WEB APLIKACIJA ZA DRUŠTVENU MREŽU IGRAČA VIDEOIGARA

Rubić, Petra

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Algebra University College / Visoko učilište Algebra**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:225:167245

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-17**

ALGEBRA VISOKO UČILIŠTE

Repository / Repozitorij:

Algebra Univerity College - Repository of Algebra Univerity College

zir.nsk.hr

dabar DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

WEB APLIKACIJA ZA DRUŠTVENU MREŽU IGRAČA VIDEOIGARA

Petra Rubić

Zagreb, veljača 2020.

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

U Zagrebu, 18.2.2020.

Petra Rubić

Predgovor

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

Zahvaljujem se obitelji i prijateljima koji su mi bili motivacija te kolegama s posla koji su mi pomogli pri smišljanju teme za izradu završnog rada i pružili mi dodatne materijale za učenje.

Sažetak

Cilj ovog završnog rada je razraditi ideju razvoja društvene mreže korisnika koji igraju videoigre te način integracije njihovih podataka s popularnih platformi za videoigre i društvenih mreža u aplikaciji. Pripadajućom Java web aplikacijom nastoji se demonstrirati brz i jednostavan razvoj moderne integracijske platforme koja korisnicima omogućava pristup svim podacima vezanih uz postignuća, događanja i novosti u videoigrama.

Ključne riječi: društvena mreža, videoigra, integracija, platforma.

Abstract

Goal of this final thesis is to elaborate the idea of social network development for users who play video games and data integration from popular video game platforms and social networks inside an application. Corresponding Java web application demonstrates fast and simple development of modern integration platform which provides users with the ability to access all data related to achievements, events and news in video games.

Keywords: social network, video game, integration, platform.

Sadržaj

1. Uvod 1

2. Razvoj videoigara i *eSport*-a 2

3. Usporedba rada s postojećim rješenjima na tržištu 3

4. Integracija društvenih mreža 4

5. Integracija platformi za videoigre 7

6. Brzi razvoj Java web aplikacije pomoću Spring programskog okvira 9

7. *Back-end* razvoj 10

   7.1. MySQL model baze podataka 10

   7.2. Implementacija repozitorija 12

   7.3. Mapiranje podatkovnih objekata i modela 13

   7.4. Implementacija REST aplikacijskog programskog sučelja 14

8. Sigurnost web aplikacije 16

   8.1. Autentifikacija 16

   8.2. Autorizacija 17

      8.2.1. JSON Web Token 17

9. *Front-end* razvoj 19

   9.1. Korištenje komponenti za višekratnu upotrebu 20

   9.2. Upravljanje stanjem komponenti 21

      9.2.1. React Hooks 21

      9.2.2. Redux 22

   9.3. Responzivni web dizajn 23

10. Testiranje aplikacije 24

11. Korištenje aplikacije 26

Zaključak 29

Popis kratica 30

Popis slika 31

Popis kôdova 32

Literatura 33

1. Uvod

Ovim završnim radom nastoje se unaprijediti funkcionalnosti koje imaju trenutne platforme za igranje videoigara (engl. *gaming*). Za demonstraciju novih funkcionalnosti napravljena je web aplikacija, uz što je pojašnjena njena arhitektura i tehnologije koje su u njoj korištene.

U radu se prvo opisuje trenutno stanje u razvoju videoigara i *eSport*-a te trenutna rješenja na tržištu za ispunjavanje zahtjeva igrača videoigara (engl. *gamer*). Zatim se objašnjavaju načini integracije društvenih mreža i postojećih *gaming* platformi pomoću REST arhitekture (engl. *Representational State Transfer*, skraćeno REST) unutar aplikacije.

Cjelokupna arhitektura aplikacije opisuje se zajedno s korištenim tehnologijama kako bi se čitatelju predstavili primjeri implementacije modernih tehnologija poput Spring programskog okvira (engl. *framework*) i React-a unutar rada.

Testovi zasebnih logičkih dijelova aplikacije opisani su na kraju samog rada kao i korisničke upute za aplikaciju koje čitatelju objašnjavaju kako su funkcionalnosti implementirane te kako može pristupiti vlastitim podacima putem kreiranja novog korisničkog računa.

2. Razvoj videoigara i e*Sport*-a

Danas videoigre postaju popularnije te se čak i *gaming* počinje smatrati sportom. Komponente računala poput grafičkih kartica, procesora i monitora dio su tehnologije koja se ubrzano razvija svake godine te *gamer*-ima pruža mogućnost igranja većeg broja igara i definiranja personaliziranih postavki unutar sve više igara.

Gaming postaje sport, ili takozvani *eSport*, upravo zbog toga što, kao u nogometu, košarci i drugim grupnim sportovima, timovi zajedničkim trudom dolaze do ispunjenja ciljeva u igri. Svaki član tima može vidjeti vlastite statističke podatke za odabranu igru te na temelju toga procijeniti koliko uspješan može biti u *eSport*-u. Voditeljima *gaming* timova također dobro služe statistički podaci pojedinih članova tima jer pomoću njih mogu (primjerice, uzimajući u obzir podatke o njihovim najviše korištenim i najviše razvijenim vještinama) kvalitetnije isplanirati put prema zajedničkom cilju - pobjedi.

Mogu se izdvojiti dvije različite grupe *gamer*-a: neformalan (engl. *casual*) i *gamer* bez inhibicije (engl. *hardcore*) (Bateman et al., 2005). *Hardcore gamer*-i su poznati po kompetitivnosti i odličnim vještinama u videoigrama te najviše koriste one aplikacije koje im omogućavaju praćenje statistika i rezultata unutar kompetitivnih igara. Međutim, takve aplikacije koriste i *casual gamer*-i te su i oni zadovoljni kada im se pruži mogućnost praćenja napretka unutar same igre za pomoć ili motivaciju tijekom detaljnijeg planiranja aktivnosti ili izvršavanja zadataka.

3. Usporedba rada s postojećim rješenjima na tržištu

Gaming platforme koje trenutno postoje na tržištu baziraju se na načinu prikaza sadržaja za kupnju videoigara, iako neka rješenja imaju implementirano i praćenje progresa unutar svake videoigre. Neke od najpopularnijih platformi su sljedeće: Steam, Epic Games i Origin.

Ovaj rad unaprijedit će nedostatak navedenih platformi, a to je da integracija korisničkih podataka kupljenih igara ne uzima u obzir na kojoj se platformi igra pokreće. Na primjer, korisnik Steam platforme koji igra videoigru ekskluzivnu za Origin ne može vidjeti svoja postignuća dobivena na Steam-u nego samo ona dobivena na Origin-u. S druge strane, platforma kao što je Epic Games korisnicima omogućuje kupnju i pokretanje videoigara, ali im ne pruža pregled postignuća i progresa za sve igre unutar sustava.

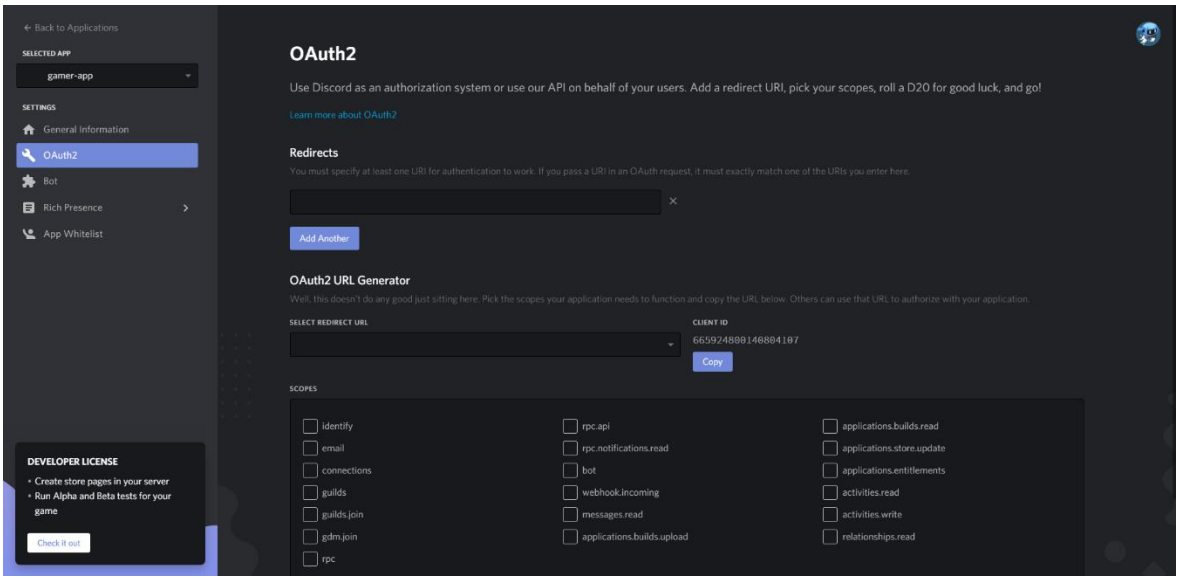
Što se tiče društvene mreže unutar *gaming* platformi, samo Steam ima opciju povezanosti računa s Discord aplikacijom uz implementaciju sobe za razgovor (engl. *chat*). Također, na Steam-u nema mogućnosti davanja direktnih preporuka prijateljima. Postoji jedino mogućnost javnih preporuka na stranicama videoigara, ali koje imaju nedostatak da su neregulirane pa korisnici čije je vrijeme igranja određene igre i manje od jedan sat mogu ostaviti preporuku za istu igru koja negativna, a uz to preporuke mogu biti i nepotpune. Iako je nedavno Steam dodao opciju Steam Labs, koja olakšava pretragu igara po najviše igranim žanrovima ili po preporukama svih igrača specifične igre, dodatna opcija poput izravnog davanja preporuka bi bilo odlično svojstvo (engl. *feature*) koje bi upotpunilo korisničko iskustvo.

Epic Games platforma, najnovija i po izgledu korisničkog sučelja najmodernija aplikacija, korisnicima ne pruža opcije poput pregleda postignuća i pregleda preporuka za igre. Zbog takvog nedostatka, programeri ekskluzivnih igara za tu platformu praćenje postignuća implementiraju unutar same igre.

4. Integracija društvenih mreža

Društvene mreže integrirane unutar aplikacije koje korisnicima pružaju bolji sustav preporuka i komunikaciju s prijateljima su Discord, Twitter i Instagram. Također, ako igra ima svoju stranicu na navedenim društvenim mrežama, korisnik može pregledati novosti o ažuriranjima i blogove napisane o igri unutar same aplikacije. Platforme koje služe za dijeljenje i gledanje videozapisa, YouTube i Twitch, te Deezer za slušanje glazbe unutar aplikacije koriste se za pregled videozapisa, *stream*-ova i glazbenih albuma na temelju svake igre koju korisnik ima u svojoj biblioteci igara (engl. *library*). Za pristup korisničkim podacima na navedenim društvenim mrežama i platformama treba se uspostaviti veza na njihovo aplikacijsko programsko sučelje (engl. *Application Programming Interface*, skraćeno API) putem kreiranja novog pristupnog tokena (engl. *access token*) prema standardnom protokolu za autorizaciju, OAuth 2.0, objašnjenom u poglavlju o autorizaciji.

Discord API korišten je u radu za dohvaćanje svih veza koje korisnik ima uspostavljene s drugim korisnicima i takozvanim udruženja (engl. *guilds*) ili servera u koje je korisnik učlanjen (Discord Developer Documentation, 2019). *Guilds* predstavljaju kolekciju korisnika i Discord kanala koji služe kao informativni forumi za pregled informacija ili razgovore između više korisnika. Za odobrenje korištenja podataka s Discord API-ja u razvoju web aplikacije odlazi se na stranicu za postavljanje pristupnog tokena koja se može vidjeti na slici 4.1. i odabiru se one postavke koje će biti korištene u aplikaciji.



Slika 4.1. Primjer odabira postavki za pristupni token prema Discord API-ju

(https://discordapp.com/developers/applications/665924800140804107/oauth)

Instagram Graph API koristi podsustavni Business Discovery API koji se u ovom radu koristi za dohvaćanje podataka o poslovnom Instagram profilu ako ga određena igra ima i o profilu influencera i javnih osoba ako su povezane uz određene objavljene YouTube ili Twitch videozapise. Ako igra nema poslovni ili javni Instagram profil, po novom Facebook zakonu o pristupu podacima ne mogu se dohvatiti opće informacije s traženog profila (Instagram Graph API, 2019).

Twitter API koristi se slično kao i Instagram Graph API, s tim da on omogućava puno pretraživanje korisnika Twittera po nazivu, što je korisno za pristup stranicama igara i pregled prijatelja povezanih s pojedinačnim korisničkim računom (Twitter Developer Docs, 2019).

Pretraga videozapisa kroz YouTube Data API odvija se korištenjem metode za pretraživanje (engl. *search*) koja vraća videozapis ili listu videozapisa prema upisanom nazivu sadržaja u metodi (YouTube Data API, 2019). Dohvaćanje videozapisa putem navedene metode omogućava prikaz ugrađenih (engl. *embedded*) videozapisa prema nazivu igre.

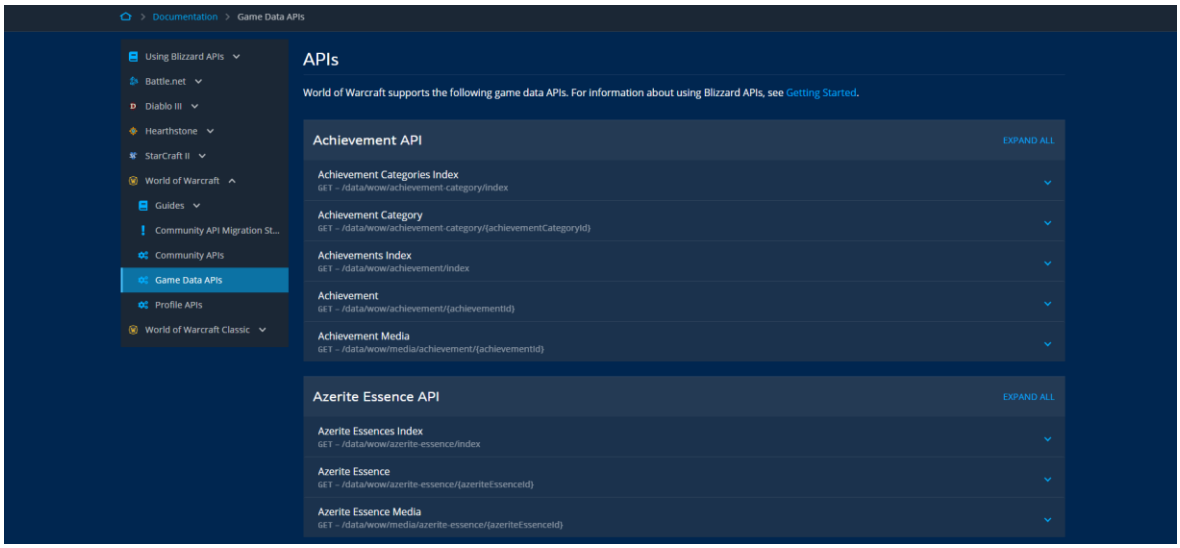
Twitch API u radu se koristi za dohvaćanje *stream*-ova i videozapisa igara po nazivima igara te broj ukupnih gledatelja putem GET metode (Twitch API, 2019). Korisniku se uvijek prikazuju najnoviji sadržaji kako bi mogao pratiti nova događanja unutar *eSports* igara ili popularne videozapise za specifične igre.

Zadnja navedena platforma, Deezer, također ima svoj API putem kojeg se dohvaćaju albumi ili popisi pjesama (engl. *playlists*) kako bi se prikazale informacije o postojećoj glazbi za pojedinačnu igru (Deezer API, 2019).

5. Integracija platformi za videoigre

Popularne aplikacijske platforme koje danas igrači najviše koriste za kupnju i igranje videoigara su Steam, Epic Games, Blizzard Battle.Net i Origin. Po informacijama s foruma Electronic Arts tvrtke, Origin trenutno nema dostupan javni API koji programeri mogu koristiti za pristup korisničkim podacima i podacima o pojedinačnoj igri. Epic Games također navode da pružaju programerima višeplatformske servise (engl. *cross-platform services*) za pristup statistikama igrača i povratnim informacijama igrača nakon igranja. Za sada je samo programerima videoigara odobren pristup analitici igre koju su objavili na Epic Games platformi te e-mail sistem koji se može koristiti za korisničku podršku unutar igre (Epic Games DEV, 2019). Pristup ostalim informacijama poput postignuća unutar igara i korisničke društvene mreže nije dostupan do neodređenog datuma u 2020. godini te se u ovom radu ne može koristiti, ali može se shvatiti kao dodatna značajka (engl. *feature*) aplikacije koja se može implementirati kasnije.

Blizzard Battle.Net platforma nudi pristup statistikama igrača unutar igara kao što su *World of Warcraft, Hearthstone, Diablo III* i *Starcraft II*. U radu se koristi API za pristup podacima pojedinog igrača unutar *World of Warcraft* igre, preko kojeg se mogu dobiti informacije o osvojenim postignućima igrača i naslovima koje je pritom zaprimio. Također, može se pristupiti informacijama o razredima i specijalizacijama lika unutar igre, što je važno prikazati igračima kako bi imali pristup informacijama, ako dođe do ažuriranja unutar igre, kako bi im bilo lakše odlučiti prije pojedinih sesija igre na koji način će igrati, ali je također korisno i novim igračima koji nisu upoznati s Warcraft svijetom (Blizzard APIs, 2019). Blizzard na web stranici za programere ima više API-ja koji su podijeljeni na smislene cjeline prema podacima koji se dohvaćaju poput postignuća ili stvari koje se mogu skupljati unutar igre, a što se može vidjeti na slici 5.1.



Slika 5.1. Prikaz web stranice s listom postojećih API-ja za pristup podacima Blizzard Entertainment igre *World of Warcraft*

(https://develop.battle.net/documentation/world-of-warcraft/guides/namespaces)

Steam API se u radu koristi za više funkcionalnosti, kao što su: prikaz korisničkih lista prijatelja u aplikaciji, prikaz postignuća igrača unutar pojedinih igara te prikaz detalja igre (novosti, ocjene, izvještaji i vodiči za igranje) koju korisnik posjeduje u svojoj Steam biblioteci (engl. *library*) igara. Pristup podacima o globalnim statistikama i postignućima unutar igara omogućava da se u radu korisniku prikazuju najbitniji podaci o igri kako bi svaki korisnik tijekom sesije igre imao bolji uvid gdje se nalazi u odnosu na druge igrače (Steam Web API, 2019).

6. Brzi razvoj Java web aplikacije pomoću Spring programskog okvira

Programski okvir koji nudi infrastrukturnu podršku Java aplikacijama je Spring. Iako Java danas pruža funkcionalni razvoj aplikacija, ono što Java aplikacijama nedostaje je organizacija komponenti ili objekata. Spring programski okvir sastoji se od konteksta aplikacije ili kontejnera (engl. *container*) koji kreira i upravlja komponentama aplikacije koje se zovu *beans*. Takav princip rada programskog okvira poznat je kao inverzija kontrole (engl. *Inversion of Control*, skraćeno IoC). Dizajnerski princip, *dependency injection*, omogućava inverziju kontrole u Spring programskom okviru tako da pojedinačnim komponentama dodjeljuje određenu funkcionalnost (Walls, 2019). Također, detalji funkcionalnosti tih komponenata skriveni su ispod njihovih instanci prema zavisnim objektima. Zbog korištenja komponenata koje nisu zavisne jedna o drugoj (engl. *loosely-coupled*), koncepti IoC i *dependency injection* smatraju se važnima kod brzog i kvalitetnog razvoja modernih aplikacija i mikroservisa.

Spring Boot je također Java programski okvir koji se najviše koristi u brzom razvoju mikroservisa. U ovom radu Spring Boot je korišten jer nudi organizaciju samog projekta, kao što su postavljanje auto konfiguracija za sve korištene vanjske pakete za razvoj aplikacije i kreiranje servisnog sloja koji odvaja poslovnu logiku repozitorija i kontrolera za *front-end* dio aplikacije (Jedrzejewski, 2018). Također koristi *dependency injection* princip putem konstruktora klasa gdje se pomoću anotacije `@Autowired` instanciraju drugi objekti koji se koriste za pristup zasebno definiranoj poslovnoj logici.

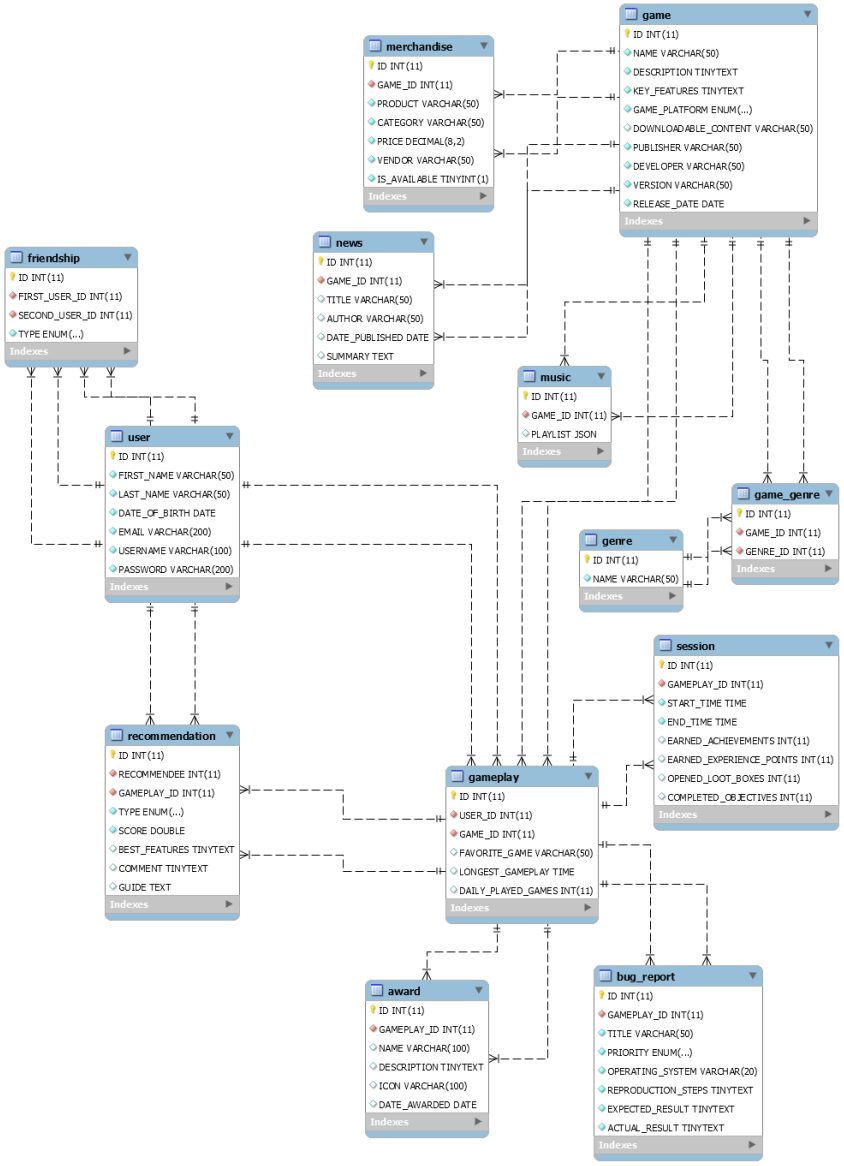
7. *Back-end* razvoj

Razvoj aplikacije za ovaj rad kreće od izrade modela baze podataka. Model baze podataka definiran je u strukturiranom upitnom programskom jeziku (engl. *Structured Query Language*, skraćeno SQL) za definiranje relacijskih baza podataka. Putem implementacije repozitorija pristupa se podacima baze podataka koji se zatim obrađuju u servisnom sloju aplikacije gdje se odvija i poslovna logika te rad s podacima. Sa servisnog sloja šalju se obrađeni podaci na kontroler koji definira API pozive za spremanje, dohvaćanje, ažuriranje ili brisanje podataka, a koji se zatim šalju prema prezentacijskom sloju ili *front-end* dijelu aplikacije za prikaz korisniku.

7.1. MySQL model baze podataka

Za kreiranje skripte baze podataka u radu se koristi MySQL sustav za upravljanje relacijskom bazom podataka (engl. *Relational Database Management System*, skraćeno RDBMS). Skripta je pisana u Liquibase modelu koja omogućava brzu izmjenu i ažuriranje podataka unutar baze podataka te pruža sinkronizaciju podataka sa Spring aplikacijom unutar IntelliJ softvera za razvoj. Liquibase je također koristan za pregled povijesti svih izmjena na bazi podataka pomoću skripti poznatih pod imenom *changesets* (Shmeltzer, 2017) gdje se kroz životni ciklus razvoja baze mogu dodavati i mijenjati tablice u bazi podataka te se uspoređivati verzije promjena nad bazom.

Na slici 7.1. prikazan je dijagram modela baze podataka koji opisuje strukturu aplikacije ovog rada. Najvažnije tablice koje se trebaju gledati su *Game*, *User* i *Gameplay*. *Game* tablica sadrži podatke igre i pomoću jedinstvenog ključa definiranog u toj tablici, druge tablice poput *Merchandise*, *News*, *Music* i *Game_Genre* se nadovezuju te omogućavaju pristup podacima specifično definiranim u tim tablicama.



Slika 7.1. Prikaz dijagrama modela baze podataka

User tablica sadrži podatke korisnika te se na nju nadovezuje tablica *Friendship* koja koristi rekurzivni strani ključ na ID tablice *User* kako bi se mogao definirati status odnosa između dva korisnika. *Recommendation* tablica pomoću stranog ključa *User* i *Gameplay* tablice povezuje podatke preporuka napisanih od strane jednog korisnika i baziranih na iskustvu igranja igre. *Gameplay* tablica označava odnos između korisnika i igre te se na nju

nadovezuje tablica *Session*, koja označava vrijeme igranja pojedine igre te statistike poput riješenih zadataka, dobivenih postignuća i otvorenih nagradnih kutija (engl. *loot boxes*). Tablice *Bug_Report* i *Award* se također nadovezuju na tablicu *Gameplay* te definiraju napisana izvješća o greškama unutar igre i nagrade koje je igrač osvojio rješavanjem zahtjeva unutar svake sesije igranja.

7.2. Implementacija repozitorija

Spring programski okvir nudi rješenje za smanjivanje repetitivnog koda (engl. *boilerplate code*) putem implementacije Spring Data JPA (engl. *Java Persistence API*, skraćeno JPA) repozitorija. JPA programsko sučelje služi za upravljanje podacima iz baza podataka te spajanje jednostavnih Java objekata s navedenim podacima (Gierke, 2008).

JPA repozitorij nudi gotove metode za pretraživanje i spremanje podataka koje ne trebaju biti definirane zasebno. Sljedeći programski kôd 7.1. prikazuje nasljeđivanje JPA repozitorija u sučelju repozitorija za `Game` objekt te definicije dodatnih metoda za dohvaćanje podataka o igrama po unesenom parametru:

```
public interface GameRepository extends JpaRepository<Game,
Integer> {

  List<Game> findAllByPublisher(String publisher);

  List<Game> findAllByDeveloper(String developer);

  List<Game> findAllByReleaseMonth(int month);

  List<Game> findAllByGenre(Genre genre);
```

Kôd 7.1. Definicija sučelja repozitorija koje nasljeđuje JPA repozitorij

Implementacija repozitorija za igre radi se u zasebnoj klasi koja se obavezno označuje s `@Repository` anotacijom, što znači da ta klasa služi za kreiranje, ažuriranje, dohvaćanje ili brisanje podataka iz baze. Sve metode koje nisu u sklopu JPA repozitorija se implementiraju te mapiraju podatke, što je dodatno opisano u sljedećem poglavlju. U sljedećem programskom kôdu 7.2., prikazan je primjer metode unutar implementacije repozitorija koja dohvaća sve Steam igre iz baze te mapira podatke koji se zatim vraćaju u obliku liste podataka spremnih za prikaz korisniku:

```
@Repository
public class GameRepositoryImpl {
  @Autowired
  private GameRepository gameRepository;
  @Autowired
  private ModelMapper modelMapper;
  public List<GameDto> getAllSteamGames() {
    List<GameDto> gameDtoList = new ArrayList<>();
    List<Game> games = gameRepository.findAllSteamGames(
      GamePlatform.STEAM);
    for(Game game : games) {
      GameDto dto = modelMapper.map(game, GameDto.class);
      gameDtoList.add(dto);
    }
    return gameDtoList;
}
```

Kôd 7.2. Implementacija metode za dohvaćanje Steam igara unutar implementacije repozitorija

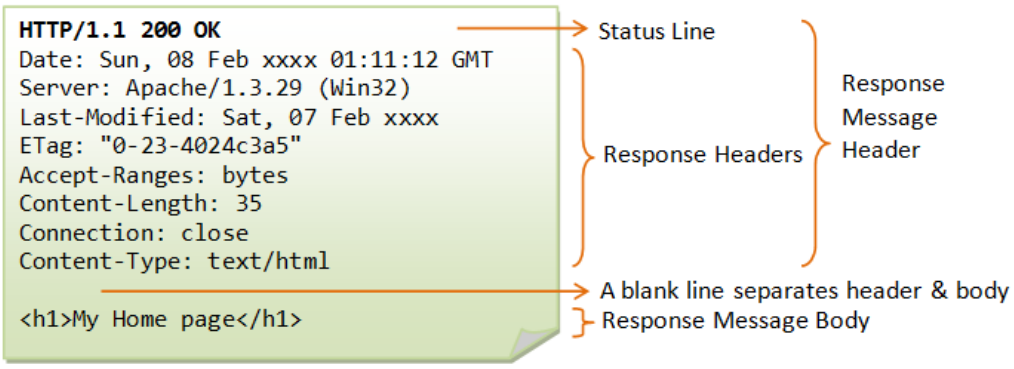
7.3. Mapiranje podatkovnih objekata i modela

Mapiranje modela ili entiteta u podatkovne objekte (engl. *Data Transfer Objects*, skraćeno DTO) važno je zbog skrivanja detalja implementacije jer izlaganje entiteta preko HTTP (engl. *HyperText Transfer Protocol*, skraćeno HTTP) krajnjih točaka na REST API-ju predstavlja sigurnosni rizik izlaganja sigurnosnih podataka korisnika. U programskom kôdu 7.2 unutar metode `getAllSteamGames()` koristi se referenca na `ModelMapper` klasu, `modelMapper.map(game, GameDto.class)`, koja može automatski prepoznati kako se jedan model s definiranim atributima mapira na drugi. Korištenjem `ModelMapper` klase izbjegava se ručno mapiranje objekata, koje bi zahtijevalo postavljanje svakog atributa podatkovnog objekta na vrijednost atributa entiteta, a što bi pak stvorilo repetitivni kôd (Halterman, 2019). Ako `ModelMapper` netočno mapira

podatke mogu se namjestiti dodatne konfiguracije i validacije tijekom povezivanja podataka.

7.4. Implementacija REST aplikacijskog programskog sučelja

REST aplikacijsko programsko sučelje čini skup javno dostupnih krajnjih točaka (engl. *endpoints*) koje služe za razmjene poruka između klijenta i poslužitelja putem web-a, najčešće HTTP protokola. U radu se REST API koristi za slanje i primanje podataka prema korisničkim zahtjevima s prezentacijskog sloja aplikacije. CRUD (engl. *Create, Read, Update and Delete*, skraćeno CRUD) operacije koriste se za upravljanje tim podacima. U Spring programskom okviru REST API kontroler klasa mora biti označena s anotacijom `@RestController` prije naziva klase kako bi svaki rezultat dobiven u API metodama bio automatski serijaliziran u `HttpResponse` format. Na slici 7.2. prikazana je struktura `HttpResponse` formata u kojoj se prikazuje status koji označava uspješnost poslane HTTP poruke, zaglavlja koja sadrže informacije poput datuma zadnje promjene i veličine poruke te tekst poruke u kojem se nalazi zatraženi objekt u obliku JavaScript objektne notacije (engl. *JavaScript Object Notation*, skraćeno JSON).



Slika 7.2. Prikaz `HttpResponse` formata

(https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html)

Također, uz `@RestController` anotaciju može se staviti `@RequestMapping` anotacija s tekst parametrom koji označava glavni URL (engl. *Uniform Resource Locator*, skraćeno URL) u obliku konstantne `String` varijable `BASE_URL`, a koji prikazuje

resurse dobivene od strane API metoda samo za taj specifični kontroler. U sljedećem programskog kôdu, kôdu 7.3., može se vidjeti primjer GET metode koja inače služi za dohvaćanje resursa, a u primjeru se dohvaća lista svih igara:

```
@GetMapping(path = "/list", produces = GameDto.CONTENT_TYPE)
public ResponseEntity<List<GameDto>> getAllGames() {
  LOGGER.info("Getting all games…");
  final List<GameDto> gameDtoList =
      gameService.getAllGames();
  LOGGER.info("getAllGames finished");
  return ResponseEntity.ok(gameDtoList);
```

Kôd 7.3. GET metoda u REST API kontroler klasi

Anotacija `@GetMapping` stavlja se ispred metoda koje vraćaju neke resurse u obliku JSON ili drugog formata. Unutar anotacija označava se put s `path` oznakom koji prikazuje URL nastavak na glavni `BASE_URL` u kojem dolazi do vraćenih resursa samo te GET metode. U klasi podatkovnog objekta `GameDto` definirano je `CONTENT_TYPE = "application/game.v1+json"`, što naznačuje da resurs koji se vraća preko REST API-ja vezan za `GameDto` klasu uvijek dolazi u tom formatu. GET metoda koristi `Logger` klasu pomoću koje se mogu ispisati informacije tijekom izvršavanja programa. Poziv na metodu `getAllGames` iz `GameService` klase omogućuje dohvaćanje liste igara iz baze podataka koje su prikazane korisniku na navedenom *endpoint*-u. Na način kako je implementirana GET metoda implementirane su POST, PUT i DELETE metode koje imaju svoje Spring anotacije i razlikuju se od GET po tome što u parametrima primaju `@RequestBody` anotaciju vezanu uz objekt koji se mijenja ili kreira ili `@PathVariable` anotaciju uz varijablu koja se unosi kao URI (engl. *Uniform Resource Identifier*, skraćeno URI) predložak.

8. Sigurnost web aplikacije

Sigurnost aplikacija danas se smatra važnom kod izrade arhitekture i same implementacije softverskih rješenja jer određuje način na koji korisnici smiju ili ne smiju pristupiti resursima aplikacije. Spring Security je Java programski okvir koji pruža osiguravanje Java i drugih web aplikacija, autentifikaciju, autorizaciju i šifriranje podataka. Spring Security auto konfiguracijom rješava osiguranost svih HTTP krajnjih točaka osnovnom provjerom autentifikacije i kreiranjem korisnika za prijavu s navedenim korisničkim imenom i generiranom zaporkom tijekom svakog pokretanja aplikacije (Alex, 2004) koji se mogu dodatno konfigurirati.

8.1. Autentifikacija

Autentifikacijom se smatra aktivnost kada korisnik ili aplikacija daju valjane vjerodajnice za ovjeru kako bi se dokazao njihov identitet. Autentifikacija se često dokazuje korisničkim imenom i zaporkom.

Spring Security modul već ima implementiranu formu za prijavu u kojoj se unosi korisničko ime i zaporka. Zadano korisničko ime je `user`, dok se zaporka generira svakim pokretanjem aplikacije. Ako se žele promijeniti navedeni autentifikacijski podaci za prijavu u aplikaciju, nove vjerodajnice korisnika mogu se postaviti u datoteci `application.properties` unutar mape resursa za Spring projekt:

```
spring.security.user.name=petra
spring.security.user.password=pass123
```

U radu se koristi autentifikacija pomoću tokena, po standardu JSON web tokena (engl. *JSON Web Token*, skraćeno JWT), koja je više objašnjena u poglavlju 8.2.1. Konfiguracija autentifikacije odvija se unutar `configure(AuthenticationManagerBuilder)` metode u `WebSecurityConfig` klasi koja nasljeđuje `WebSecurityConfigurerAdapter` klasu za postavljanje sigurnosnih postavka unutar aplikacije. U `configure` metodi uzimaju se autentifikacijski podaci definirani u implementaciji `UserDetailsService` klase koja poziva repozitorij za preuzimanje

podataka iz baze podataka te se postavlja metoda kriptiranja zaporke na `BCryptPasswordEncoder` klasu (Krebs, 2018).

8.2. Autorizacija

Autorizacijom se smatra aktivnost u kojoj se određuje razina pristupa ili privilegija korisnika prema mrežnim resursima. Korisniku se može odobriti ili uskratiti pristup određenom resursu na temelju njegove autentifikacije koja je povezana s određenom rolom u sustavu.

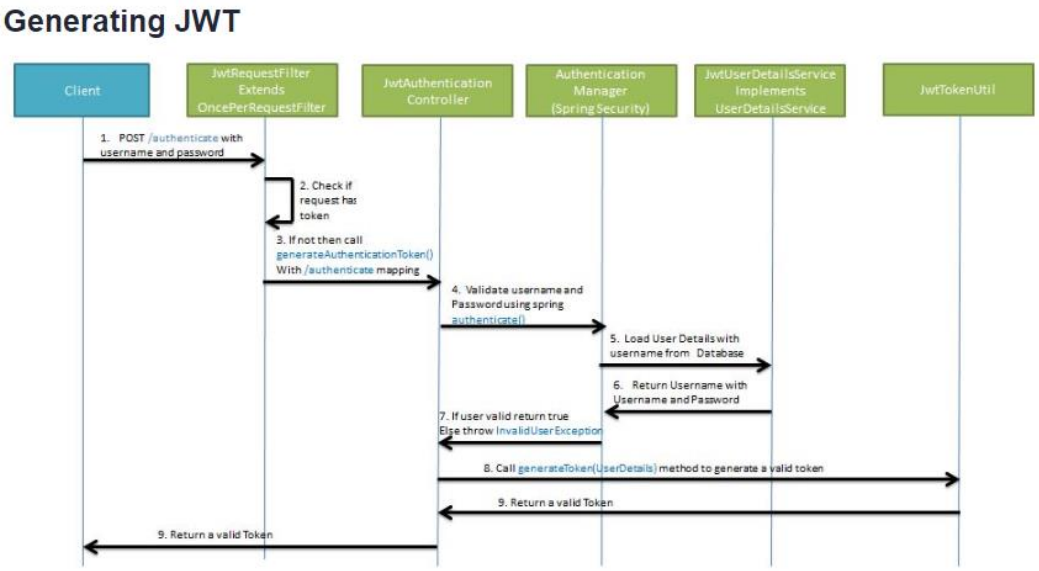
Postavke autorizacije postavljene su isto u `WebSecurityConfig` klasi, ali u `configure(HttpSecurity http)` metodi u kojoj su postavljena ograničenja pristupa na određene krajnje točke aplikacije. Ako se svima omogućuje pristup krajnjoj točki, koristi se sljedeći poziv metode:

```
http.authorizeRequests().antMatchers(API metoda/link krajnje
točke).permitAll();
```

Za one krajnje točke koje pristup krajnjim točkama omogućuju samo određenim rolama korisnika, umjesto `permitAll()` metode koristi se metoda `access("hasRole('NAZIV_ROLE')")` te se postavlja da svaki zahtjev korisnika koji odlazi prema tim krajnjim točkama mora biti od korisnika s postojećim autentifikacijskim podacima unutar aplikacije pomoću poziva `anyRequest().authenticated()`. Za dodavanje autorizacije na pojedinačne REST API metode unutar kontroler klase koriste se Spring anotacije `@PreAuthorize` i `@PostAuthorize` s definiranom rolom korisnika koja ima pristup metodi (Baeldung, 2020). Najčešće se koristi anotacija `@PreAuthorize` koja prije ulaska u metodu provjerava rolu korisnika koji pokušava pristupiti resursima dok `@PostAuthorize` anotacija provjerava rolu nakon što je metoda završena.

8.2.1. JSON Web Token

JSON web token služi kao standard za autentikaciju korisnika u web aplikacijama u obliku JSON objekta. Token se šalje putem URL-a u parametru ili zaglavlju POST metode REST API-ja te sadrži autentifikacijske podatke korisnika. Način na koji se generira JWT vidi se na slici 8.1. na kojoj je prikazano da je na kraju svakog procesa autentifikacije korisnika u aplikaciji uvijek generiran validan token koji se vraća tom korisniku:



Slika 8.1. Prikaz procesa generiranja JSON web tokena

(https://dzone.com/articles/spring-boot-security-json-web-tokenjwt-hello-world)

U radu se unutar klase `SecurityConstants` postavlja konfiguracija JWT-a gdje se definiraju statične varijable:

- `EXPIRATION_TIME`: vrijeme koliko dugo je token validan
- `TOKEN_PREFIX`: prefiks ispred tokena, koristi se Bearer shema
- `SECRET_KEY`: algoritam kojim se potpisuje JWT

Nakon definiranja varijabli za JWT, implementira se autentifikacijski filter za provjeru izdanih JWT tokena korisnicima koji prema aplikaciji šalju svoje vjerodajnice. Metoda `doFilterInternal` unutar klase `JwtAuthFilter` koja nasljeđuje `OncePerRequestFilter` služi za provjeru ispravnosti JWT tokena u trenutku kada korisnik šalje zahtjev prema serveru. Za implementaciju korisničkih vjerodajnica u aplikaciji se koristi `UserDetailsServiceImpl` klasa koja implementira Spring Security sučelje `UserDetails` kako bi se provjerila točnost korisničkih vjerodajnica u bazi podataka kroz `User` klasu. Za obrađivanje iznimki tijekom navedenih procesa kreira se `JwtAuthEntryPoint` klasa koja implementira `AuthenticationEntryPoint` sučelje te pomoću metode `commence` prolazi kroz iznimke i povratne poruke (Krebs, 2018). Nakon ispravne i sigurne provjere JWT tokena, konfiguriraju se rute zahtjeva definiranjem koji su resursi javni, a koji su osigurani, što je objašnjeno u poglavlju 8.2. o autorizaciji.

9. *Front-end* razvoj

Za razvoj prezentacijskog sloja aplikacije odabran je React programski okvir koji nudi fleksibilnosti i prednosti korištenja preko čistog HTML-a (engl. *HyperText Markup Language*, skraćeno HTML) i CSS-a (engl. *Cascading Style Sheets*, skraćeno CSS) za razvoj web stranica. React je JavaScript biblioteka koja se koristi za razvoj interaktivnih korisničkih sučelja (engl. *user interface*, skraćeno UI) te se bazira na razvoju jednostavnih komponenata koje imaju svoju spremljenu logiku poznatu kao stanje (engl. *state*) (Facebook Open Source, 2019). Ono što je odlično kod React-a je to što se pri promjeni stanja unutar komponenti izgradi novi virtualni model za prikaz i interakciju s objektima u HTML-dokumentu (engl. *Document Object Model*, skraćeno DOM) i uspoređuje se sa starim kako bi se našli dijelovi koji se trebaju izmijeniti umjesto da se renderira cijeli DOM, što bi zahtijevalo dodatno vrijeme i ponovno učitavanje web stranice.

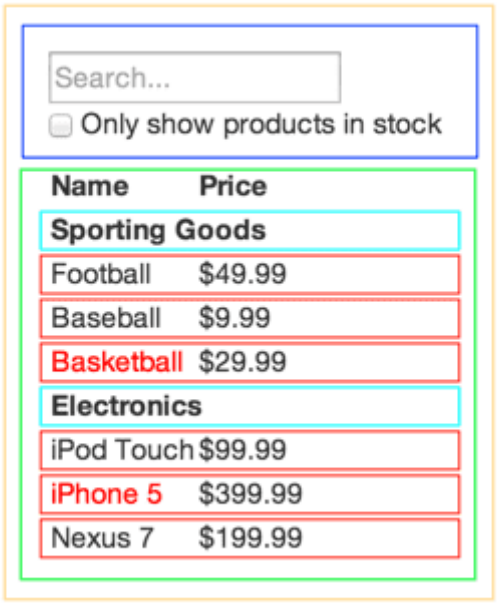
Struktura React aplikacije, dobivena kao početna verzija aplikacije nakon izrade putem naredbe `create-react-app naziv_aplikacije`, sastoji se od `index.js` datoteke koja u DOM-u renderira komponentu `App` s identifikatorom `root`:

```
ReactDOM.render(<App />, document.getElementById('root'));
```

Pod renderiranje se u React-u misli na prikaz svih React DOM elemenata na korisničkom sučelju. Svaka instalirana vanjska biblioteka ili modul sprema se u `package.json` klasu.

9.1. Korištenje komponenti za višekratnu upotrebu

U React-u se komponente rade kreiranjem JavaScript klase koja se može izvoziti (engl. *export*) ili uvoziti (engl. *import*) što pruža najveći dio korisne funkcionalnosti u razvoju web aplikacija. Postoji hijerarhija komponenata u korisničkom sučelju, prikazana na slici 9.1., prema kojoj se roditeljska komponenta može sastojati od više manjih komponenata kroz koje se mogu slati podaci. Takva tehnika slanja podataka između komponenata poznata je pod nazivom „komponente višeg reda“ (engl. *Higher-Order Components*, skraćeno HOC).



Slika 9.1. Prikaz hijerarhije komponenata u primjeru React aplikacije

(https://reactjs.org/docs/thinking-in-react.html)

Svaka komponenta treba biti zadužena za rad na jednom zadatku po principu softverskog dizajna (engl. *Single responsibility principle*). To omogućava višekratnu upotrebu komponenti jer se jedna može koristiti za identičnu funkciju na više mjesta u aplikaciji. Također, komponente ne smiju sadržavati kompleksnu poslovnu logiku jer ih je onda teže ponovno upotrijebiti i postaju manje održive. Na primjer, može se kreirati `Button` funkcionalna komponenta koja ima svoja svojstva (engl. *properties*) poput ikone, teksta, boje i dr. *Props* se smatraju nepromjenjivima te ih komponenta u kojoj su definirani, u ovom slučaju `Button`, ne bi smjela mijenjati. Roditeljska komponenta, na primjer App, može definirati više `Button` komponenti i svakoj putem definiranih atributa poslati

različite *props* što znači da se `Button` komponenta može iskoristiti na više načina jer ima jednostavno definiranu logiku.

9.2. Upravljanje stanjem komponenti

Za razliku od *props*, *state* komponente je promjenjiv na lokalnoj razini komponente što znači da se ne može slati iz roditeljske komponente u dječju komponentu nego se može samo mijenjati unutar komponente u kojoj definiran. Također, postoji mogućnost da neke komponente šalju svoj *state* u obliku *props*-a prema dječjim komponentama. *State* se inicijalizira na početku komponente te se vrijednosti *state* objekta pristupa pomoću `{this.state.state_name}`. Vrijednost se jedino može mijenjati s `this.setState` asinkronom metodom koja ima funkciju koja se izvršava onda kada se *state* ažurira.

9.2.1. React Hooks

React Hooks su novost unutar React programskog okvira te omogućavaju višekratnu uporabu *state*-a komponente bez mijenjanja hijerarhije komponenata. Prije nego što su Hooks uvedene, nije postojao način dijeljenja *state*-a između različitih komponenti, što je dovodilo do restrukturiranja komponenti i kôda unutar njih. Hooks pruža mogućnost podjele jedne komponente u manje funkcije bazirano na tome čemu služe (npr. funkcije za pretplatu na neki događaj ili funkcije za dohvaćanje podataka). Također, smatra se da Hooks služe boljoj optimizaciji softvera nego komponente klase koje potiču nenamjerne obrasce programske strukture.

State Hook se prikazuje u obliku funkcije `useState()` te se može pozvati unutar funkcijske komponente kako bi dodali lokalni *state* koji ima inicijalnu vrijednost definiranu u parametru funkcije:

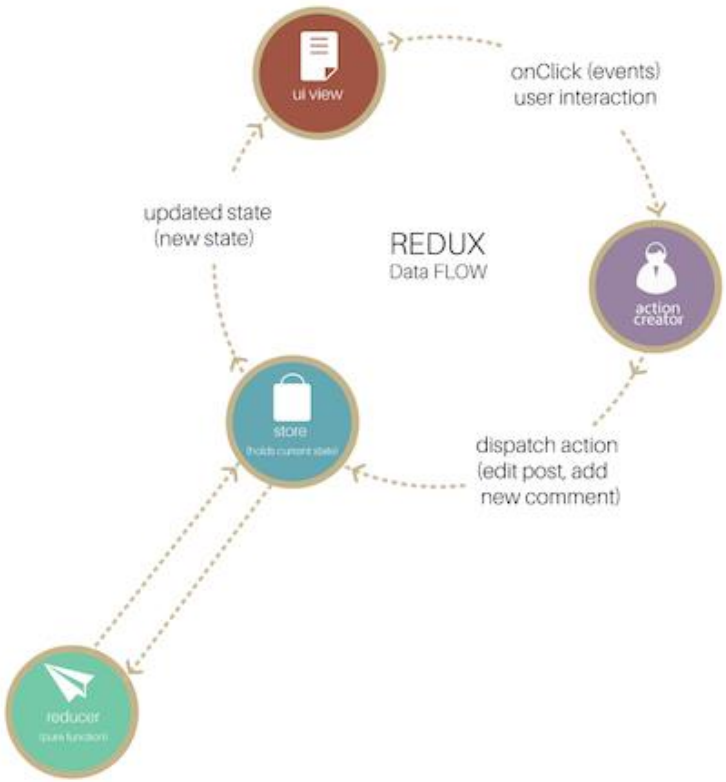
```
const [score, setScore] = useState(0);
```

Funkcija `useState` vraća dvije stvari: varijablu `score` čija je vrijednost inicijalno postavljena na 0 i metodu koja poziva promjenu *state*-a, a to je u ovom primjeru `setScore`. Za ažuriranje vrijednosti `score` varijable, poziva se `setScore` i React između renderiranja zapamti koja je bila inicijalna vrijednost te se ona ažurira kod novog poziva (Facebook Open Source, 2019).

Postoji i Effect Hook koji se prikazuje u obliku funkcije `useEffect()`, ali ne služi za manipuliranje *state*-a unutar komponenata nego za renderiranje efekata nakon ažuriranja React DOM-a.

9.2.2. Redux

Redux održava stanje cijele aplikacije u jednom nepromjenjivom objektu te pri svakoj promjeni stvara novi objekt pomoću *action* i *reducer* objekata. U React-u se koristi za održavanje višekratne upotrebe komponenti. Prednost Redux-a je ta što je neovisna biblioteka koja se može koristiti s bilo kojim JavaScript programskim okvirom kako bi se obrađivalo i mijenjalo stanje u aplikacijama. Tri glavna dijela koja čine strukturu Redux-a su: *actions*, *store* i *reducers* objekti (Bachuk, 2016).



Slika 9.2. Prikaz protoka podataka unutar Redux-a

(https://www.smashingmagazine.com/2016/06/an-introduction-to-redux/)

Actions objekti smatraju se kao događaji unutar aplikacije. Oni šalju podatke dobivene od strane aplikacije, npr. tijekom korisničkog unosa, prema *store* objektu. *Reducers* objekti su funkcije koje dohvaćaju trenutno stanje aplikacije i *action* te vraćaju novo stanje:

```
function handleAuth(state, action) {
 return _.assign({}, state, {
        auth: action.payload
    });
}
```

Store je objekt koji drži *state* cijele aplikacije te nudi metode pomoću kojih se može pristupiti *state*-u, poslati *actions* ili registrirati slušatelje (engl. *listeners*) na određene događaje (engl. *events*). Svaka *action* vraća novi *state* putem *reducer* objekata što daje Redux-u jednostavnost i predvidljivost tijekom rada (Bachuk, 2016). Objašnjen protok podataka između tih objekata može se vidjeti na slici 9.2.

9.3. Responzivni web dizajn

Responzivni web dizajn jednostavno se implementira uz korištenje `flexbox` CSS definicije. Svaka komponenta na web stranici koja služi kao kontejner za druge manje komponente stilizira se pomoću atributa `display: flex` te dodatnim atributima poput `flex-direction`, `flex-wrap` i `flex-flow` za definiranje smjera u kojem se prikazuju manje komponente i broj prikazanih komponenti unutar jednog stupca ili reda definiranog kontejnera. Za određenu rezoluciju ekrana za koju se mijenja prikaz sadržaja naveden je atribut poput `@media screen and (min-width: value)` te su definirani atributi komponenti koji se mijenjaju.

10. Testiranje aplikacije

Svaka aplikacija koja je u tijeku razvoja treba biti testirana kako bi se što prije uočile greške ili promjene u kôdu koje mogu utjecati na trenutni rad aplikacije i na način na koji se buduća ažuriranja razvijaju. Postoje *unit* testovi koji testiraju funkcionalnost manjih dijelova kôda (npr. metoda ili klasa) i integracijski testovi koji testiraju odnos između više komponenata ili rad cijele aplikacije. U radu se koriste *unit* testovi kojima se testira funkcionalnost REST API kontrolera. Napisani su tako da budu brzi i jednostavni te da ne zahtijevaju da se cijela aplikacija pokreće kod izvršavanja testova. Za Java aplikacije postoji jUnit programski okvir za testiranje koji koristi `@Test` anotacije kako bi se označila testna metoda i tvrdnje (engl. *asserts*) kojima se može provjeriti očekivani rezultat u odnosu na stvarni (Vogel, 2019). U programskom kôdu 10.1. može se vidjeti primjer testne metode za testiranje čitanja podataka o igrama unutar `GameController` klase gdje se ispituje postojanost podataka u listi igara prije i nakon promjene jednog podatka u listi :

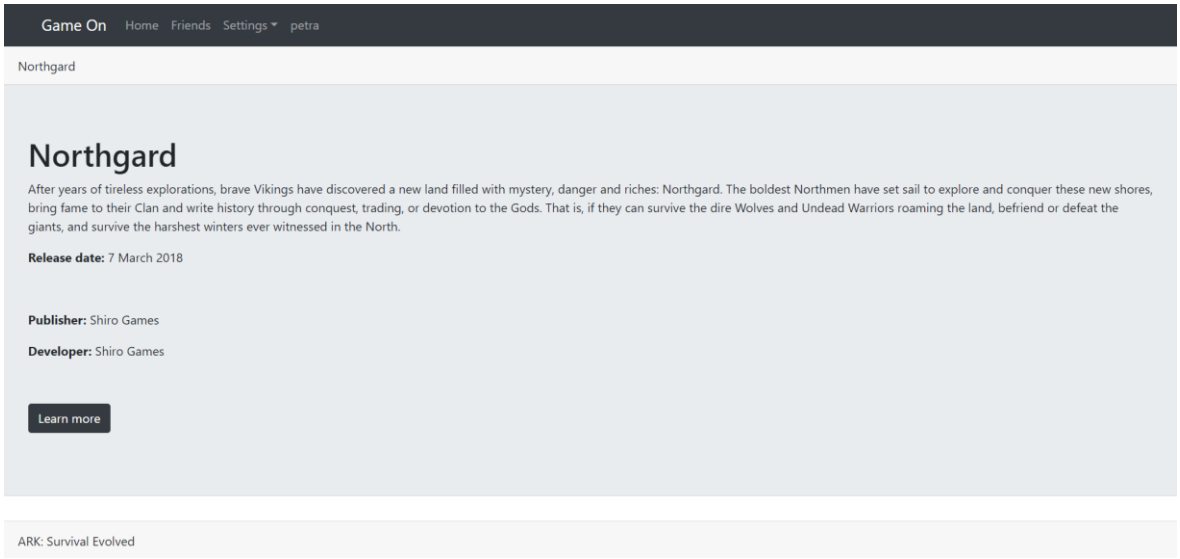
```
@Test
@Transactional
public void readAll() {
  final ResponseEntity<List<GameDto>> response =
      gameController.getAllGames();
  final List<GameDto> games = Objects.requireNonNull(
      Objects.requireNonNull(response.getBody()));
  assertEquals(HttpStatus.OK, response.getStatusCode());
  assertNotEquals(0, games.size());
  final GameDto gameDto = createGameDto();
  gameController.createGame(gameDto);
  final ResponseEntity<List<GameDto>> response2 =
      gameController.getAllGames();
  final List<GameDto> games2 = Objects.requireNonNull(
      response2.getBody());
  assertEquals(games2.size() - 1, games.size());
}
```

Kôd 10.1. Implementacija testne metode za čitanje podataka o igrama

U `readAll()` testnoj metodi poziva se metoda `getAllGames()` unutar `GameController` klase te se provjerava postoji li već trenutno lista igara. Pomoću `assertEquals` poziva postavlja se zahtjev da metoda mora imati nešto zadovoljeno, što je u ovom slučaju da mora vratiti HTTP status 200 za uspješnost API GET poziva, kako bi test prošao dalje. Za razliku od `assertEquals` postoji i `assertNotEquals` metoda koja provjerava točnost očekivanog podataka i vraćenog, što se može vidjeti u programskom kôdu 10.1. u kojem je prikazana provjera da lista igara nije prazna kako bi se testirala uspješnost dohvaćanja podataka. Čitanje podataka o igrama provjerava se dva puta unutar metode kako bi se moglo vidjeti ostaju li podaci ažurni nakon unosa nove igre. U radu su napisane testne metode i za ostale REST API pozive koje se ne razlikuju previše od navedene metode za dohvaćanje podataka jer se time cijela funkcionalnost kontroler klase može testirati, što je ujedno i najvažniji zadatak *unit* testova.

11. Korištenje aplikacije

U aplikaciju se korisnik prijavljuje ispravnim korisničkim imenom i lozinkom te dolazi na početnu stranicu web aplikacije, prikazanu na slici 11.1. gdje se može vidjeti popis igara. Sve igre koje korisnik ima na korisničkom računu prikazane su na tom popisu. Klikom na jednu od igara otvara se dodatni panel koji prikazuje više informacija o igri.

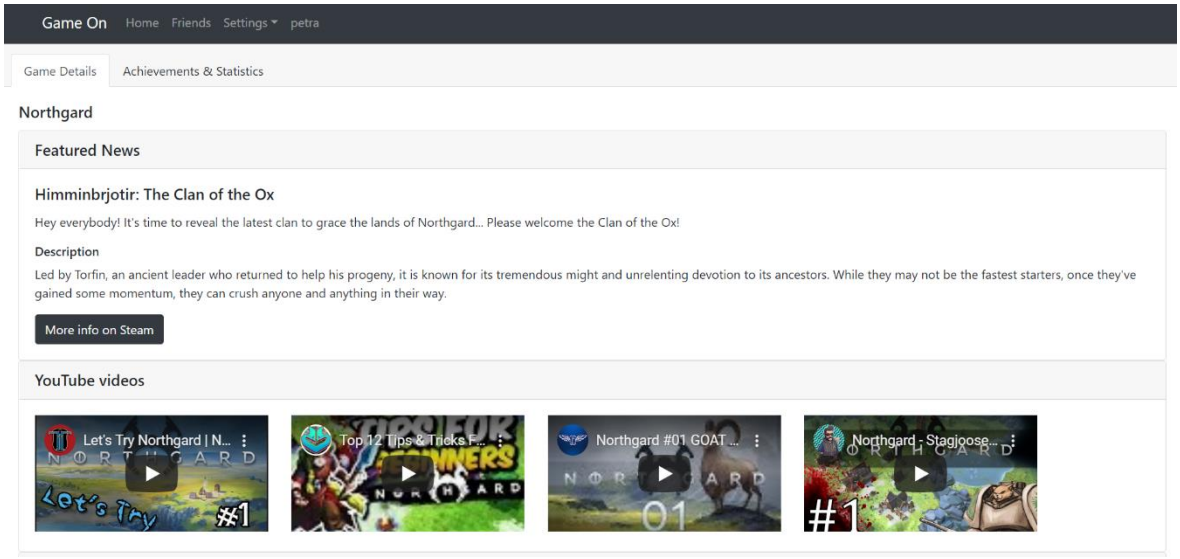


Slika 11.1. Prikaz početne stranice s prikazom detalja igre

Dodatne opcije na početnoj stranici su postavljanje povezanog korisničkog računa koji se povezuje s računom s platformi za igre i računom s neke od društvenih mreža te pregled liste prijatelja. Korisnik može vidjeti listu prijatelja sa Steam i Discord platforme te listu sljedbenika s Twitter društvene mreže što se može vidjeti na slici 11.2.

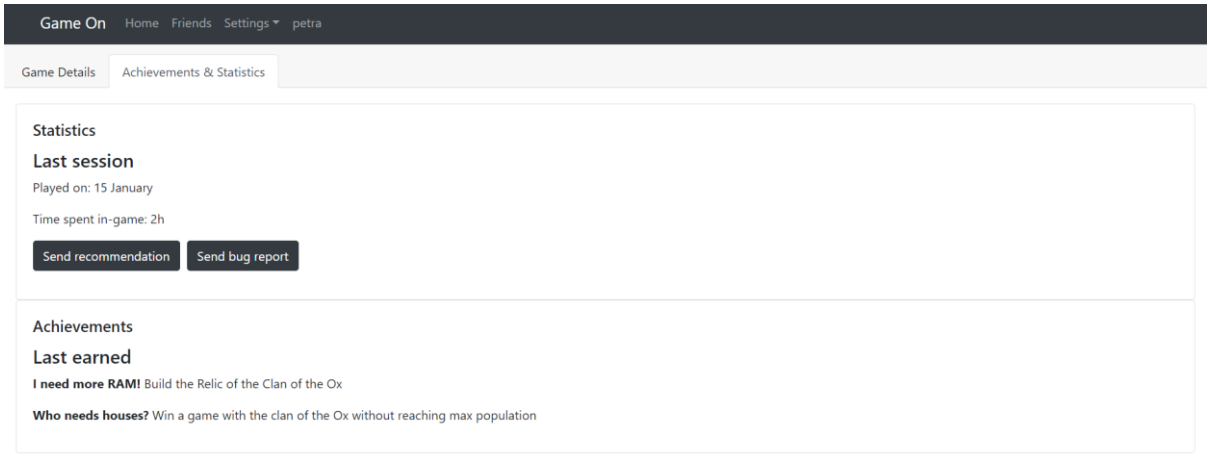
Slika 11.2. Prikaz liste prijatelja u *dropdown* listi za odabrani *tab*

Klikom na gumb *Learn more* u panelu pojedine igre odlazi se na drugu stranicu gdje se mogu vidjeti novosti, glazba, videozapisi i lista dućana u kojem se prodaju proizvodi vezani za igru što se može vidjeti na slici 11.3.



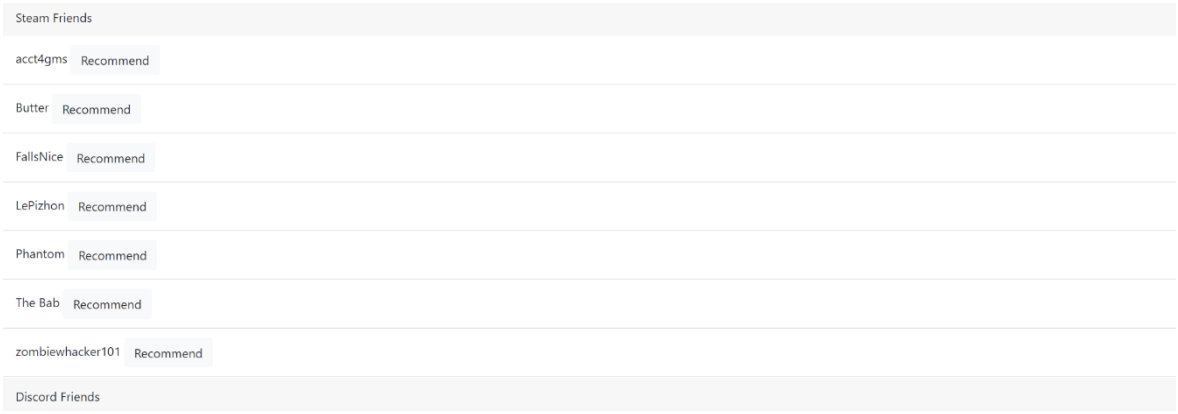
Slika 11.3. Prikaz stranice s aktivnim *tab*-om za prikaz detalja odabrane igre

Dodatne opcije na stranici, prikazane na slici 11.4. su *tab* koji otvara novi panel umjesto glavnog za igru u kojemu se prikazuju sve korisničke statistike za sveukupno vrijeme provedeno unutar igre te trajanje zadnje odigrane sesije.



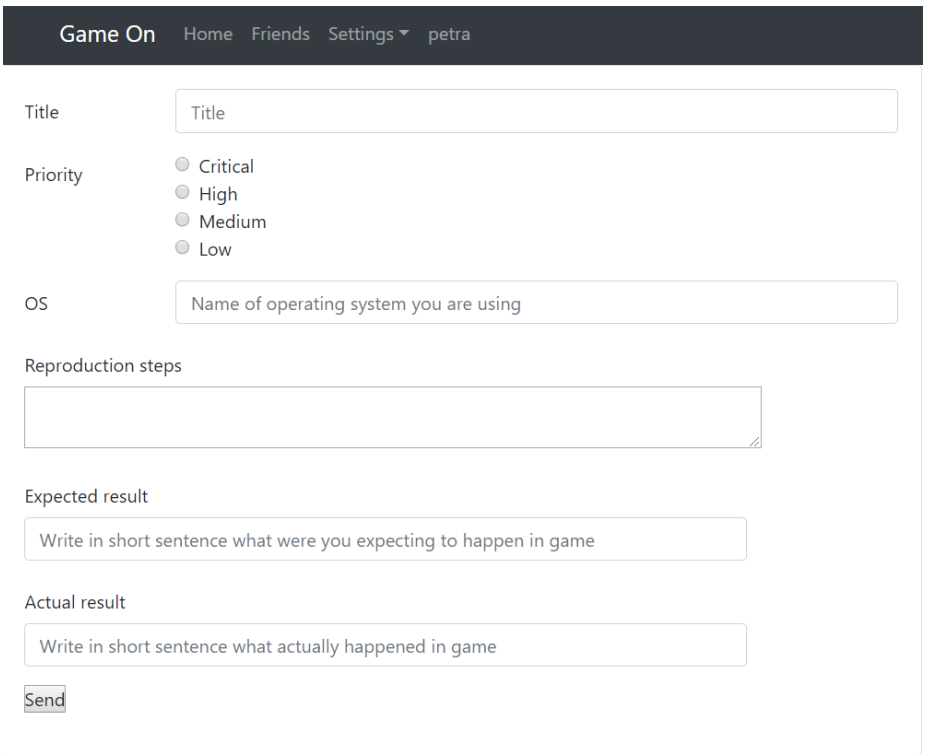
Slika 11.4. Prikaz stranice s aktivnim *tab*-om za prikaz postignuća i statistika

Također, unutar tog panela korisnik može odabrati pisanje preporuke za zadnje odigranu sesiju igre koji zatim dovodi na link *Friends* s listom popisa prijatelja i omogućenim gumbom za slanje preporuke prema njihovoj mail adresi što se može vidjeti na slici 11.5.



Slika 11.5. Prikaz liste prijatelja u *dropdown* listi s omogućenim gumbom za slanje preporuke

Korisnik također može ispuniti formu za prijavu grešaka unutar igre (engl. *bug report*) koja se automatski šalje na e-mail programerima odabrane igre. Struktura navedene forme može se vidjeti na slici 11.6. gdje korisnik mora unijeti sam naslov koji opisuje grešku, koliki je prioritet za rješavanje greške s obzirom na to kako utječe na *gameplay*, operacijski sustav koji se koristi tijekom igranja, koraci reprodukcije greške, očekivani rezultat i stvarni rezultat nakon događaja greške.



Slika 11.6. Prikaz forme za prijavu grešaka unutar igre

Zaključak

Pisanje završnog rada i izrada web aplikacije za društvenu mrežu korisnika koji igraju videoigre pokazuje kako se moderna integracijska platforma, koja nudi korisnicima pristup većim brojem podataka, može razvijati. Spring Boot unutar Spring programskog okvira koji nudi auto konfiguraciju i implementaciju dodatnih klasa ili modula koji su spremni za korištenje ubrzava proces razvoja aplikacije. Prezentacijski sloj aplikacije odvojen je kao poseban projekt te se razvija u React programskom jeziku koji također pokriva jednostavnost i brzinu razvoja pomoću novog dodatka, React Hooks. Sama web aplikacija može se i dalje razvijati kako nastaju nove platforme za igranje igara koje nude pristup korisničkim podacima. Također, kojom brzinom se *gaming* danas razvija, aplikacija može istom brzinom dalje mijenjati svoju arhitekturu ili dizajn kako bi odgovarala zahtjevima korisnika. Uz navedene moderne tehnologije aplikacija se može promijeniti ili nadograditi u kratkom roku s novim funkcionalnostima.

Popis kratica

API	*Application Programming Interface*	aplikacijsko programsko sučelje
CRUD	*Create, Read, Update and Delete*	kreiranje, čitanje, izmjenjivanje i brisanje
CSS	*Cascading Style Sheets*	stilski jezik za web stranice
DOM	*Document Object Model*	model za prikaz i interakciju s objektima u HTML-dokumentu
DTO	*Data Transfer Objects*	podatkovno prijenosni objekti
HOC	*Higher-Order Components*	komponente višeg reda
HTML	*HyperText Markup Language*	prezentacijski jezik za izradu web stranica
HTTP	*HyperText Transfer Protocol*	protokol za prijenos hipertekstualnih dokumenata
IoC	*Inversion of Control*	inverzija kontrole
JPA	*Java Persistence API*	Java aplikacijsko programsko sučelje za dosljednost podataka
JWT	*JSON Web Token*	JSON web token
JSON	*JavaScript Object Notation*	JavaScript objektna notacija
RDBMS	*Relational Database Management System*	sustav za upravljanje relacijskim bazama podataka
REST	*Representational State Transfer*	reprezentacijsko stanje prijenosa
SQL	*Structured Query Language*	strukturirani upitni jezik
UI	*User Interface*	korisničko sučelje
URI	*Uniform Resource Identifier*	usklađeni identifikator sadržaja
URL	*Uniform Resource Locator*	usklađeni lokator sadržaja

Popis slika

Slika 4.1. Primjer odabira postavki za pristupni token prema Discord API-ju 5

Slika 5.1. Prikaz web stranice s listom postojećih API-ja za pristup podacima Blizzard Entertainment igre *World of Warcraft* 8

Slika 7.1. Prikaz dijagrama modela baze podataka 11

Slika 7.2. Prikaz `HttpResponse` formata 14

Slika 8.1. Prikaz procesa generiranja JSON web tokena 18

Slika 9.1. Prikaz hijerarhije komponenata u primjeru React aplikacije 20

Slika 9.2. Prikaz protoka podataka unutar Redux-a 22

Slika 11.1. Prikaz početne stranice s prikazom detalja igre 26

Slika 11.2. Prikaz liste prijatelja u *dropdown* listi za odabrani *tab* 26

Slika 11.3. Prikaz stranice s aktivnim *tab*-om za prikaz detalja odabrane igre 27

Slika 11.4. Prikaz stranice s aktivnim *tab*-om za prikaz postignuća i statistika 27

Slika 11.5. Prikaz liste prijatelja u *dropdown* listi s omogućenim gumbom za slanje preporuke 28

Slika 11.6. Prikaz forme za prijavu grešaka unutar igre 28

Popis kôdova

Kôd 7.1. Definicija sučelja repozitorija koje nasljeđuje JPA repozitorij 12

Kôd 7.2. Implementacija metode za dohvaćanje Steam igara unutar implementacije repozitorija..... 13

Kôd 7.3. GET metoda u REST API kontroler klasi 15

Kôd 10.1. Implementacija testne metode za čitanje podataka o igrama..... 25

Literatura

[1] ALEX B. *Spring Security Reference*, https://docs.spring.io/spring-security/site/docs/5.2.0.RC1/reference/htmlsingle/, 2004.

[2] BAELDUNG. *Introduction to Spring Method Security*. Baeldung, https://www.baeldung.com/spring-security-method-security, 2020.

[3] BACHUK A. *Redux – An Introduction*. Smashing Magazine, https://www.smashingmagazine.com/2016/06/an-introduction-to-redux/, 2016.

[4] BATEMAN C., BOON R. *21st Century Game Design*. Massachusetts: Charles River Media, 2005.

[5] BLIZZARD APIS. *Blizzard Battle.net Developer Portal*. Blizzard Entertainment, https://develop.battle.net, 2019.

[6] DEEZER API, Deezer Developers, https://developers.deezer.com, 2019.

[7] DISCORD DEVELOPER DOCUMENTATION, DiscordApp, https://discordapp.com/developers/docs, 2019.

[8] EPIC GAMES DEV. Epic Online Services, https://dev.epicgames.com/en-US/services, 2019.

[9] FACEBOOK OPEN SOURCE. *React Docs*. Facebook Inc., https://reactjs.org/docs, 2019.

[10] GIERKE O. *Spring Data Commons - Reference Documentation*, https://docs.spring.io/spring-data/commons/docs/current/reference/html/, 2008.

[11] HALTERMAN J. *ModelMapper – Simple, Intelligent, Object Mapping*, http://modelmapper.org/getting-started/, 2019.

[12] JEDRZEJEWSKI B. *Spring Boot - Best practices*, https://www.e4developer.com/2018/08/06/spring-boot-best-practices/, 2018.

[13] KREBS B. *Implementing JWT Authentication on Spring Boot APIs*. Auth0 Blogs, https://auth0.com/blog/implementing-jwt-authentication-on-spring-boot/, 2018.

[14] VOGEL L. *Unit Testing with JUnit*. vogella, https://www.vogella.com/tutorials/JUnit/article.html, 2019.

[15] WALLS C. *Spring in Action (5th Edition)*. Manning Publications, 2019.

[16] SHMELTZER S. *Introduction to Liquibase and Managing Your Database Source Code*. Oracle Blogs, https://blogs.oracle.com/shay/introduction-to-liquibase-and-managing-your-database-source-code?0, 2017.

[17] STEAM WEB API. Valve Developer Community, https://developer.valvesoftware.com, 2019.

[18] TWITCH API, Twitch Developers, https://dev.twitch.tv/docs/api, 2019.

[19] TWITTER DEVELOPER DOCS, Twitter Inc., https://developer.twitter.com/en/docs, 2019.

[20] YOUTUBE DATA API, Google Developers, https://developers.google.com/youtube, 2019.