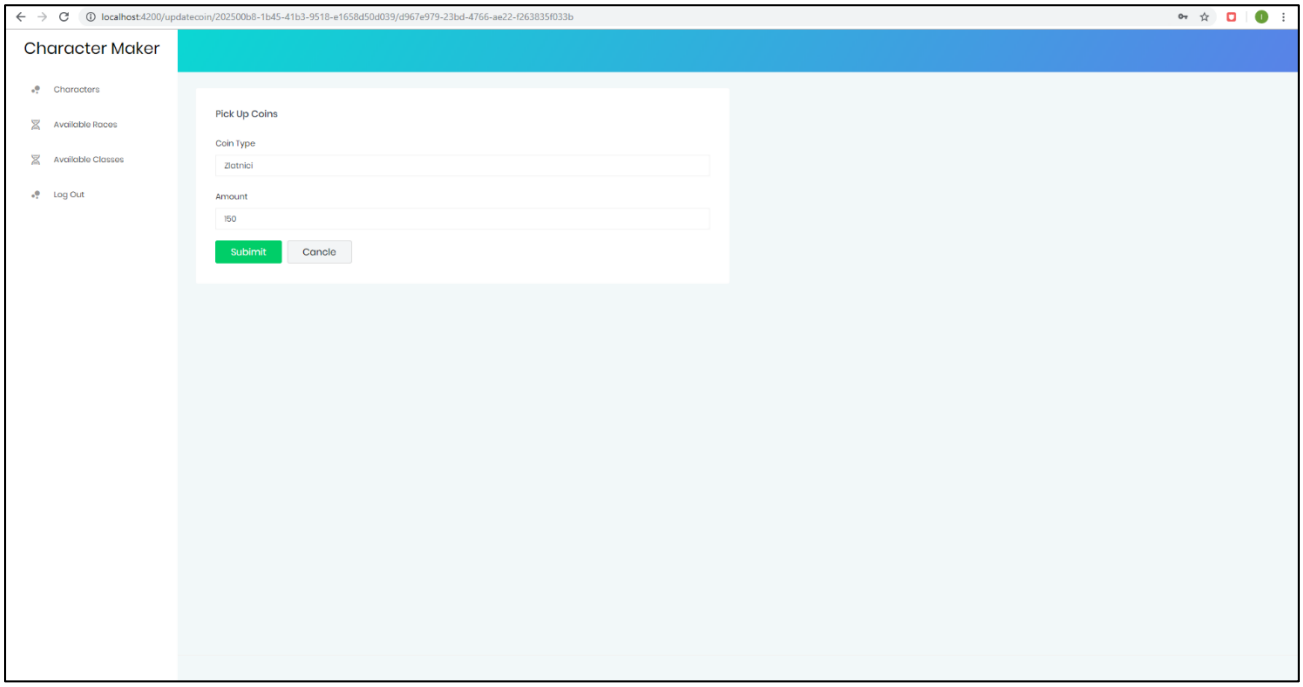
Slika 4.18. Prikaz forme za stvaranje novca

Svaki avatar raspolaže s određenom količinom i određenim tipom novca. Korisnik novac stvara popunjavanjem forme (Slika 4.18.)



Slika 4.19. Prikaz svog stvorenog novca

Korisnik u bilo kojem trenutku može vidjeti koji tip novca i kojom svotom odabrani avatar raspolaže (Slika 4.19.)



Slika 4.20. Prikaz forme za izmjenu novca

U bilo kojem trenutku korisnik može izmijeniti tip i količinu novca svom avataru (Slika 4.20.)

Svaki lik može biti uklonjen u bilo kojem trenutku. Ukoliko se lik obriše sve stvari, čarolije, oružja i novci vezani uz tog lika se brišu iz baze zajedno sa svim podatcima o liku.

5. ZAKLJUČAK

Zahtjevi za web-stranicama su sve češći i sve veći te se tako i razvoj novih tehnologija koje ubrzavaju i olakšavaju izradu istih sve više ubrzava. Kako bi bilo moguće izraditi ovaj završni rad bilo je potrebno proučiti mnogo različitih tehnologija i naći najpovoljnije koje su odgovarale stvaranju korisničkog rješenja. Kao najbolje alate koji bi pomogli pri izradi odabrani su: IntelliJ, jer nudi najbolji pregled stvorenih paketa i datoteka te zbog njegove kvalitetne mogućnosti ispravljanja grešaka, automatskog nadopunjavanja teksta i lakog stvaranja projekta te Visual Studio Code zbog mogućnosti promjene boje sintakse ovisno o korištenom jeziku. Kako bi stvaranje web-stranica bilo moguće, potrebno je uložiti dosta vremena u učenje, kako novih tako i starih tehnologija koje se iz dana u dan mijenjaju. Ova web-stranica za stvaranje izmišljenih avatara za igre igranja uloga poslužila je kao primjer naučenog znanja o Angular i Spring Boot tehnologijama. Pokazalo se kako ovim tehnologijama vrlo brzo dolazimo do željenih rješenja. Dakako, proučavanjem tehnologija ovaj rad ima mogućnosti daljnjeg razvijanja i poboljšavanja.

LITERATURA

[1] Technopedia, ItelliJ, dostupno na:

https://www.techopedia.com/definition/7755/intellij-idea

[travanj 2019.]

[2] Jetrains, Studentm dostupno na:

https://www.jetbrains.com/student/

[travanj 2019.]

[3] Wikipedia, Java, dostupno na:

https://en.wikipedia.org/wiki/Java_(programming_language)

[travanj 2019.]

[4] Spring, Spring Boot, dostupno na:

https://spring.io/projects/spring-boot

[travanj 2019.]

[5] Wikipedia, TypeScript, dostupno na:

https://en.wikipedia.org/wiki/Microsoft_TypeScript

[travanj 2019.]

[6] Wikipedia, Bootsrap, dostupno na:

https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)

[travanj 2019.]

SAŽETAK

U ovom završnom radu prikazano je korištenje Angular i Spring Boot tehnologija u izradi web-stranice. Izrađena web-stranica omogućuje korisnicima stvaranje avatara za igre igranja uloga te da kroz nju brže i lakše prate razvoj svog izmišljenog avatara. Ova stranica poslužila je kao primjer primjene naučenog znanja o odabranim tehnologijama te prikaz procesa izrade web-stranica. Za izradu pozadinske aplikacije korišten je okvir Spring Boot koji koristi Java programski jezik. Pozadinska aplikacija napisana je u integriranom razvojnom okruženju IntelliJ. Korisničko sučelje napravljeno je u Angular okviru koji koristi kombinaciju programskog jezika TypeScript i opisnog jezika HTML. Teorijska podloga, proces izrade web stranice i sam prikaz korisničkog rješenja popraćeni su slikama kako bi prikaz bio što detaljniji.

Ključne riječi: Angular, Spring Boot, web-stranica

ABSTRACT

An application of Angular and Spring Boot technology in website design

In the bachelor's thesis, application of Angular and Spring Boot technologies for making web page is shown. Users are allowed to create a fictional characters for role playing games and they are given easier and faster way of following characters' progress. The page is an example of applied knowledge and how it is created. Spring Boot framework, which uses Java programming language, is used to make back-end application. Its code is written in IntelliJ Integrated Development Environment. Front-end application is made in Angular framework which uses TypeScript programming language and HTML mark-up language. Pictures are used for more detailed description of theory, development process and final result.

Key words: Angular, Spring Boot, web page

ŽIVOTOPIS

Ivan Sertić rođen je 26.svibnja 1997. godine u Požegi. Završio je Tehničku Školu Daruvar u Daruvaru smjer Elektrotehničar. 2016. godine upisao je Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, preddiplomski sveučilišni studij, smjer računarstvo.

Ivan Sertić