Platforma za kupovanje kolekcija društvenih igara

Crnogorac, Bernarda

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:743606

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-01-11**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

Platforma za kupovanje kolekcija društvenih igara

Diplomski rad

Bernarda Crnogorac

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 19.09.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Bernarda Crnogorac
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1192R, 08.10.2021.
OIB studenta:	46593626448
Mentor:	izv. prof. dr. sc. Ivica Lukić
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 1:	izv. prof. dr. sc. Ivica Lukić
Član Povjerenstva 2:	izv. prof. dr. sc. Mirko Köhler
Naslov diplomskog rada:	Platforma za kupovanje kolekcija društvenih igara
Znanstvena grana diplomskog rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	U diplomskom radu radu treba izraditi aplikaciju koja svaki mjesec pruža određenu tematsku kolekciju igara koju korisnici mogu kupiti te također pretplatiti se za nadolažeće kolekcije. Nadalje, dopušten je pristup i partnerima (trgovine društvenih igara) koji bi imali uvid u narudžbe i kolekcije te na temelju njih pomogli slagati buduće kolekcije i isporuku trenutnih. Tema je rezervirana za: Bernarda Crnogorac
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	19.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 01.10.2023.

Ime i prezime studenta:	Bernarda Crnogorac
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1192R, 08.10.2021.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Platforma za kupovanje kolekcija društvenih igara**

izrađen pod vodstvom mentora izv. prof. dr. sc. Ivica Lukić

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1

1.1. Zadatak diplomskog rada 2

2. PREGLED PODRUČJA TEME 3

2.1. Pregled sličnih rješenja 3

3. PREGLED KORIŠTENIH TEHNOLOGIJA 6

3.1. Angular 6

3.2. NET 8

3.3. MySQL 11

4. PRIJEDLOG ZAHTJEVA WEB APLIKACIJE 13

4.1. Prijedlog poslovne logike platforme 13

4.2. Prijedlog strukture baze podataka 14

5. PRIJEDLOG PROGRAMSKOG RJEŠENJA WEB APLIKACIJE 16

5.1. Postavljanje razvojnih okolina 16

5.1.1. Postavljanje Angular okruženja 16

5.1.2. Postavljanje *NG-Zorro* biblioteke 17

5.1.3. Postavljanje .NET okoline 18

5.2. Povezivanje klijentske i poslužiteljske aplikacije 19

5.3. *Entity Framework* i kreiranje baze podataka 20

5.4. Autentikacija 21

5.5. Upravljanje proizvodima i kolekcijama društvenih igara 25

5.5.1. *AutoMapper i FluentValidation* 29

5.6. Kreiranje nove teme kolekcije te slanje elektroničke pošte 31

5.6.1. *MailKit* i slanje elektroničke pošte 31

5.7. Kreiranje narudžbe i pojam pretplate 34

5.7.1. Kreiranje narudžbe 34

5.7.2. Pojam pretplate 34

5.8. Filtriranje i pretraživanje kolekcija 35

6. KORIŠTENJE WEB APLIKACIJE I ANALIZA PROGRAMSKOG RJEŠENJA 36

6.1. Korisničko iskustvo krajnjeg korisnika 36

6.2. Korisničko iskustvo tvrtke 41

6.3. Administracijsko sučelje 43

7. ZAKLJUČAK 45

LITERATURA 46

SAŽETAK 47

ABSTRACT 48

PRILOZI 49

1. UVOD

Društvene igre imaju dugu i bogatu povijest koja seže tisućama godina unatrag. Postoje dokazi da su društvene igre bile prisutne još u drevnim civilizacijama. Primjerice, u Egiptu su pronađene ploče s igrom *Senet* koje datiraju oko 3500. godine prije Krista. U antičkoj Grčkoj i Rimu društvene igre su bile popularne među građanima. *Petteia* je bila jedna od najstarijih igara, slična modernom šahu. Rimljani su igrali *Tabula*, preteču modernog *Backgammona*. Tijekom srednjeg vijeka, mnoge društvene igre su se pojavile u Europi. *Čovječe ne ljuti se* je igra koja je postala popularna diljem kontinenta, šireći se do Europe putem trgovaca i putnika. Tijekom 19. i 20. stoljeća, društvene igre su postale sve više komercijalizirane i šire poznate. U ovo doba nastale su mnoge klasične igre poput *Monopoly*, *Scrabble*, *Cluedo* i *Risika*. Razvoj tehnologije, poput televizije i računala, također je utjecao na promjenu društvenih igara.

Danas, društvene igre doživljavaju pravi procvat. Postoje tisuće različitih igara koje pokrivaju različite teme i žanrove, od klasičnih igara, poput šaha, do modernih strateških igara, poput igre *Settlers of Catan*. Društvene igre pružaju zabavu i socijalnu interakciju ljudima diljem svijeta.

Otvaraju se razni dućani, usredotočeni na širenje svijeta društvenih igara. Naravno, procvatom interneta, otvaraju se *online* trgovine i platforme, poput *Kickstarter* platforme, koja je omogućila dizajnerima igara da sami financiraju i izdaju svoje igre. [1] Ova vrsta financiranja omogućuje izravnu vezu između dizajnera i igrača te je potaknula razvoj novih ideja i neovisnih izdavača.

No, u moru silnih društvenih igara, igračima postaje sve teže odabrati pravu igru. Razni načini igranja, razne priče ispričane igrom, kao i razne strategije, čine odabir igre vrlo teškim. Zbog ovog razloga, igrači posežu za *online* stranicama koje preporučuju društvene igre ovisno o kriterijima. U ovome problemu se vidi potreba za stvaranjem servisa koji bi, na temelju korisnikovog kriterija ili želje, odabrao društvene igre za korisnika te mu tako olakšao izbor.

Cilj ovog diplomskog rada je kreirati takav servis, odnosno platformu, koja bi igraču omogućila kupnju društvenih igara s elementima pretplate. Nadalje, kako bi se riješio problem odluke, svaki mjesec bi bile dostupne različite tematske kolekcije društvenih igara, poput kolekcije igara za *Noć vještica*. Korisnik se može pretplatiti na informacije o mjesečnim kolekcijama koje bi mu bile dostavljene kući ili jednostavno može kupiti određene kolekcije bez pretplate. Platforma bi olakšala korisnicima izbor igara, ali i pomogla pri dostupnosti i raširenosti igara igračima.

Rad je predstavljen u sedam poglavlja, u kojem su prva dva kratak uvod o povijest društvenih igara te opis samog diplomskog rada. Nadalje, treće poglavlje donosi pregled korištenih tehnologija u

diplomskom radu, a četvrto opis zahtjeva kojih web aplikacija mora implementirati. S petim poglavljem kreće opis programskog rješenja aplikacije, uključujući i postavljanje svih potrebnih okolina za rad svih okolina opisanih u trećem poglavlju. Korištenje i izgled gotove web aplikacije su dani u šestom poglavlju putem triju korisnika kojima je namijenjena aplikacija, te se cijeli rad sažima zaključkom u sedmom poglavlju.

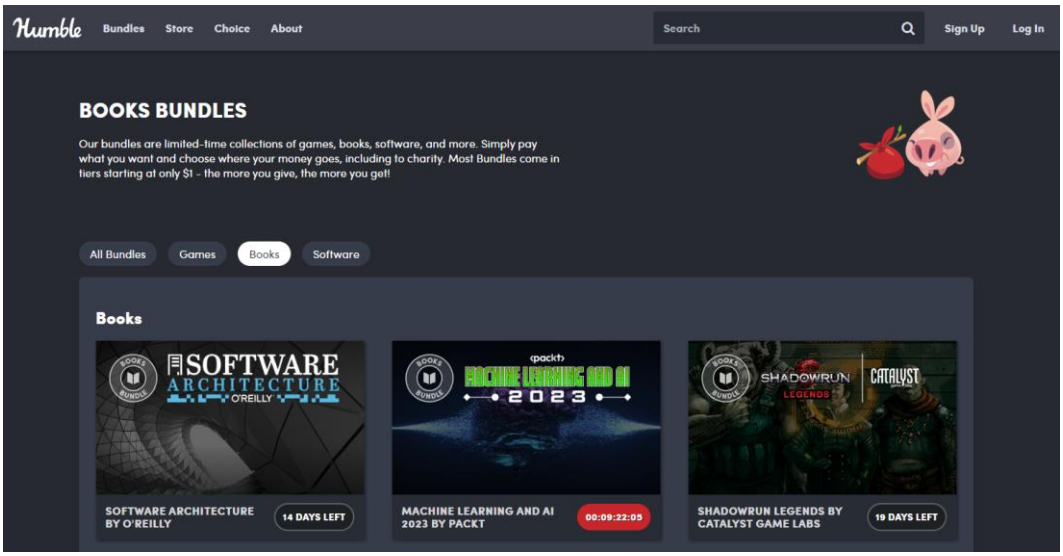
1.1. Zadatak diplomskog rada

U diplomskom radu treba izraditi aplikaciju koja svaki mjesec pruža određenu tematsku kolekciju igara koju korisnici mogu kupiti te također pretplatiti se za nadolazeće kolekcije. Nadalje, dopušten je pristup i partnerima (trgovine društvenih igara) koji bi imali uvid u narudžbe i kolekcije te na temelju njih pomogli slagati buduće kolekcije i isporuku trenutnih.

2. PREGLED PODRUČJA TEME

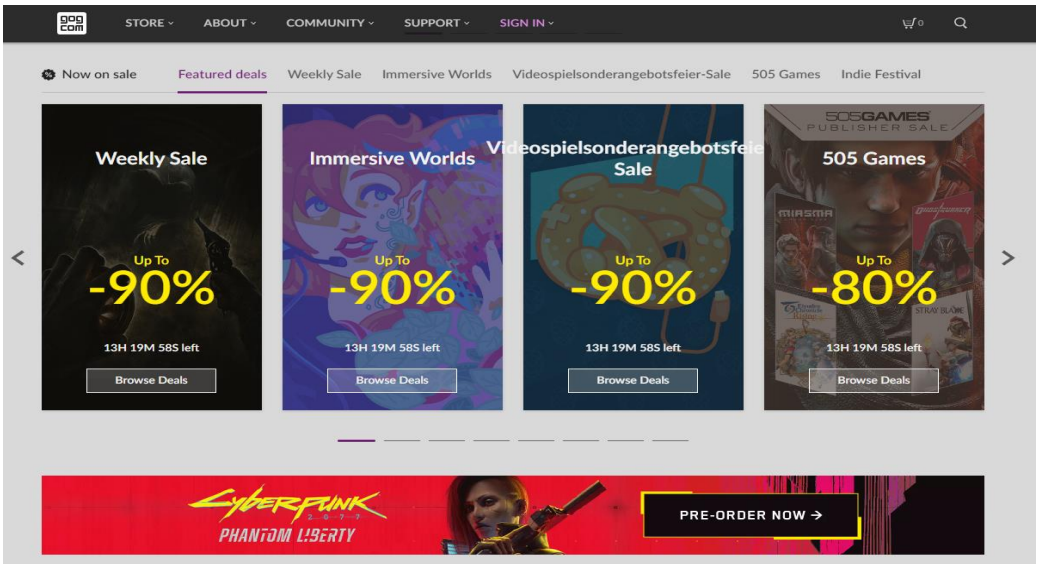
2.1. Pregled sličnih rješenja

Prvi primjer sličnog rješenja je *Humble Bundle*, internetska platforma koja nudi pakete digitalnih proizvoda, uključujući video igre, e-knjige, softvere i druge digitalne sadržaje. [2] Posebnost *Humble Bundle*-a je što korisnici sami određuju cijenu koju su spremni platiti za paket proizvoda, pri čemu je dostupan fleksibilan model plaćanja. Postoje razni paketi (engl. *bundle*) koji se grupiraju tematski i prema izdavačima igara. Primjerice, *Humble Book Bundle* s različitim e-knjigama (slika 2.1.). Mnogi korisnici vole ovaj model jer im omogućuje pristup sadržaju po pristupačnoj cijeni, a istovremeno podržavaju dobrotvorne inicijative. Problem, ili bolje nedostatak, ove platforme je što vrlo rijetko budu paketi s društvenim igrama, pošto je riječ o digitalnom sadržaju.



Slika 2.1. *Humble Bundle stranica*

Iduće rješenje koje se ističe u svijetu paketa digitalnog sadržaja je *Good Old Games*, ili skraćeno *GOG* (slika 2.2.). To je digitalna platforma za distribuciju video igara i softvera, koja često drži ponudu paketa sniženih igara. Također, postoji i alternativa *Steam* s istim principom povremene ponude paketa, no posjeduje isti problem – nisu društvene igre. Sva navedena rješenja su odlična za digitalni sadržaj, no ne za društvene igre.

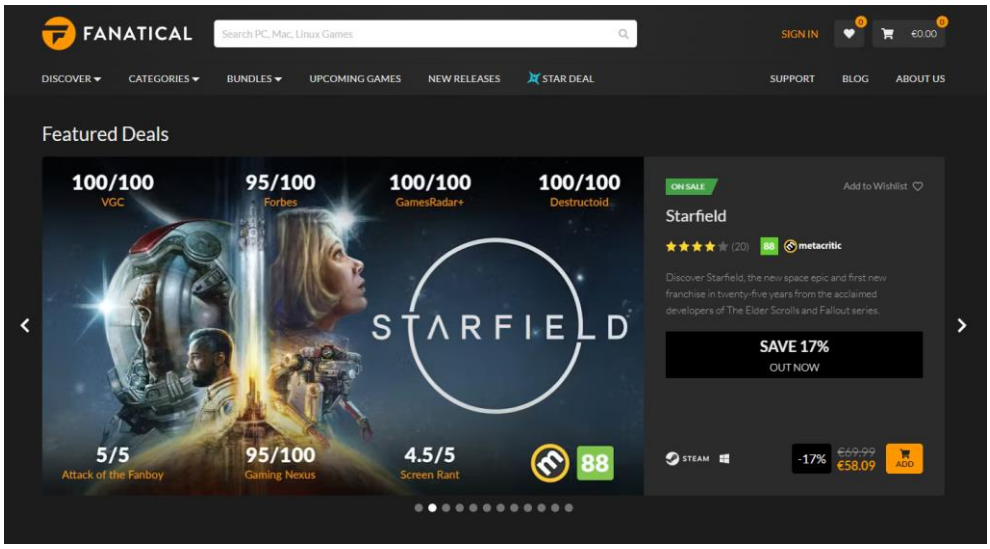


Slika 2.2. *Good Old Games stranica*

Carta Magica je vrijedna spomena jer jedina odgovara temi društvenih igara. [3] *Carta Magica* je lanac trgovina specijaliziranih za prodaju i promociju društvenih igara, kartičnih igara, kolekcionarskih karata, figurica i pripadajućeg pribora. Prvi *Carta Magica* dućan otvoren je 1993. godine u Montrealu, Kanada, i od tada se proširio na nekoliko lokacija diljem svijeta. Postala je prepoznatljiva marka među ljubiteljima društvenih igara i kolekcionara karata zbog svoje stručnosti, širokog asortimana i angažmana u promicanju igara. Također, postoji i *online* web shop trgovine gdje se može naručiti asortiman. No, iako je vrlo popularna trgovina, ne nudi mogućnost kupnje paketa društvenih igara ili predlaganja igara na temelju određenih tema ili koncepata.

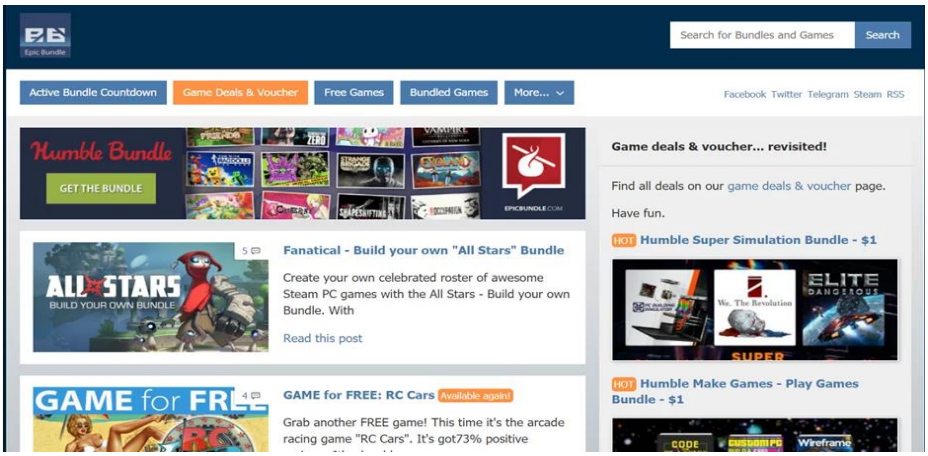
Slika 2.3. *Carta Magica stranica*

Nadalje, kao slično rješenje se spominje i *Fanatical*, što je online prodavač digitalnog sadržaja, ponajviše videoigara (slika 2.3.). Osnovano u Ujedinjenom Kraljevstvu 2012. godine, *Fanatical* je mjesto gdje se mogu pronaći naslovi kategorizirani prema najprodavanijim, najnovijim ponudama i sl. Osim ponude videoigara, također su ponuđene kolekcije softvera i e-knjiga za tečajeve kodiranja i programiranja i sl. Iako imaju zanimljivu i široku ponudu, ne sadrže kolekcije društvenih igara ili sličnih proizvoda.



Slika 2.6. *Fatantical stranica*

Naposlijetku, *Epic Bundle* stranica je dobra alternativa u aspektu ponude videoigara, kao i softverskih kolekcije te kolekcija e-knjiga. Stranica nudi široki spektar igara i softvera te se uvijek može pronaći nešto korisno i vrijedno kupnje, no problem je kako bi se dobile najbolje ponude, potrebno je svakodnevno provjeravati i osvježavati stranicu. Nedostaje element pretplate koji bi obavijestio korisnika o novoj kolekciji te tako olakšao korisničko iskustvo. Također, stranica nema u ponudi kolekcije društvenih igara, što je još jedan minus.



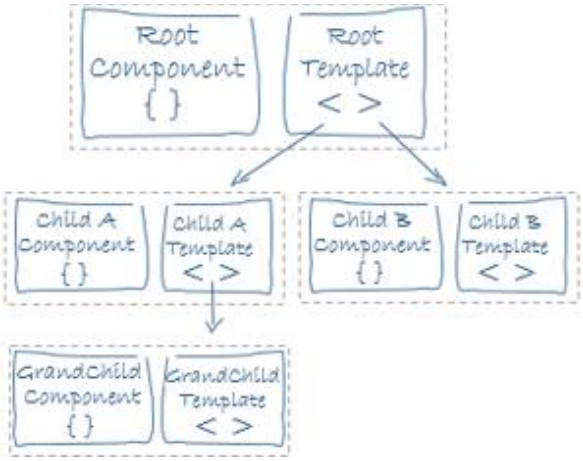
Slika 2.5. *Epic Bundle stranica*

3. PREGLED KORIŠTENIH TEHNOLOGIJA

3.1. Angular

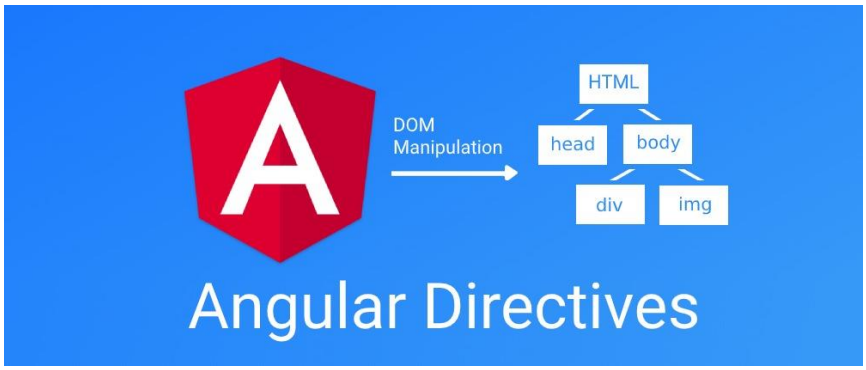
Angular je otvoreni, besplatni okvir za izgradnju web aplikacija kojeg je razvio Google. Baziran je na *TypeScriptu*, jeziku koji pruža strogu tipizaciju (u odnosu na sestrinski jezik *JavaScript*) i dodatne značajke za razvoj web aplikacija. Neke od dodatnih značajki su: koncept komponenti, direktive, servisi, moduli i usmjeravanje (engl. *routing*). [4]

Komponente (slika 3.1.) su modularni blokovi koji kombiniraju HTML, CSS i *TypeScript* logiku kako bi prikazali određeni dio korisničkog sučelja. Komponente su samostalne, ponovno upotrebljive i mogu se koristiti za izgradnju složenih aplikacija. Sastoje se od *TypeScript* klasa koji definira logiku i predloške koji definiraju izgled korisničkog sučelja. Predlošci se koriste za prikazivanje podataka i vezivanje događaja. [5]



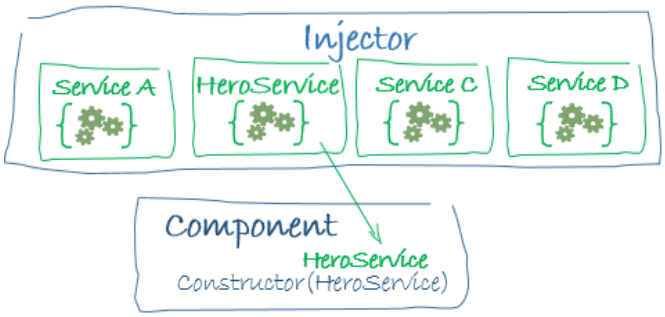
Slika 3.1. *Angular komponente*

Nadalje, direktive (slika 3.2.) proširuju mogućnosti HTML-a i koriste se za manipulaciju korisničkog sučelja, pružanje dodatnih funkcionalnosti i interakciju s komponentama. Postoje tri vrste direktiva: komponentne, strukturne i atributne. Komponentne su najčešće korištene te predstavljaju komponentu s vlastitom šablonom i logikom koja se može koristiti kao oznaka u HTML-u. Strukturne direktive mijenjaju DOM (*Document Object Model*) dodavanjem, uklanjanjem ili zamjenom elemenata u HTML-u. Naposlijetku, atributne direktive mijenjaju izgled i ponašanje elemenata dodavanjem ili promjenom atributa, poput boje pozadine elementa i dr.



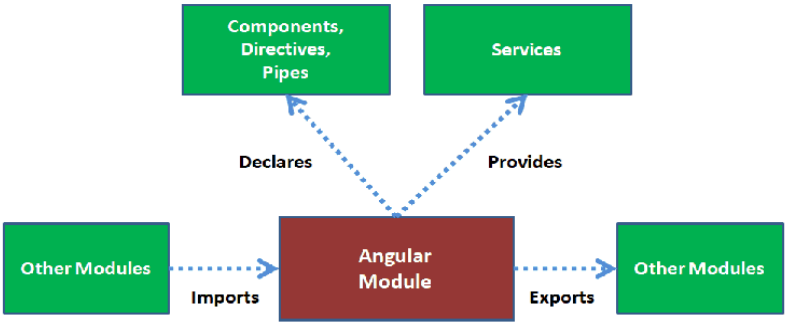
Slika 3.2. *Angular direktiva*

Servis se može opisati kao funkcionalnost koja se dijeli između komponenti. Tehnički, to su klase koje pružaju funkcionalnosti koje se mogu ponovno iskoristiti u cijeloj aplikaciji. Postoji par glavnih koncepata korištenja, od kojih je prvi ubrizgavanje ovisnosti. Servisi se ubrizgavaju (engl. *inject*) u komponente (slika 3.3.), druge servise ili direktive kako bi se iskoristile njihove funkcionalnosti. Idući koncept je ponovna uporaba, odnosno izdvajanje zajedničke logike ili funkcionalnosti iz komponenti u servis, gdje će biti lakše upravljanje same funkcionalnosti. Nadalje, servisi su najbolji primjer komunikacije s *back end*-om, jer je sama funkcionalnost izdvojena i olakšano testiranje servisa neovisno o prikazu na sučelju. Naposlijetku, servisi mogu držati stanje aplikacije i dijeliti podatke među različitim dijelovima aplikacije. Na primjer, servis za upravljanje korisničkom prijavom i autentifikacijom može pratiti trenutno prijavljenog korisnika i dijeliti tu informaciju s ostalim dijelovima aplikacije koji je trebaju.



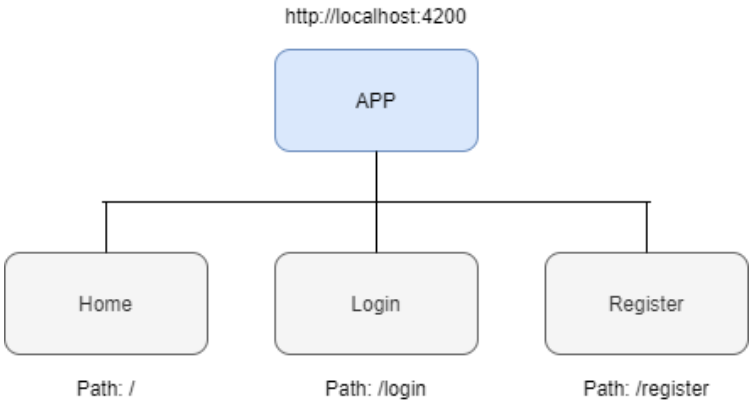
Slika 3.3. *Ubrizgavanje servisa u komponentu*

Zatim, Angular spominje module (slika 3.4.) koje koristi kao za organizaciju i dijeljenje komponenti, direktiva, servisa i drugih resursa. Moduli pružaju jasno definiranu granicu za funkcionalnosti i olakšavaju organizaciju koda. Vrlo su važan koncept za razumijevanje prilikom izgradnje većih aplikacija u Angularu, jer olakšavaju dijeljenje resursa i organizaciju aplikacije, što su vrlo bitne prednosti kod izrade većih aplikacija.



Slika 3.4. *Angular modul*

Zadnji, ali možda jedan od najvažnijih koncepta Angulara je usmjeravanje (engl. *routing*). Angular ima podršku za spomenuti vodeći koncept, što omogućuje definiranje ruta (slika 3.5.) i navigaciju između različitih dijelova aplikacije bez osvježavanja cijele stranice. Omogućuje se izgradnja više straničnih aplikacija (engl. SPA - *Single Page Applications*), što je noviji koncept izgradnje i rada web stranica. Takva aplikacija komunicira s korisnikom dinamički prepisivanjem trenutne web stranice s novim podacima web poslužitelja, umjesto prijašnjeg koncepta učitavanja cijele nove stranice. Usmjeravanje je zapravo zaduženo za omogućavanje realizacije koncepta SPA aplikacije. [6][7]



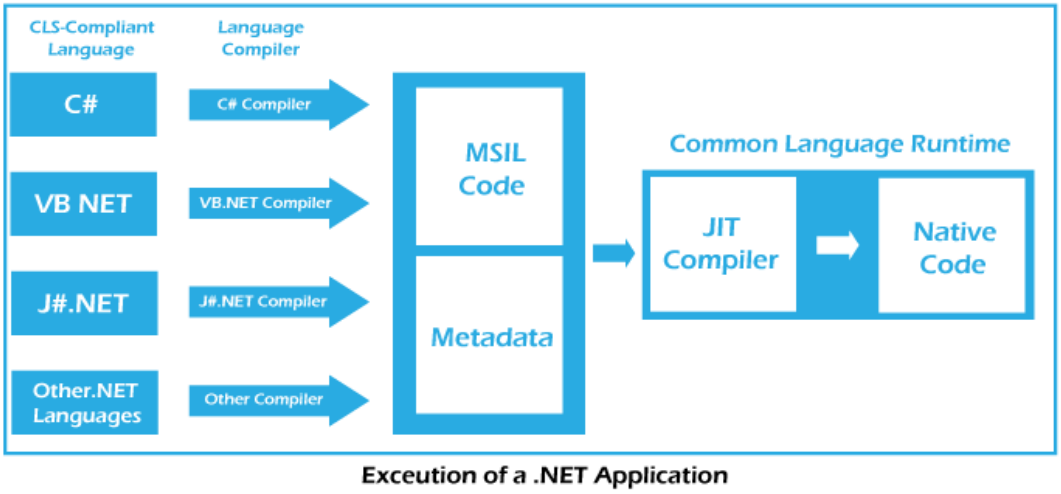
Slika 3.5. *Angular rute*

3.2. NET

.NET je razvojni okvir (engl. *framework*) razvijen od strane Microsofta koji pruža sredstva i alate za izgradnju raznovrsnih vrsta aplikacija, uključujući web aplikacije, desktop aplikacije, mobilne aplikacije, servise i mnoge druge.

Najvažnija sastavnica .NET-a zove se *Common Language Runtime* ili skraćeno CLR. To je okruženje za izvršavanje .NET aplikacija koje pruža mnoge značajke, kao što su upravljanje

memorijom, prikupljanje otpada (engl. *garbage collection*), sigurnosni mehanizmi, *just-in-time* ili skraćeno JIT kompilacija i druge (slika 3.6.). Upravljanje memorijom se odvija preko sustava za prikupljanje otpada koji automatski prati i oslobađa memoriju koju više aplikacija ne koristi. Time je olakšan rad memorijom i spriječeno njeno curenje, kao i drugi problemi vezani za upravljanje memorijom. Nadalje, CLR pruža slojeve sigurnosti koji štite aplikaciju od zlonamjernog koda i potencijalnih prijetnji. Ovi slojevi uključuju mehanizme kao što su *Code Access Security* ili skraćeno CAS, i *Role-Based* Security ili skraćeno RBS, koji omogućuju kontrolu pristupa resursima i ograničavanje privilegija. Događa se i JIT kompilacija, odnosno pretvorba koda napisanog u drugim jezicima, kao što su C#, u strojni jezik koji može izravno izvršavati računalni *hardware*. Ovakva kompilacija se odvija dinamički tijekom izvršavanja programa, što optimizira performanse pružajući dodatnu sigurnost. [8]

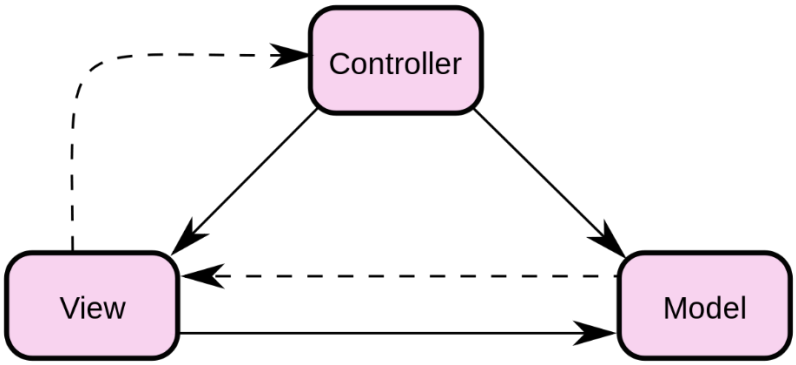


Slika 3.6. *Common Language Runtimeu .NET-u*

Iduća sastavnica je sam .NET Framework, odnosno originalna verzija .NET-a koja je dostupna za razvoj aplikacija na Windows platformi. Pruža širok skup biblioteka i klasi koje olakšavaju pri razvoju aplikacija. Koristi se jezik C# kao glavni jezik za razvoj aplikacija. Na drugu stranu, postoji i .NET Core, odnosno verzija .NET-a otvorenog koda (engl. *open source*) koja je dostupna za razvoj aplikacija na Windowsu, Linuxu i macOS-u. Razlika ove dvije sastavnice leži u njihovoj platformskoj podršci, funkcionalnostima i dostupnosti: .NET Core je platformski neovisan, dok je .NET namijenjen za razvoj aplikacija na Windows platformi.

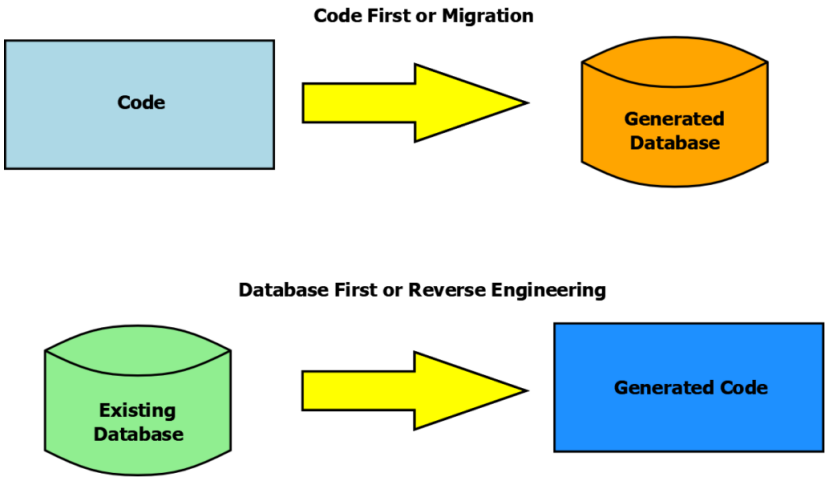
Osim .NET Core-a i .NET Framework-a, postoji i ASP.NET sastavnica koja se koristi za izgradnju web aplikacija. Omogućuje razvoj dinamičnih web aplikacija, web servisa i web stranica koristeći programski jezik C#, Visual Basic.NET (VB.NET) i dr. ASP.NET koristi *server-side* izvršavanje, što znači da se logika aplikacije izvršava na web poslužitelju prije prikaza rezultata na klijentskom

uređaju. Aplikacija su tada dinamične i složene, te mogu obrađivati podatke, pristupati bazama podataka i izvršavati druge zadatke na serveru. Nadalje, ASP.NET pruža bogat skup unaprijed izrađenih mrežnih kontrola koje olakšavaju izgradnju web stranica, pružajući prikladne funkcionalnosti poput upravljanja unosima, prikaza podataka, obrade događaja i dr. Također, pružena je MVC (engl. *Model-View-Controller*) arhitektura ASP.NET MVC-om (slika 3.7.). Takva arhitektura olakšava organizaciju koda, testiranje, no i održivost aplikacije. [9]



Slika 3.7. *MVC arhitektura*

Zadnja sastavnica bitna za spomenuti je Entity Framework, *Object-Relational Mapping* ili skraćeno ORM alat koji se koristi pristup i manipulaciju podacima u relacijskim bazama podataka. On olakšava rad s bazama podataka i pruža apstrakciju nad složenostima izravnog pisanja SQL upita, što je glavni koncept *code-first* pristupa gradnje aplikacije. Kod ovog pristupa, glavni fokus je na programiranju i definiranju modela aplikacije, a baza podataka se generira automatski na temelju tog modela. Važno je napomenuti da *code-first* pristup ne znači da se baza podataka potpuno ignorira. Ona i dalje igra važnu ulogu u funkcioniranju aplikacije, ali njezina struktura se generira i održava kroz kod i ORM alate, poput Entity Framework-a. Nadalje, alat se usko integrira s jezikom upita *Language Integrated Query* ili skraćeno LINQ. To omogućuje pisanje upita nad objektima, poput filtriranja, sortiranja i grupiranja, koristeći LINQ sintaksu, a Entity Framework automatski prevede te upite u odgovarajuće SQL upite. Jedna od glavnih značajki alata je upravljanje vezama i transakcijama, odnosno lako definiranje veza između entiteta i održavanje dosljednosti tih veza tijekom operacija nad podacima. Iako je *code-first* prvi najkorišteniji pristup ovog alata, omogućeni su i drugi pristupi poput *database-first* i *model-first* (slika 3.8.). [10]

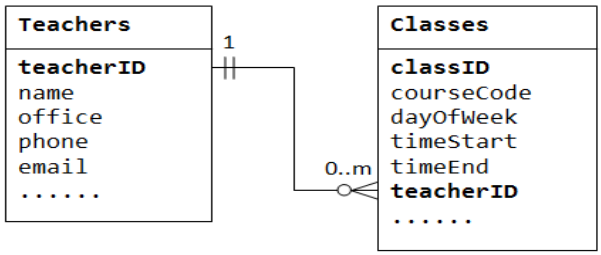


Slika 3.8. *Razlika između code-first i database-first pristupa*

3.3. MySQL

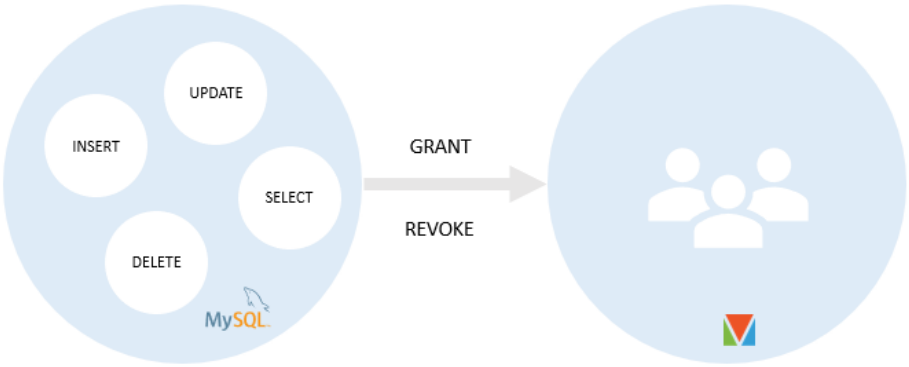
MySQL je jedan od najpopularnijih otvorenih sustava za upravljanje bazama podataka koji se temelji na jeziku SQL (*Structured Query Language*). To je relacijski sustav koji pruža pouzdanu, brzu i skalabilnu platformu za pohranu, upravljanje i pristup podacima.

MySQL koristi relacijski model podataka (slika 3.9.), gdje su podaci organizirani u tablicama koje imaju redove i stupce. Podaci su organizirani u tablicama koje se nazivaju relacije. Svaka relacija ima definirane stupce koji predstavljaju pojedinačne atribute ili karakteristike podataka, dok redovi predstavljaju pojedinačne zapise ili instance podataka. Omogućeno je definiranje veza ili odnosa između entiteta putem primarnih i stranih ključeva. Primarni ključ je jedinstven identifikator za svaki zapis u tablici, odnosno stupac ili skup stupaca koji identificiraju svaki redak u tablici. Najvažniji zadatak primarnog ključa je jedinstvenost, Svaka tablica u relacijskom modelu podataka treba imati primarni ključ. Nadalje, strani ključ je stupac ili skup stupaca u jednoj tablici koji uspostavlja vezu s primarnim ključem u drugoj tablici. Zadatak ključa je povezivanje između tablica i definiranje njihovih odnosa. Strani ključ omogućuje referencijalnu integritet, što znači da se podaci u tablicama održavaju dosljednima prema definiranim vezama.



Slika 3.9. *Primjer relacijskog modela podataka*

MySQL je multiplatformski sustav koji podržava širok raspon operacijskih sustava, kao što su Windows, Linux, macOS i dr. Također, MySQL je vrlo brz te ima velike performanse jer pruža učinkovit sustav za obradu i izvršavanje SQL upita. No, jedna od najvažnijih značajki sustava koji radi s podacima je, naravno, sigurnost. MySQL pruža nekoliko mehanizama za pouzdanost i sigurnost podataka, kao autentifikacija korisnika (slika 3.10.) putem korisničkog imena i lozinke, sigurnosnih certifikata, vanjskih mehanizama autentifikacije ili putem *Lightweight Directory Access Protocol* ili skraćeno LDAP sustavima. Idući mehanizam sigurnosti je enkripcija samih podataka kako bi se zaštitili osjetljivi podaci od neovlaštenog pristupa. Podatke je moguće kriptirati na razini transporta od klijenta do poslužitelja putem SSL/TLS (*Secure Sockets Layer/Transport Layer Security*) protokola, Također se koristi kriptiranje na razini pohrane podataka kako bi se zaštitili podaci pohranjeni na disku. Nadalje, MySQL podržava transakcije koje omogućuju grupiranje više upita u jednu logičku jedinicu. U pitanju je dosljednost podataka, čime se osigurava da promjene na podacima budu trajne te da je moguća potvrda ili poništenje promjene u slučaju greške.



Slika 3.10. *Upravljanje ulogama u MySQL-u*

Naposlijetku, potrebno je spomenuti skalabilnost u kontekstu MySQL-a. Postignuta je vertikalna skalabilnost, odnosno sposobnost sustava za povećanjem snage obrade dodavanjem snažnijeg *hardware*-a na postojeći poslužitelj. Također, postignuta je i horizontalna skalabilnost, sposobnost MySQL-a za repliciranjem podataka na više poslužitelja. Time je omogućena distribucija podataka i povećanje dostupnosti podataka. Zatim, koristi se *Load Balancing* za ravnomjernu raspodjelu opterećenja između poslužitelja, te se može postaviti korištenje rješenja poput proxy poslužitelji ili distribuirani sustavi za upravljanje opterećenjem.

4. PRIJEDLOG ZAHTJEVA WEB APLIKACIJE

4.1. Prijedlog poslovne logike platforme

Platforma za kupovanje kolekcija društvenih igara je nazvana *MuchBunch*. No, kako je riječ o platformi, ključna razlika između platforme i web stranice leži u obujmu usluga i funkcionalnosti koje pružaju. Prema [11], web stranica je pojedinačna online stranica s ograničenim funkcionalnostima, dok je platforma kompleksnija i može uključivati mnoge različite komponente, uloge korisnika i usluge kako bi podržala širok spektar interakcija i aktivnosti korisnika. Također, platforma može sadržavati integraciju s drugim sustavima, poput elektroničke pošte, društvene mreže i dr. Potrebno je sadržavati razne funkcionalnosti, no u isto vrijeme predstaviti ugodno i pojednostavljeno korisničko iskustvo krajnjem korisniku. U okviru diplomskog rada, ciljani krajnji korisnici su kupci, no također i tvtke.

Vodeći se načelom „manje je više“, potrebno je pružiti intuitivno i pojednostavljeno sučelje koje će privući korisnike, a ne komplicirano s puno funkcionalnosti. Potrebno je definirati paletu boja koje će se provlačiti kroz komponente sučelja kao tema aplikacije, no boje moraju biti neutralne i ugodne. Kako bi se platforma istaknula i dobila dojam ozbiljnosti, potrebno je kreirati jedinstvenu logo sliku i koristiti ju kroz platformu.

Nadalje, pošto postoje uloge korisnika, potrebno je kreirati autentikaciju korisnika u aplikaciju. Uloge su krajnji kupac, tvrtka i administrator aplikacije. Svaka od uloga ima razinu pristupa sadržaju aplikacije, koje je potrebno definirati. Krajni kupac može napraviti račun na platformi, odnosno prijaviti se u aplikaciju. Potom, kupac može pretraživati kolekcije društvenih igara te ih raspoznavati po dvije stavke: tematske i netematske kolekcije. Tematske su one koje sama platforma kreira (administrator) na mjesečnoj bazi, te ih kupac može kupovati po fiksnoj cijeni – neovisno koja tvrtka je napravila koju kolekciju. Nakon kupnje, kupac može posjetiti svoj kreirani profil, gdje će se nalaziti sve njegove dotadašnje kupnje, kao i osnovne informacije.

Tvrtka kao uloga korisnika aplikacije mora sadržavati sve funkcionalnosti krajnjeg kupca, no i samo kreiranje kolekcija. Kako bi tvrtka znala kako kreirati kolekcije, koja je nova tema taj mjesec i kolika je fiksna cijena nove kolekcije, administrator mora moći obavijestiti tvrtku putem platforme. Predlaže se rješenje slanjem elektroničke pošte tvrtkama kada administrator kreira nadolazeću temu u samoj platformi, te zada specifikacije kolekcije s tom temom. Nakon primljene pošte, tvrtka mora moći unijeti same proizvode koje planira koristiti u kolekcijama te specifikacije istih, poput dostupne količine proizvoda u skladištu, bazna cijena i sl. Tek nakon kreiranja

proizvoda u vlastitom inventaru na platformi, tvrtka može kreirati kolekciju po specifikacijama administratora. Nadolazeća tema, kao i povezane kolekcije, ne postanu javne krajnjem kupcu kad budu kreirane, nego kada administrator odluči – odnosno, nakon mjesec dana.

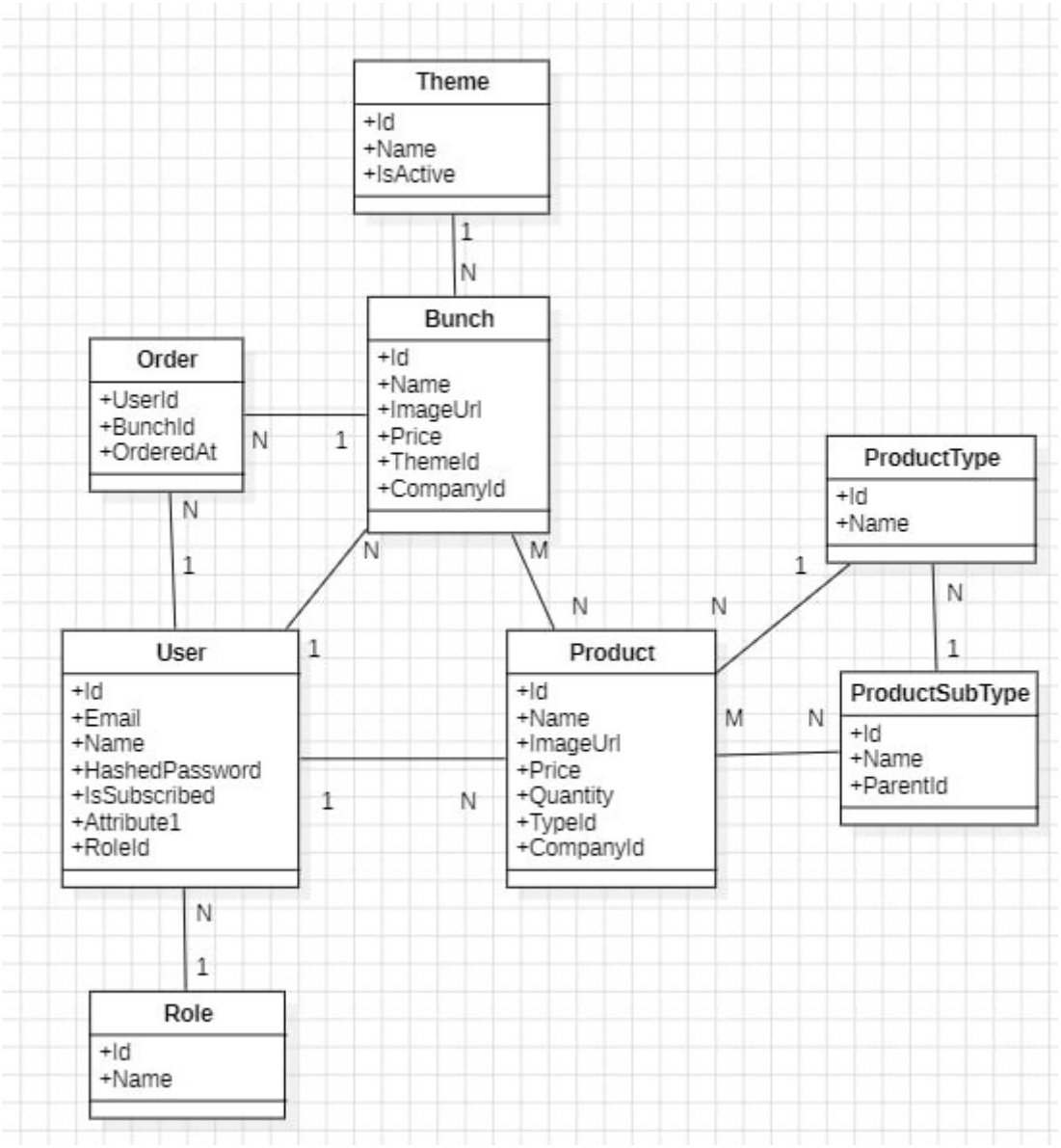
Administrator uloga mora obuhvaćati sve što i uloga kupca i administratora, te još više. Administrator mora moći upravljati specifikacijama proizvoda, poput tipovi i podtipovi proizvoda (je li proizvod društvena igra, neki dodatak za igru i sl.) te ulogama korisnika u aplikaciji. Jednom riječju, administrator mora moći upravljati šifrarnicima. Osim toga, mora također moći kreirati nadolazeću temu putem platforme i definirati sve potrebno tvrtkama pri kreaciji kolekcije.

Sustav kupovanja kolekcija treba funkcionirati na način „tko prvi – njegovo“, odnosno svaka narudžba nosi po jednu instancu svakog proizvoda, te se količina smanjuje ekvivalentno zbroju naručenih kolekcija. Kako tvrtka može isti proizvod staviti u više kolekcija, čim jednog od proizvoda nema u skladištu, potrebno je zabraniti kupovinu svih kolekcija gdje se spominje instanca proizvoda. Zbog tog istog razloga, potrebno je uvesti pojam pretplate. Pojam pretplate će biti predstavljen preko pretplate na elektroničku poštu, tj. korisnici koji su preplaćeni na platformu prvi dobivaju obavijesti kako su došle nove kolekcije – eliminirajući brigu oko isteka zaliha.

Naposlijetku, potrebno je tvrtkama omogućiti kreiranje netematskih kolekcija zbog raznolikosti sadržaja i opcija koje platforma predstavlja kupcima. Takve kolekcije ne trebaju imati zadane specifikacije kojih se moraju držati, stoga i takve kolekcije mogu biti jednistvene te jednostavno, zabavne i inovativne.

4.2. Prijedlog strukture baze podataka

Odmah na prvi dojam poslovne logike platforme, ističu se bazni entiteti koji će postati tablice u bazi, a to su korisnik, proizvod te kolekcija. Kako bi pobliže definirati bazne entitete, potrebno je definirati šifrarnike tip i podtip proizvoda, uloga korisnika, tema kolekcije te narudžba korisnika. Sve tablice je potrebno međusobno povezati po pravilima normalne forme relacijskih baza podataka. Prijedlog entiteta i relacija nalazi se na slici 4.1.



Slika 4.1. *Prijedlog strukture baze podataka*

5. PRIJEDLOG PROGRAMSKOG RJEŠENJA WEB APLIKACIJE

5.1. Postavljanje razvojnih okolina

5.1.1. Postavljanje Angular okruženja

Kako bi krenuo razvoj sučelja aplikacije, mora se postaviti okruženje Angular projekta. Postavljanje okruženja se događa preko *Node.js* i *Node Package Manager (npm)* paketa. *Node.js* se koristi za postavljanje i upravljanje poslužiteljskom stranom aplikacije koja pruža programsko sučelje aplikacije (engl. *Application Programming Interface*, API), koje dalje Angular aplikacija koristi za komunikaciju s poslužiteljem. *Node Package Manager* se koristi za upravljanje ovisnostima i paketima koje Angular aplikacija koristi tijekom razvoja. Nakon instalacije obje tehnologije, prelazi se na instalaciju Angular CLI (*Command Line Interface*). Upisivanjem slijedeće naredbe (programski kod 5.1.) u naredbeni redak instalira se alat koji omogućuje listu korisnih Angular naredbi, poput kreiranja nove komponente, posluživanje aplikacije i dr.

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

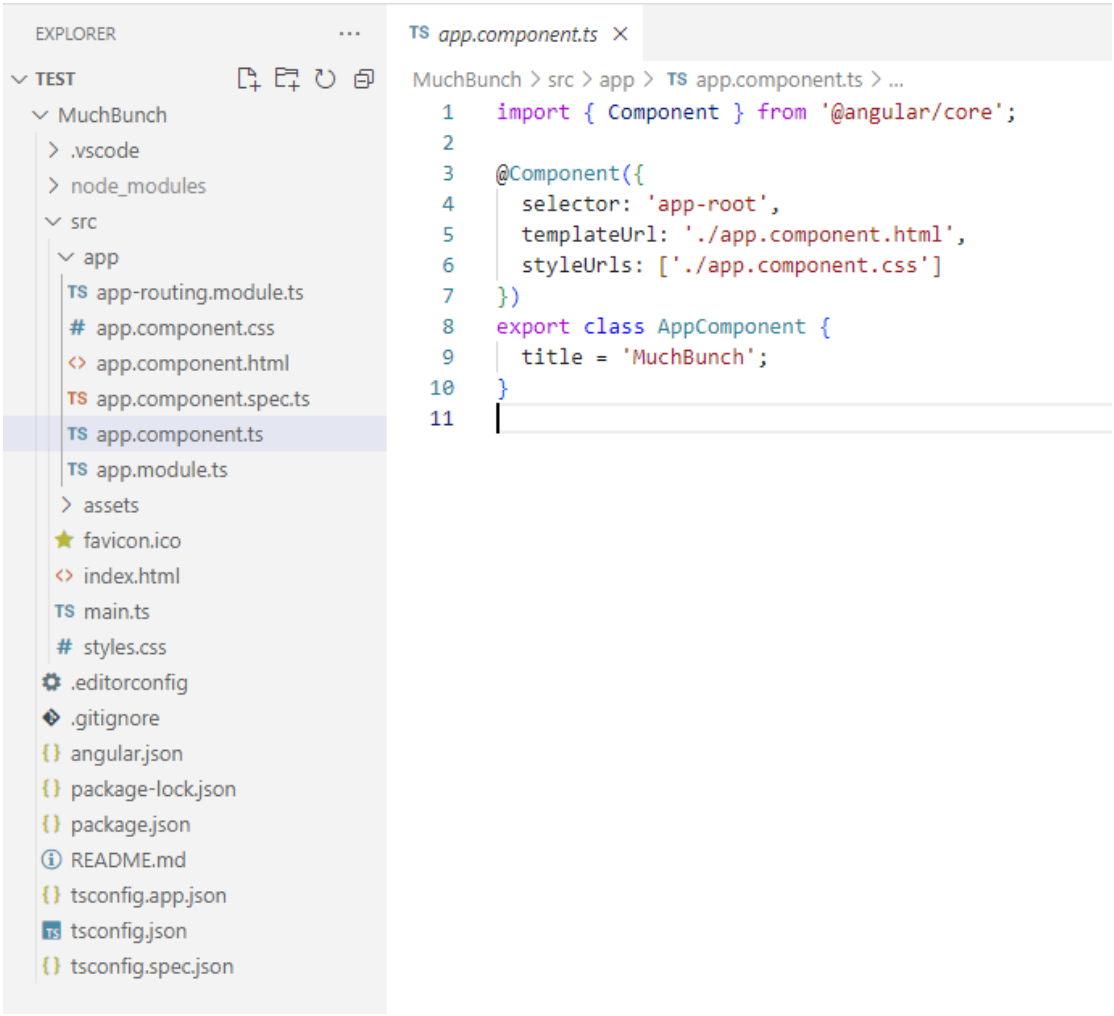
Programski kod 5.1. *Instalacija Angular CLI*

Zatim, kreira se nova Angular aplikacija naziva *MuchBunch* upisivanjem slijedeće naredbe u terminal (programski kod 5.2.).

```
ng new MuchBunch
```

Programski kod 5.2. *Stvaranje nove Angular aplikacije*

Naposlijetku, dobije se osnovna struktura Angular projekta, koja se sastoji od početne glavne komponente nazvane *app.component.* Svaka Angular komponenta se sastoji od tri dokumenta od kojih svaki ima drugu funkcionalnost. HTML dokument komponente se brine za korisničko sučelje te za sve elemente s kojima korisnik dolazi u dodir, napisan HTML jezikom. CSS dokument stilizira HTML elemente, dok TypeScript dokument opisuje funkcionalnosti HTML elementima. Nadalje, svaki Angular projekt posjeduje glavni modul, već sadržan u baznoj aplikaciji. Modul je mjesto gdje se definiraju sve kreirane komponente aplikacije, kako bi Angular znao i očekivao što raditi s određenim komponentama. Uz osnovni modul, kreiran je modul za usmjeravanje (engl. *routing*). Taj modul sadrži popis svih ruta korištenih u aplikaciji te govori Angularu kako doći do određene rute, koju komponentu učitati, je li ta ruta zaštićena ili ne i dr. Osnovna struktura početne Angular aplikacije dana je na slici 5.3.



Slika 5.3. *Osnovna struktura početne Angular aplikacije*

5.1.2. Postavljanje *NG-Zorro* biblioteke

NG-Zorro je popularna biblioteka komponenti za Angular, koja pruža razne korisne komponente korisničkog sučelja i alate za izradu modernih web aplikacija. Posebno je kreiran za integraciju s Angular okolinom te olakšava razvoj interaktivnih i profesionalnih korisničkih sučelja. Korisnička sučelja koja sadrži ova biblioteka su: gumbi, obrasci, kartice, navigacijske trake, dijalozi, ikone i mnoge druge. Kako bi se ova biblioteka nadodala u Angular projekt, potrebno je upisati slijedeću naredbu u terminal (programski kod 5.4.):

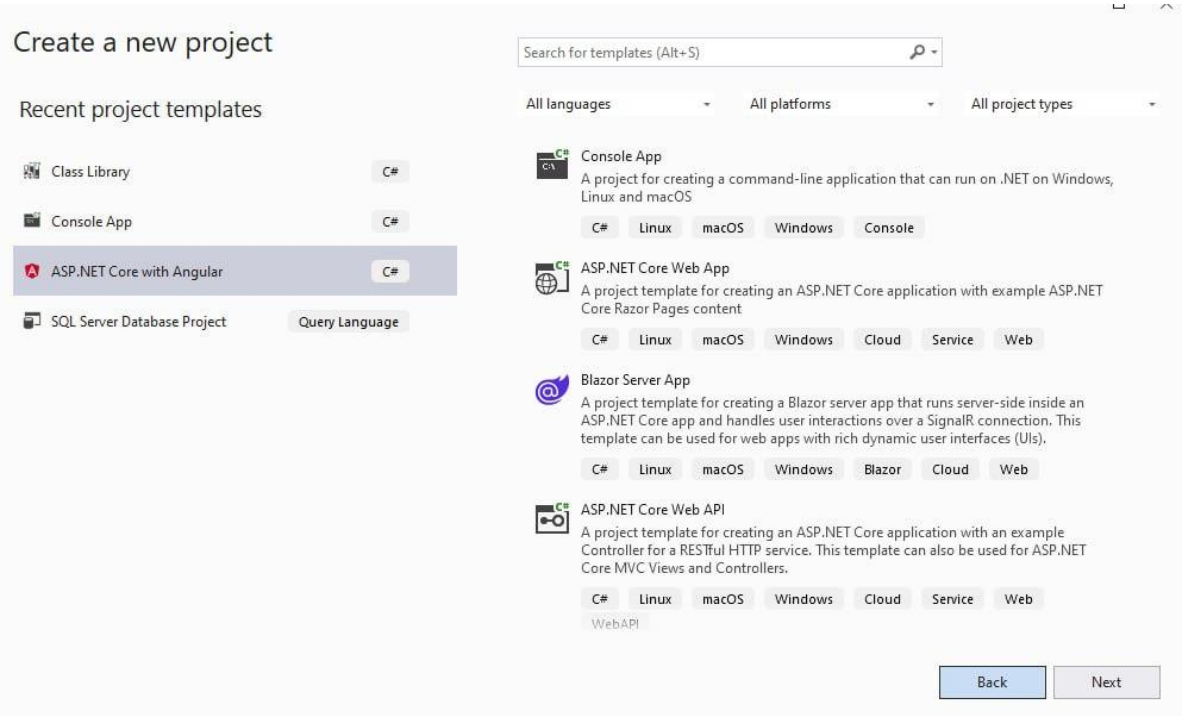
```
npm install ng-zorro-antd --save
```

Programski kod 5.4. *Instalacija NG-Zorro biblioteke*

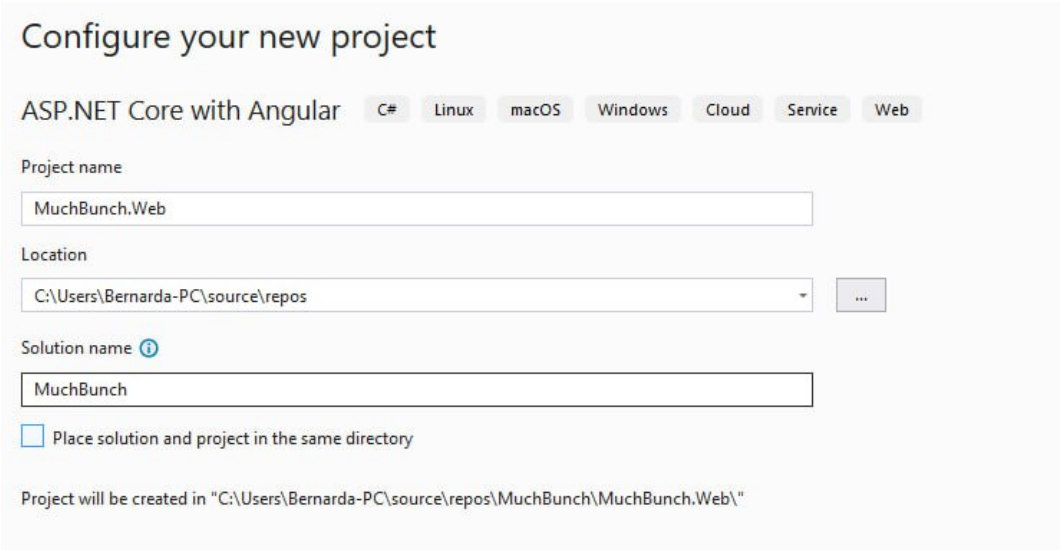
Nakon instalacije, radi bolje organizacije koda, kreira se novi modul u Angular aplikaciji koji sadrži sve definicije NG-Zorro komponenti. Takav modul je potrebno spomenuti u glavnom modulu aplikacije.

5.1.3. Postavljanje .NET okoline

Postavljanje poslužiteljske strane aplikacije kreće kreiranjem novog ASP.NET Core projekta s Angularom (slike 5.5. i 5.6.)



Slika 5.5. *Stvaranje novog projekta*



Slika 5.6. *Stvaranje novog projekta - nastavak*

Prvi projekt u .NET 6.0 rješenju (engl. *solution*) koji nastaje je *MuchBunch.Web*. Služi za definiranje API poziva, odnosno koja *http zahtjev* (engl. *http request*) metoda odlazi na koju rutu i koji posao se obavlja, te također za pozivanje logike servisa. Također, to je mjesto gdje se nalazi sva definirana logika za zabranu pristupa rutama. *Web* projekt je zapravo onaj u kojemu se nalazi Angular aplikacija, odnosno klijentska aplikacija koja se poslužuje na web-u.

Dalje, kreiraju se novi projekti u rješenju zvani *MuchBunch.EF* i *MuchBunch.Service*. *Entity Framework (EF)* projekt služi za modeliranje baze podataka u kojom se nalaze sve definicije modela, migracije i dvije klase. Klasa koja drži trenutnu verziju baze, odnosno njenu sliku se naziva *database snapshot* i klasa koja pruža sve entitete i njihove veze iz baze servisnom sloju aplikacije se naziva *database context*. *Entity Framework Core* okvir koji je potreban za rad *EF* projekta instalira se NuGet paketima te omogućuje kreiranje baze podataka kroz kod (engl. *code-first approach),* stoga je taj pristup korišten u projektu. [12]

Servisni sloj opisuje *Service* projekt koji drži poslovnu logiku aplikacije. Zaslužan je za sve funkcionalnosti nad podacima koje dolaze preko API poziva. Za uspješan rad projekta, potrebno je instalirati *AutoMapper* biblioteku, kako bi pojednostavili proces kopiranja podataka iz jednog objekta u drugi, *MailKit* biblioteku zbog mogućnosti rada s električnom poštom putem SMTP-a, te biblioteku *FluentValidation* kako bi se definirala pravila validacije podataka. Ovim korakom je kreirana osnovna poslužiteljska .NET aplikacija.

5.2. Povezivanje klijentske i poslužiteljske aplikacije

Pokretanjem poslužiteljske aplikacije, ona se otvara na portu koji je postavljen u JSON datoteci koja konfigurira postavke posluživanja aplikacije (*launchSettings.json,* slika 5.7.). Zatim, pokreće se klijentska aplikacija na portu definiranom u *csproj* datoteci, koja drži osnovne informacije o samome projektu i poslužiteljskoj aplikaciji (*MuchBunch.Web.csproj*). Od porta na kojem je definirana poslužiteljska aplikacija se očekuje komunikacija s istom, što se omogućuje CORS politikom.

```
},
"profiles": {
  "MuchBunch.Web": {
    "commandName": "Project",
    "launchBrowser": true,
    "applicationUrl": "https://localhost:7299;http://localhost:5164",
    "environmentVariables": {
      "ASPNETCORE_ENVIRONMENT": "Development",
      "ASPNETCORE_HOSTINGSTARTUPASSEMBLIES": "Microsoft.AspNetCore.SpaProxy"
    }
  },
```

Slika 5.7. *Datoteka launchSettings.json*

Kako bi povezali u suprotnom smjeru, odnosno klijentsku aplikaciju s poslužiteljskom, potrebno je definirati baznu rutu na koju će se slati svi API pozivi, odnosno port gdje se nalazi poslužiteljska aplikacija. Definiranje se kreira u *environment.ts* datotekama koje su dio konfiguracijskog sustava u Angularu (programski kod 5.8.). Primarno služe radi postavljanja različitih konfiguracijskih parametara koji ovise o različitim okolinama izvršavanja Angular aplikacije, poput produkcijske, razvojne i testne okoline. Za svaku od okolina se kreira zasebna *environment.ts* datoteka, te se u glavnoj konfiguracijskoj datoteci *angular.json* definira u kojoj okolini se određena *environment* datoteka koristiti.

src > environments > TS environment.ts > ...

```ts
// This file can be replaced during build by using the `fileReplacements` array.
// `ng build` replaces `environment.ts` with `environment.prod.ts`.
// The list of file replacements can be found in `angular.json`.

export const environment = {
  production: false,
  baseUrl: 'https://localhost:7299',
};

/*
 * For easier debugging in development mode, you can import the following file
 * to ignore zone related error stack frames such as `zone.run`, `zoneDelegate.invokeTask`.
 *
 * This import should be commented out in production mode because it will have a negative impact
 * on performance if an error is thrown.
 */
// import 'zone.js/plugins/zone-error';  // Included with Angular CLI.

```

Programski kod 5.8. *Datoteka environment.ts*

5.3. *Entity Framework* i kreiranje baze podataka

Prateći danu strukturu baze podataka sa slike 4.1., kreira se baza podataka u *Entity Framework*-u. Definiraju se modeli na temelju kojih će se stvarati entiteti u bazi, odnosno tablice. Svaka tablica koja ima atribut tipa integer zvan Id (int Id), automatski postaje primarni ključ te tablice. Koristeći spomenuti *code-first* pristup kreiranja baze, potrebno je kroz kod definirati veze između tablica. Ako neka tablica ima ključ na drugu koji je u skladu s konvencijama imenovanja, *Entity Framework* prepoznaje to kao strani ključ na drugu tablicu te kreira točnu vezu. U slučaju da je u pitanju više-naspram-više veza (M-N), kreira se dodatna pivot tablica, po pravilima normalne forme (NF). Samo one veze ili ograničenja koja odstupaju od standardnih pretpostavki o tablici ili vezi se trebaju definirati u *database context* klasi. Tako se, primjerice, postavljaju dodatni atributi u automatski generiranoj M-N tablici. Također, potrebno je pokriti slučaj što će se dogoditi obriše li se redak u tablici koja ima povezanosti s drugim tablicama definiranjem ponašanja brisanja i

ostalih redova u drugim povezanim tablicama. Programski kod definicije modela korisnika i njegovih atributa dan je u programskom kodu 5.9.

```
namespace MuchBunch.EF.Database.Models
{
    public class User
    {
        public int Id { get; set; }
        [Required]
        public string Email { get; set; }
        [Required]
        public string Name { get; set; }
        [Required]
        public string HashedPassword { get; set; }
        public bool IsSubscribed { get; set; }
        [Required]
        public int RoleId { get; set; }
        public Role Role { get; set; }
        public ICollection<Product>? Products { get; set; }
        public ICollection<Bunch>? Bunches { get; set; }
        public ICollection<Bunch>? BoughtBunches { get; set; }
        public ICollection<Order>? Orders { get; set; }
    }
}
```

Programski kod 5.9. *Model korisnika*

Nakon kreiranja modela i *database context* klase, spreman je idući korak migracije koda u bazu podataka. Kako bi se sve kodom napisane izmjene na bazi primijenile, potrebno je dodati novu migraciju naredbom *Add-Migration*. *Entity Framework* potom uspoređuje trenutno stanje modela i pravila postavljenih u *database context* klasi s trenutnom slikom baze, odnosno klasom *database snapshot*. Na temelju rezultata usporedbe, kreira se migracija koja ažurira bazu podataka na novo stanje.

5.4. Autentikacija

Proces postavljanja autentikacije kreće na poslužiteljskoj aplikaciji. Autorizacija i autentikacija se postavljaju u ključnoj klasi *Program.cs* koja predstavlja ulaznu točku prilikom izvođena aplikacije (programski kod 5.10.)

Pokretanjem aplikacije, pokreće se *Program.cs* u kojoj se izvršava *Main* metoda. Metoda inicijalizira aplikaciju, postavlja okolinu i poziva glavnu logiku aplikacije. Sukladno tome, u njoj se određuje način na koji će se autenticirati i autorizirati zahtjev. Odabrana je implementacija JSON Web Tokena, ili skraćeno JWT.

```
builder.Services.AddAuthentication(x =>
{
    x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(x =>
{
    x.TokenValidationParameters = new TokenValidationParameters
    {
        ValidIssuer = jwtSettings.Issuer,
        ValidAudience = jwtSettings.Audience,
        IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSettings.Key)),
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        RoleClaimType = ClaimTypes.Role
    };
});
```

Programski kod 5.10. *Program.cs klasa*

JWT token je standardizirani način za predstavljanje informaciju između dvije strane kao JSON objekta. Token je siguran i samostalan te je zbog tog razloga odabrano njegovo korištenje. U tokenu se nalaze podaci o korisniku za kojeg se kreira te također, i njegov potpis. Svi podaci u tokenu se nalaze u ključ-vrijednost parovima nazvanim *Claims*. Običan tijek autorizacije putem JWT tokena glasi: prilikom autentikacije, poslužitelj izdaje JWT token klijentskoj strani, koja može uključiti isti u svaki zahtjev prema poslužitelju kao dio zaglavlja zahtjeva. Poslužitelj na temelju poglavlja tokena provjerava potpis tokena kako bi osigurao autentičnost i nepromjenjivost tokena. Ako je potpis valjan, poslužitelj izvlači korisničke informacije i koristi ih pri autorizaciji i obradi zahtjeva. [13][14]

Autentikacija se odvija na temelju tri podatka iz konfiguracije projekta, *Audience, Issuer* i ekriptiranog tajnog ključa, te se sva tri podatka postavljaju u *appsettings.json* konfiguracijskoj datoteci. Na zaštićenim rutama, *middleware UseAuthentication* (programski kod 5.11) provjerava spomenute parove ključ-vrijednosti *Claims* i potpis tokena u zaglavlju. Ako se vrijednosti poklapaju s očekivanim, autentikacija je uspješna. Jedan od spomenutih parova u tokenu je podatak o ulozi korisnika. Dodavanjem anotacije vidljive na programskom kodu 5.12, moguće je autorizacijom ograničiti pristup ruti samo na zahtjeve korisnika koji imaju ulogu administratora.

```
            app.UseHttpsRedirection();

            app.UseAuthentication();
            app.UseRouting();
            app.UseCors(GlobalConstants.CORS_POLICY);
            app.UseAuthorization();

            app.MapControllers();
```

Programski kod 5.11. *UseAuthentication middleware*

```
[Authorize(Roles = Role.Admin)]
[HttpGet("userProtected")]
public IActionResult GetUsersProtected()
{
    return Ok(userService.GetUsers());
}
```

Programski kod 5.12. *Zaštićena ruta dopuštena samo ulozi administratora*

S druge strane, na klijentskom dijelu, kreiraju se komponente registracije i prijave korisnika u aplikaciju. Nakon kreiranja HTML elemenata u koje će se upisivati podaci traženi kod autentikacije korisnika, poput imena, adrese korisničke pošte i lozinke, kreće se na pisanje servisa autentikacije. Uobičajeni slijed razvoja Angular klijentske aplikacije je slijedeći: kreira se komponenta, definiraju se HTML elementi i potrebni stilovi, te se opisuje funkcionalnost u TypeScript datoteci. Potom, kreiraju se servisi koji zapravo drže svu logiku klijentske aplikacije, odnosno sve definicije API poziva na poslužiteljski dio (programski kod 5.13.)

```
  login(loginModel: LoginBM): Observable<TokenDto> {
    return this.http.post<TokenDto>(`${this.serviceBaseUrl}/login`,
loginModel);
  }

  register(registerModel: RegisterBM): Observable<TokenDto> {
    return this.http.post<TokenDto>(
      `${this.serviceBaseUrl}/register`,
      registerModel
    );
  }

  logout() {
    localStorage.removeItem('token');
    localStorage.clear();
    this.user.next(null);
  }
```

Programski kod 5.13. *Metode servisa za autentikaciju*

Pošto je TypeScript striktan jezik u svjetlu deklaracije tipova, obavezno je i kreirati modele odgovora koje Angular dobije u rezultatu API poziva. Ti modeli moraju biti identični modelima na poslužiteljskoj strani, stoga se kreiraju dvije vrste modela: modeli s nastavkom BM (*binding model*) te DTO model (*data transfer object*). BM model se koristi u smjeru komunikacije s

klijentske strane na poslužiteljsku, dok je DTO s poslužiteljske na klijentsku stranu. Primjer DTO modela korisnika dan je u programskom kodu 5.15.

```
export class UserDTO {
  id: number;
  name: string;
  email: string;
  role: string;
  isSubscribed: boolean;
  orders?: UserOrderDto[];

  constructor(
    id: number,
    name: string,
    email: string,
    role: string,
    isSubscribed: boolean,
    orders?: UserOrderDto[]
  ) {
    this.id = id;
    this.name = name;
    this.email = email;
    this.role = role;
    this.isSubscribed = isSubscribed;
    this.orders = orders;
  }
}
```

Programski kod 5.15. *UserDTO model*

Nakon kreiranja servisa i modela, servise je potrebno ubrizgati u komponentu koja želi koristiti njegove metode, odnosno API pozive, u konstruktor klase komponente (programski kod 5.16.)

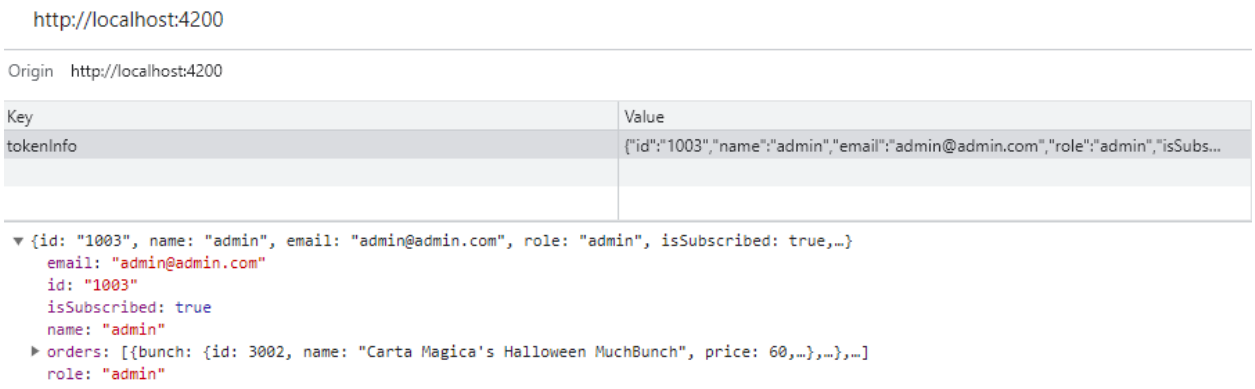
```
constructor(public authService: AuthService) {}

(..)

submitForm() {
 this.authService.login(formModel).subscribe((token) => {
        this.authService.storeTokenToLocalStorage(token.token);
}
```

Programski kod 5.16. *Ubrizgavanje servisa u komponentu i korištenje*

Nakon primitka tokena, sprema se u lokalnu pohranu preglednika kako bi se pratio prijavljeni korisnik (slika 5.17).



Slika 5.17. *Token spremljen u lokalnu pohranu preglednika*

Također, u servisu je kreirana instanca prijavljenog korisnika koja je dostupna svim komponentama koje imaju ubrizgani servis. Razlog tomu je brz pristup atributima trenutnog korisnika koje se koriste u ostalim dijelovima aplikacije. Kreira se kao *BehaviourSubject* (programski kod 4.15.), što je jedan od tipova subjekata (engl. *Subjects*) u Angularu i biblioteci RxJS (*Reactive Extensions for JavaScript*). Subjekti su posebni tip *Observable*-a koji omogućuje komunikaciju između različitih komponenata na principu pretplate – proces primanja obavijesti o emitiranim događajima ili podacima. Primjer u programskom kodu 5.18. prikazuje pretplatu na instancu korisnika te izvlačenje atributa korisnika, kako bi se provjerilo je li korisnikova uloga administrator.

```
public user = new BehaviorSubject<UserDTO | null>(null);

(..)

this.authService.user.subscribe((response) => {
   this.isAdmin = this.authService.getUserProperty('role') == 'admin';
});
```

Programski kod 5.18. *Kreiranje instance korisnika te pretplata na istu*

5.5. Upravljanje proizvodima i kolekcijama društvenih igara

Kako bi se ispunila poslovna logike aplikacije, potrebno je omogućiti unos i ostale povezane funkcionalnosti nad proizvodima tvrtke. Kreirani model proizvoda u *Entity Framework* projektu sadrži primarni ključ koji po standardnoj konvenciji imenovanja mora glasiti *int Id*. U slučaju bilo koje veze između entiteta, objekt ima pristup relacijskim podacima, odnosno zapisima ostalih tablica povezanim s trenutnom. Relacijski se podaci dohvaćaju samo ako se to eksplicitno traži, što omogućuje optimizirano dohvaćanje određenih relacijskih podataka koji su potrebni tijekom mapiranja u DTO model. Na temelju ključeva, anotacija i nulabilnosti tipova varijabli, kreiraju se

pravila za stvaranje tablice u bazi podataka. Sva odstupanja od standarda se ručno postavljaju u *database context* klasi, koja pruža pristup metodama za rad nad podacima.

Zahtjevi za rad nad podacima dolaze putem API poziva. Klase koje definiraju strukturu podataka koje API poziv prima imaju nastavak BM. U slučaju izmjene podataka, dobiveni podaci se prvobitno provjeravaju, te se model iz kontrolera predaje servisnom sloju. U slučaju da je zahtjev dohvaćanje podataka, u servisnom sloju se podaci dohvaćaju iz baze podataka, potom mapiraju u odgovarajući DTO model.

Kako bi tvrtka mogla kreirati kolekcije društvenih igara, mora posjedovati proizvode. Tvrtkama je omogućen unos vlastitih proizvoda i specifikacija, poput cijene i tipa proizvoda. Nakon unosa tipova i podtipova proizvoda od strane administratora (kao dio šifrarnika), tvrtke mogu davati tipove i podtipove proizvodima. Kako podtipovi proizvoda ovise o tipu, primjerice ako je tip društvena igra, podtip može biti Obiteljska igra, Strateška te Zabavna, napravljena je metoda koja se okida u trenutku odabira tipa proizvoda (programski kod 5.19). Metoda je, naravno, dio klijentske aplikacije, jer je direktno povezana s korisnikovim akcijama na sučelju.

```
onSelectedPanel(productTypeId: number) {
  if (this.isAdmin) {
    this.productService
      .getProductsByType(productTypeId)
      .subscribe((response) => {
        this.products = response;
      });
  }
}
```

Programski kod 5.19. *Metoda koja se okida nakon odabira tipa proizvoda*

Nakon unosa svih željeni proizvoda, tvrtka može kreirati novu kolekciju. Model kolekcije u bazi dan je u programskom kodu 5.20. Osim osnovnih atributa poput imena, cijene, slike i dr., model sadrži i informacije o proizvodima koji se nalaze u njoj, te o svim narudžbama u kojima se kolekcija nalazi i njenim kupcima.

Prilikom svake interakcije s kolekcijama, potrebno je napisati servis na klijentskoj strani koji će definirati sve rute na koje idu API pozivi. Krerian je *bunchService.ts* gdje su opisane sve osnovne operacije nad kolekcijama (unos, dohvaćanje, uređivanje i brisanje), odnosno sve API rute (programski kod 5.21.)

```
namespace MuchBunch.EF.Database.Models
{
    public class Bunch
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string ImageUrl { get; set; }
        public float Price { get; set; }
        public int ThemeId { get; set; }
        public Theme Theme { get; set; }
        public int? CompanyId { get; set; }
        public User? Company { get; set; }
        public ICollection<Product>? Products { get; set; }
        public ICollection<User>? Buyers { get; set; }
        public ICollection<Order>? Orders { get; set; }
    }
}
```

Programski kod 5.20. *Model kolekcije u bazi*

```
@Injectable({
  providedIn: 'root',
})
export class BunchService {
  private serviceBaseUrl = '';

  constructor(
    public http: HttpClient,
    @Inject('API_BASE_URL') public baseUrl: string,
    public router: Router
  ) {
    this.serviceBaseUrl = `${this.baseUrl}/api/bunch`;
  }

  insertBunch(bunch: BunchBM): Observable<BunchBM> {
    return this.http.post<BunchBM>(`${this.serviceBaseUrl}`, bunch);
  }

  deleteBunch(bunchId: number): Observable<void> {
    return this.http.delete<void>(`${this.serviceBaseUrl}/${bunchId}`);
  }

  editBunch(bunch: BunchDTO): Observable<BunchDTO> {
    return this.http.post<BunchDTO>(`${this.serviceBaseUrl}/edit`, bunch);
  }
```

Programski kod 5.21. *Isječak servisa za kolekcije*

Prilikom jedne od operacije nad kolekcijama, primjerice unosa, stvorena je logika na poslužiteljskoj strani koja govori što napraviti prilikom poziva te API rute. Programski kod 5.22. daje objašnjenje toga što se događa kada podaci za unos kolekcije dođu do poslužiteljske strane. BM model je mapiran u model domene, nakon čega se dohvaćaju izabrani proizvodi preko Id atributa i povezuju s novim objektom. Naposlijetku, model se ubacuje u bazu i promjene se spremaju.

```
[HttpPost]
public async Task<IActionResult> InsertBunch([FromBody] InsertBunchBM model)
{
        var result = await validationService.ValidateInsertBunch(model);

        if (!result.IsValid)
        {
            return BadRequest(result);
        }

        bunchService.InsertBunch(model);

        return Ok();
}
```

Programski kod 5.22. *Isječak kontrolera za kolekcije*

Tijekom ažuriranja kolekcije slijed operacija je isti, jedina razlika je u korištenju DTO modela umjesto BM. Glavni razlog je što DTO model posjeduje identifikacijski ključ kolekcije, kako bi se znalo koju kolekciju se uređuje. Prilikom ažuriranja prvo je potrebno poništiti trenutno povezane podatke, te stvoriti nove veze i ažurirati podatak novim zapisom (programski kod 5.23.)

```
public void EditBunch(EditBunchBM model)
{
    var bunch = dbContext.Bunches.Include(b => b.Products).First(b => b.Id ==
model.Id);

    var products = dbContext.Products.Where(p =>
model.ProductIds.Contains(p.Id)).ToList();

    bunch.Products.Clear();
    bunch.Name = model.Name;
    bunch.Price = model.Price;
    bunch.ImageUrl = model.ImageUrl;
    bunch.ThemeId = model.ThemeId;
    bunch.CompanyId = model.CompanyId;
    bunch.Products = products;

    dbContext.Bunches.Update(bunch);
    dbContext.SaveChanges();
}
```

Programski kod 5.23. *Uređivanje kolekcije u servisu poslužiteljske aplikacije*

U slučaju poziva API rute za brisanje kolekcije, provjerava se ima li trenutni prijavljeni korisnik prava za brisanje tog zapisa. Pošto se slučaj brisanja kolekcije nekoj drugoj tvrtki ne bi smio dogoditi, programski kod 5.24. prikazuje metodu kontrolera koja objašnjava način dohvaćanja identifikacijskog ključa korisnika iz tokena u zaglavlju zahtjeva.

Naposlijetku, pruženo je i dohvaćanje pojedinačne kolekcije te popisa. U programskom kodu 5.25. je prikazan način kako se dohvaćaju relacijski podaci kolekcije. Objekt sa svim dohvaćenim

podacima se mapira u DTO i šalje natrag na kontroler, koji zatim prosljeđuje podatke klijentskoj aplikaciji.

```
public void DeleteBunch(int id, int userId)
{
        var bunch = dbContext.Bunches.Find(id);
        var isAdmin = dbContext.Users
            .Include(u => u.Role)
            .Where(u => u.Role.Name == Enums.Role.Admin)
            .Any(u => u.Id == userId);

        if (bunch != null && (isAdmin || bunch.CompanyId == userId))
        {
            dbContext.Remove(bunch);
            dbContext.SaveChanges();
        }
}
```

Programski kod 5.24. *Brisanje kolekcije*

```
public BunchDTO GetBunchById(int id)
{
    var bunch = dbContext.Bunches
          .Include(b => b.Company)
          .Include(b => b.Theme)
          .Include(b => b.Products)
              .ThenInclude(p => p.Type)
          .Include(b => b.Products)
              .ThenInclude(p => p.SubTypes)
          .FirstOrDefault(b => b.Id == id);

    return mapper.Map<BunchDTO>(bunch);
}
```

Programski kod 5.25. *Dohvaćanje kolekcije*

5.5.1. *AutoMapper i FluentValidation*

Prilikom operacija unosa, brisanja i ostalih nad podacima, potrebno je spomenuti *AutoMapper* i *FluentValidation* biblioteke. U programskom kodu 5.19. prikazana je uobičajena logika na kontroler metodi. Prvo se dobiveni model dobiveni od strane klijenta validira, kako ne bi došlo do neželjenih nuspojava. Za svaki model koji se želi validirati se kreira vlastiti validator koji u svom konstruktoru drži pravila koja model mora sadržavati, ako želi biti proslijeđen servisnom sloju. U svjetlu kreiranja nove kolekcije, u tom validatoru se definiraju poruke koje će predstavljati povratnu informaciju za klijenta u slučaju da je zahtjev odbijen ili neuspješno obavljen (programski kod 4.26.)

```
public class InsertBunchValidator : AbstractValidator<InsertBunchBM>
{
(..)

    public InsertBunchValidator(MBDBContext dbContext)
    {

(..)
        RuleFor(x => x.ThemeId)
            .MustAsync(async (themeId, ct) =>
            {
                var exists = await dbContext.Themes.AnyAsync(t => t.Id ==
themeId, ct);
                return exists;
            }).WithMessage(InvalidThemeId);

(..)

}
```

Programski kod 5.26. *Validator unosa kolekcije*

Nakon validacije, *AutoMapper* služi za smanjenje dupliciranog koda tijekom mapiranja podataka iz jednog modela u drugi. Najčešći slučajevi korištenja mapiranja su mapiranje iz BM u model domene ili iz model domene u DTO. Sva pravila se određuju u klasama koje implementiraju roditelj klasu *Profile* (programski kod 5.21.), te nakon implementacije, u svom konstruktoru definiraju pravila koja vrijednost će se dodijeliti na koji atribut. Istoimeni atributi se automatski prepisuju, stoga se samo odstupanja od konvencije trebaju ručno postaviti. Klase koje služe za definiranje pravila mapiranja se moraju nadodati u *Program.cs* datoteku.

```
public class MappingProfile : Profile
{
    public MappingProfile()
    {
        (..)

        CreateMap<ProductType, ProductTypeDTO>();
        CreateMap<ProductSubType, ProductTypeDTO>();
        CreateMap<Product, ProductDTO>();
        CreateMap<Role, RoleDTO>();
        CreateMap<User, UserDTO>()
            .ForMember(dest => dest.Role, opt => opt.MapFrom(src =>
src.Role.Name));
        CreateMap<Bunch, BunchDTO>();
        CreateMap<Theme, ThemeDTO>();
        CreateMap<Theme, ThemeSimpleDTO>();
        CreateMap<Order, OrderDTO>();
        CreateMap<InsertBunchBM, Bunch>()
            .ForMember(dest => dest.Products, opt => opt.Ignore());

    }
}
```

Programski kod 5.27. *Klasa MappingProfile za kreiranje mapiranja*

5.6. Kreiranje nove teme kolekcije te slanje elektroničke pošte

Kada administrator odluči da je vrijeme za novu tematsku kolekciju, prvo mora unijeti novu temu u aplikaciju. Temu prvo unosi u šifrarnik kako bi se prepoznalo njeno postojanje u aplikaciji, no ostaje neaktivna. Zatim, administrator pokreće proces aktiviranja nove teme, popunjavajući formu danu u programskom kodu 4.28. Administrator ispunjava specifikacije novih kolekcija koje će nositi ime nove teme, te time kreira obavijest koja će se proslijediti svim tvrtkama. Sve rute vezane uz kreiranje nove teme su omogućene samo administratoru.

```
initForm(): void {
    this.formGroup = this.fb.group({
      themeId: [''],
      productsQuantity: [''],
      bgQuantity: [''],
      accessoriesQuantity: [''],
      booksQuantity: [''],
      miniaturesQuantity: [''],
      price: [''],
    });
}
```

Programski kod 5.28. *Inicijalizacija forme za aktivaciju nove teme*

5.6.1. *MailKit* i slanje elektroničke pošte

Kako bi tvrtka dobila obavijest o postojanju nove teme, odlučeno je korištenje slanja elektroničke pošte. U tu svrhu koristi se biblioteka *MailKit*. To je NuGet paket instaliran za kreiranje SMTP (*Simple Mail Transfer Protocol*) klijenta, povezivanje na SMTP poslužitelja i proces slanja elektroničke pošte (programski kod 5.29.) Tijelo pošte može biti napisan u HTML-u te se mogu dizajnirati i personalizirati sadržaj poruke.

```
private static void SendMail(Email emailInfo, SmtpConfig smtpConfig)
{
            using var smtp = new SmtpClient();
            smtp.Connect(smtpConfig.Host, smtpConfig.Port,
SecureSocketOptions.StartTls);
            smtp.Authenticate(smtpConfig.Username, smtpConfig.Password);

            var email = CreateEmail(emailInfo);

            smtp.Send(email);
            smtp.Disconnect(true);
            Console.WriteLine(DateTime.UtcNow.ToString());
}
```

Programski kod 5.29. *Slanje elektroničke pošte*

Prema zahtjevu, potrebno je poslati elektroničku poštu tvrtkama u dva trenutka: kada administrator kreira novu temu te kada ta tema bude aktualna (po zahtjevu mjesec dana). Prva obavijest govori

tvrtkama da će se aktivirati nova tema za mjesec dana, te da voditelji tvrtke trebaju stvoriti kolekcije na temu u tom roku. Također, dane su im specifikacije kolekcije kao smjernice. Iz tog razloga, kreirana su dva različita predloška HTML dokumenta, od kojih svaki sadrži različite informacije (programski kod 5.30.) U svakom predlošku postoje rezervirana mjesta za buduće konkretne podatke, poput imena teme, cijene, proizvodi i sl. Tijekom procesa slanja pošte, prvo se učitava HTML predložak kao *string* tip podatka, te se preko definiranih ključeva zamjenjuju rezervirana mjesta s pravim podacima dobivenih preko API poziva od klijentske strane. Nakon dovršetka mijenjanja tijela pošte, model se predaje metodi za slanje. Elektronička pošta se šalje, te se SMTP klijent oslobađa.

```
<body>
    <div class="container">
        <div class="header">
            <h1>Welcome the upcoming MuchBunch Theme!</h1>

        </div>
        <div class="content">
            <p>Hello,</p>
            <br><br>
            <p>Next month, we are rolling out a new MuchBunch Theme: {{ theme
name }}!</p>
            <br>
            <p>Here are themed MuchBunch requirements: </p>
            <br>
            <p>There should be {{ products quantity }} products in the Bunch.
Product types are the following: {{ board game count }} Board game(s), {{
accessories quantity }} Accessories, {{ books quantity }} Books and {{
miniatures quantity }} Miniatures.</p>
            <p>The price of this MuchBunch should be {{ price }} Euros.</p>
            <br>
            <p>Now, let's go and create that themed MuchBunch!</p>
            <br><br>
            <p>Kind regards,</p><br>
            <p>MuchBunch team.</p>
        </div>
    </div>
</body>
```

Programski kod 5.30. *Predložak elektroničke pošte*

No, kako bi se znalo u kojem trenutku poslati drugu poštu, kreira se pozadinski posao koji se brine za navedeno. Proces slanja druge pošte je identičan, osim korištenja drugog HTML predloška, no okida ju pozadinski posao, ne administrator ili korisnik. Programski kod 5.31. opisuje nastajanje novog pozadinskog posla, što će taj posao obavljati i što se očekuje. Kako se posao odvija neovisno o aplikaciji (engl. *async*), potrebno je definirati kontekst u kojem će se obavljati. Omogućuje se pristup svim servisima koji će se koristiti u procesu slanja pošte te pristup bazi podataka. Nakon postavljanja okoline pozadinskog posla, piše se logika slanja nove elektroničke pošte, no i

aktiviranje nove teme. Kako bi se izbjegao slučaj da krajnji kupac vidi sve nadolazeće kolekcije, takva tema je postavljena neaktivnom u bazi, te klijentska strana prikazuje samo aktivne teme i njene kolekcije. Pozadinski posao postavlja temu aktivnom, što mijenja prikazane teme i kolekcije krajnjem korisniku.

```
private async Task SetupLiveTheme(EmailText model, int themeId)
{
    DateTime scheduledTime = DateTime.UtcNow.Date.AddMonths(1);

    TimeSpan delay = scheduledTime - DateTime.UtcNow;

    if (delay < TimeSpan.Zero)
    {
        throw new Exception("Delay is negative");
    }

    // Setup worker environment
    IServiceCollection services = new ServiceCollection();

    services.AddDbContext<MBDBContext>();
    services.AddAutoMapper(typeof(MappingProfile));
    services.AddScoped<NotificationService>();
    services.AddScoped<IdentityService>();
    services.AddScoped<UserService>();
    services.AddScoped<ThemeService>();
    services.AddSingleton(_configuration);

    IServiceProvider serviceProvider = services.BuildServiceProvider();

    var timer = new Timer((state) =>
    {
        using (var scope = serviceProvider.CreateScope())
        {
            var notificationService =
scope.ServiceProvider.GetRequiredService<NotificationService>();
            var themeService =
scope.ServiceProvider.GetRequiredService<ThemeService>();

            themeService.SetThemeAsActive(themeId);
            notificationService.SendLiveThemeEmail(model);
        }
        ((Timer)state).Dispose();
    });
    timer.Change(delay, TimeSpan.Zero);
}
```

Programski kod 5.31. *Postavljanje pozadinskog posla*

5.7. Kreiranje narudžbe i pojam pretplate

5.7.1. Kreiranje narudžbe

Web aplikacija ima funkcionalnosti dodavanja, uređivanja i dohvaćanja proizvoda i kolekcija, kao i slanje elektroničke pošte, no potrebno je implementirati narudžbe. U bazi podataka, korisnik ima M-N vezu s tablicom kolekcija, što zapravo predstavlja narudžbu. Narudžba je nova tablica koja, uz strane ključeve, sadrži i podatak o vremenu stvaranja.

U trenutku kupnje kolekcije na klijentskoj strani se okida *onBuyBunch* metoda dana u programskom kodu 5.32. Skupljaju se podaci o narudžbi, poput koja kolekcija je kupljena i od strane koga, te se poziva metoda servisa za kupnju. Važno je napomenuti kako je u tome trenutku potrebno i ažurirati lokalnu instancu korisnika u klijentskoj aplikaciji i podatke u lokalnoj pohrani preglednika kako bi se uključila nova narudžba.

```
onBuyBunch(bunch: EditBunchBM) {
    let order: OrderBM = new OrderBM(+this.userId, bunch.id);

    this.bunchService.placeOrder(order).subscribe((_) => {
      this.shouldDisableBuyButton = true;
      this.message.success(
        `MuchBunch ${bunch.name} was successfully ordered!.`
      );
      this.authService.getUserOrders(this.userId).subscribe((response) => {
        this.authService.user.value.orders = response;
        localStorage.setItem(
          'tokenInfo',
          JSON.stringify(this.authService.user.value)
        );
      });
    });
  }
```

Programski kod 5.32. *Metoda za kupnju kolekcije*

Nadalje, potrebno je kupcu isporučiti račun kupnje. Korišteno je slanje računa preko elektroničke pošte, kao u poglavlju 4.8. Proces je identičan; dohvaća se HTML predložak računa, zamjenjuju se podaci s podacima narudžbe te se nakon dobivanja gotovog tijela pošte, šalje pošta korisniku.

5.7.2. Pojam pretplate

Kako proces naručivanja funkcionira na način „tko prvi - njegovo“, odlučeno je uvesti pojam pretplate na platformu. Pretplata je riješena preko slanja elektroničke pošte – dodaje se i krajnji korisnik na listu primatelja pošte koja govori da je nova tema aktualna. Korisnik se navigacijom na profil u svakom trenutku može maknuti s pretplate.

5.8. Filtriranje i pretraživanje kolekcija

Kako bi se korisniku olakšalo korisničko iskustvo i skratilo vrijeme traženja određene kolekcije, klijentska aplikacija implementira filtriranje kolekcija na temelju tipova proizvoda unutar. Programski kod 5.33. se okida na trenutak kada je korisnik odabrao koji tip proizvoda želi vidjeti, te se dohvaćaju sve kolekcije koje postoje (ali nisu aktivne i nisu tematske) i filtriraju se kolekcije koje sadržavaju taj tip proizvoda. Potom, dohvaćaju se i svi podtipovi u slučaju da korisnik dalje filtrira svoj izbor.

```
onSelectedType(selectedType: number) {
    if (selectedType != null) {
      this.themeService
        .getBunchesByThemeId(this.themes.find((e) => e.isActive ===
false).id)
        .subscribe((response) => {
          this.bunches = response.bunches;

          this.productService
            .getSubtypesByParentId(selectedType)
            .subscribe((response) => {
              this.subtypes = response;
            });

          this.bunches = this.bunches.filter(
            (e) => e.products.find((a) => +a.type.id == selectedType) != null
          );
        });
    }
  }
```

Programski kod 5.33. *Filtriranje kolekcije na temelju tipova proizvoda unutar*

Nadalje, u slučaju da korisnik zna ime kolekcije koju traži, pruženo je pretraživanje po imenu kolekcije. Prilikom procesa traženja, dohvaćaju se sve kolekcije iz baze podataka te se prikazuju korisniku na sučelju u obliku *dropdown* elementa (programski kod 5.34.)

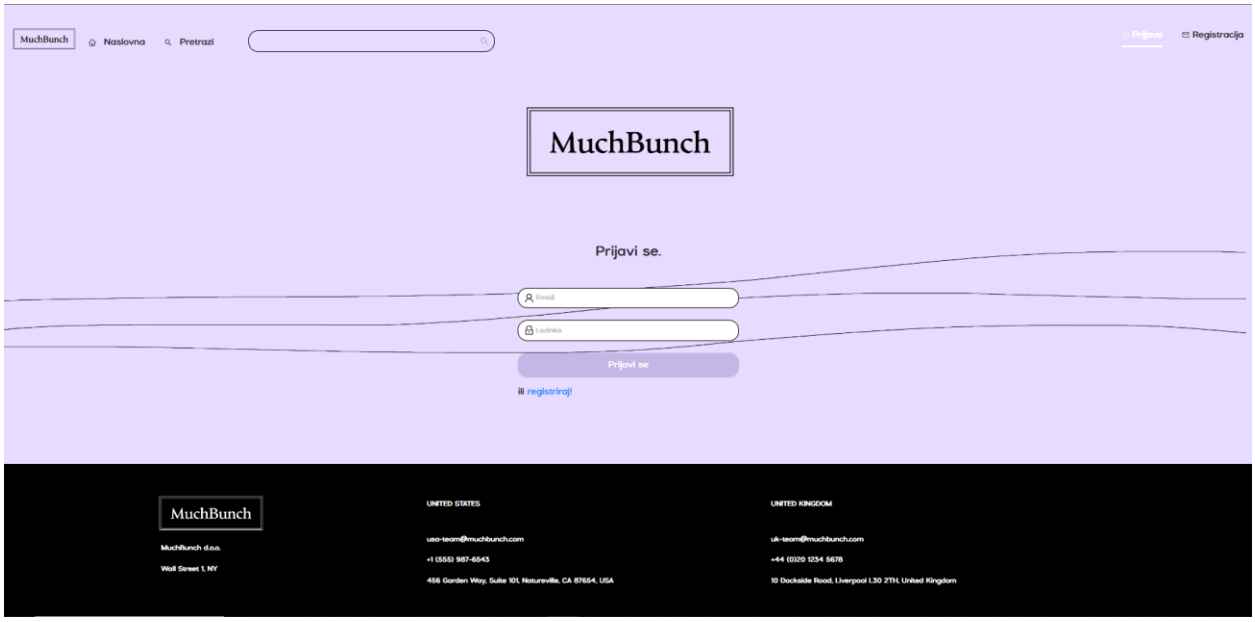
```
<nz-select
        nzShowSearch
        nzAllowClear
        [(ngModel)]="selectedValue"
        (ngModelChange)="onSelectedBunch(selectedValue)"
        [nzSuffixIcon]="templateIcon"
      >
(..)
        <nz-option
          *ngFor="let bunch of bunches"
          [nzLabel]="bunch.name"
          [nzValue]="bunch"
        ></nz-option>
</nz-select>
```

Programski kod 5.34. *HTML predložak za tražilicu kolekcija*

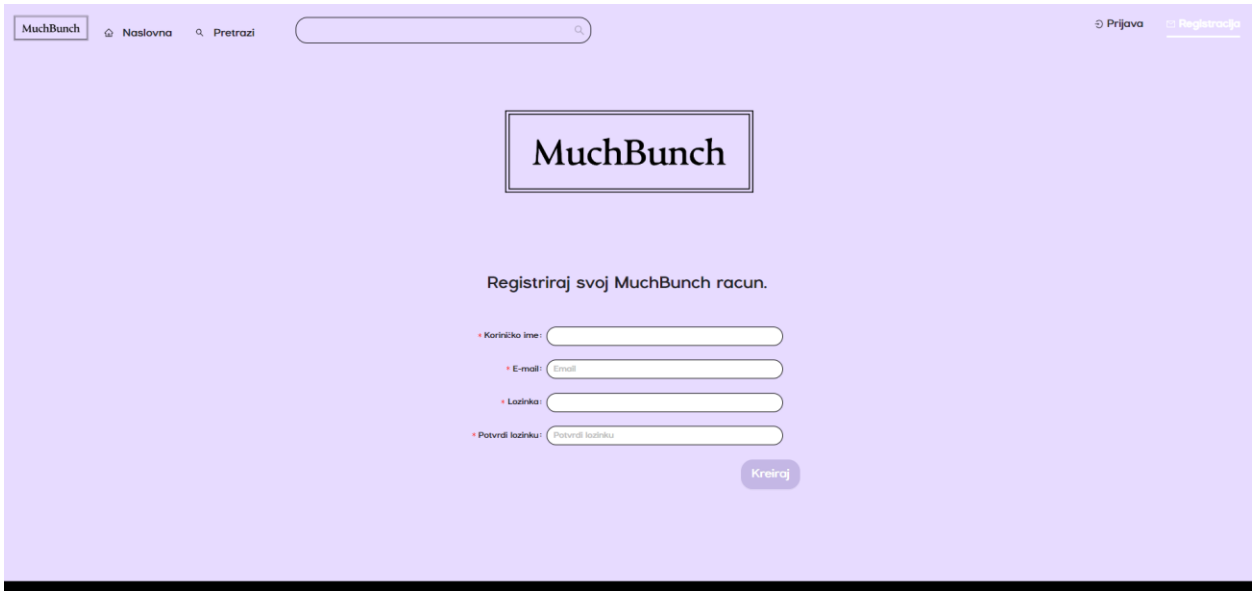
6. KORIŠTENJE WEB APLIKACIJE I ANALIZA PROGRAMSKOG RJEŠENJA

6.1. Korisničko iskustvo krajnjeg korisnika

Korisnik dolaskom na platformu se prijavljuje ili registrira novim računom (slike 6.1. i 6.2.)



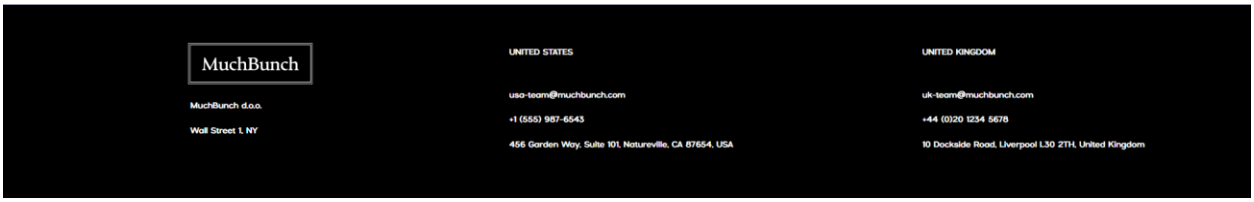
Slika 6.1. *Prijava korisnika*



Slika 6.2. *Registracija korisnika*

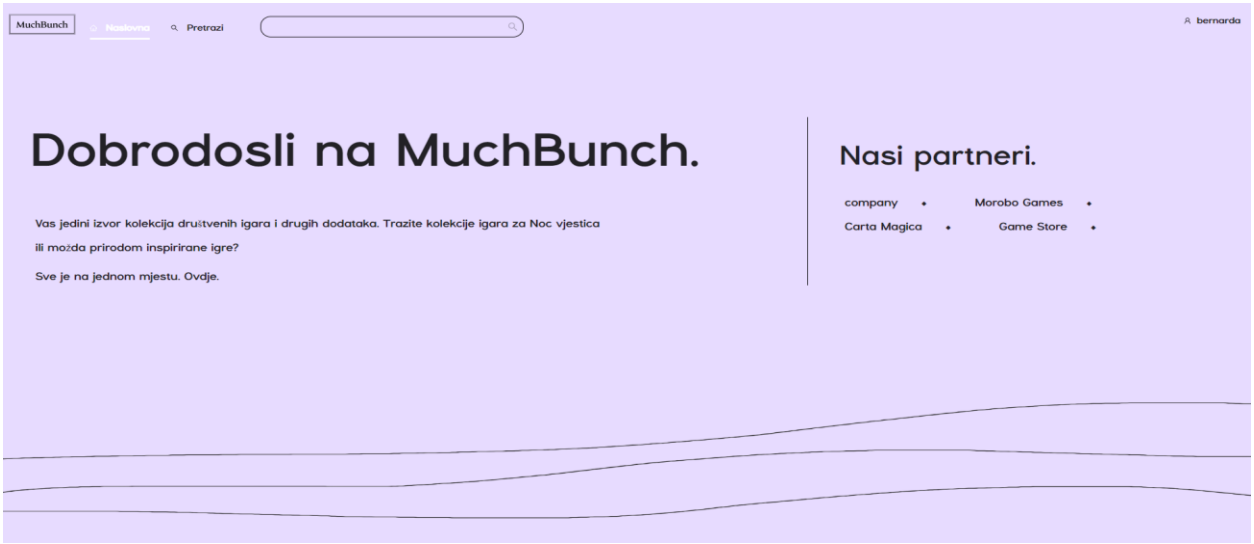
Važno je napomenuti kako korisnik kao neprijavljen može pretraživati kolekcije, no ne i kupovati. Nadalje, glavni elementi koji se nalaze u aplikaciji su zaglavlje (slika 6.3.) i poglavlje (slika 6.4.)

Slika 5.3. *Zaglavlje aplikacije*

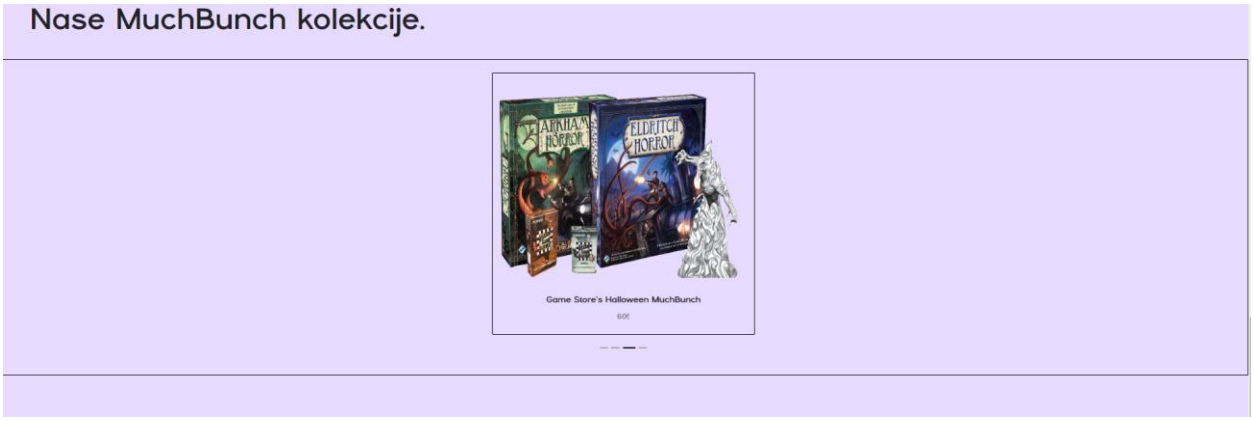


Slika 6.4. *Poglavlje aplikacije*

Akcije u zaglavlju su prečaci koji vode na određene module aplikacije, poput naslovna stranica, stranica za pretraživanje proizvoda, te stranica korisničkog profila. Naslovna stranica sadrži općenite informacije o platformi, poput imena, manjeg objašnjenja, popis partnera i izdvojene kolekcije (slike 6.5. i 6.6.)



Slika 6.5. *Naslovna stranica*



Slika 6.6. *Naslovna stranica - nastavak*

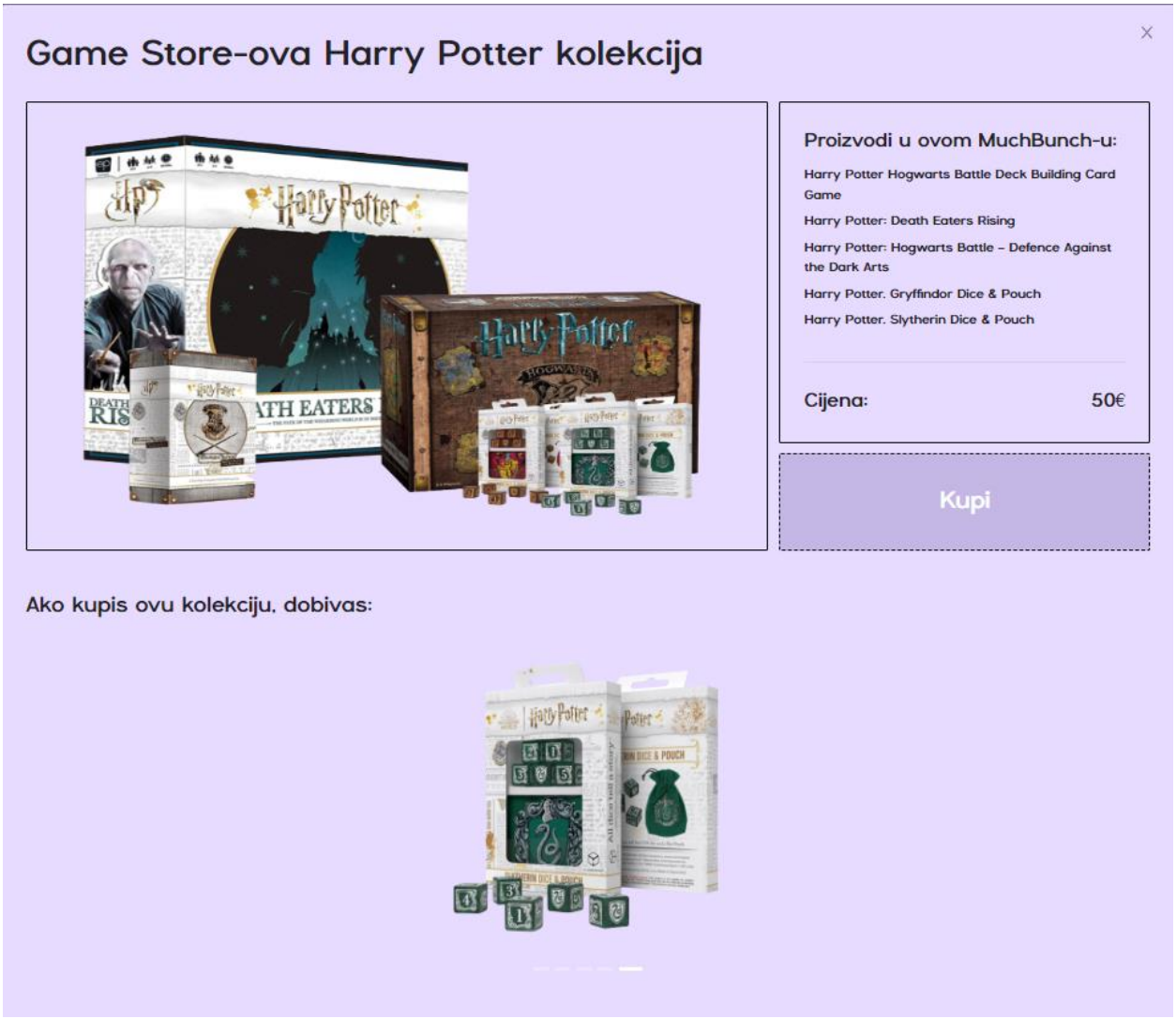
Navigacijom na stranicu za pretraživanje, korisniku su ponuđene dvije opcije kolekcija: tematske i netematske, uz to da netematske može filtrirati po želji. Filtriranje funkcionira na način odabira tipa proizvoda unutar kolekcije, te po odabiru podtipa proizvoda (slika 6.7.)

Slika 6.7. *Pregled netematskih kolekcija*

Ako korisnik filtrira proizvode na temelju društvenih igara podtipa ekonomske igre, filtriraju se sve kolekcije koje sadrže takvu igru, primjerice igre Monopoly (slika 6.8.)

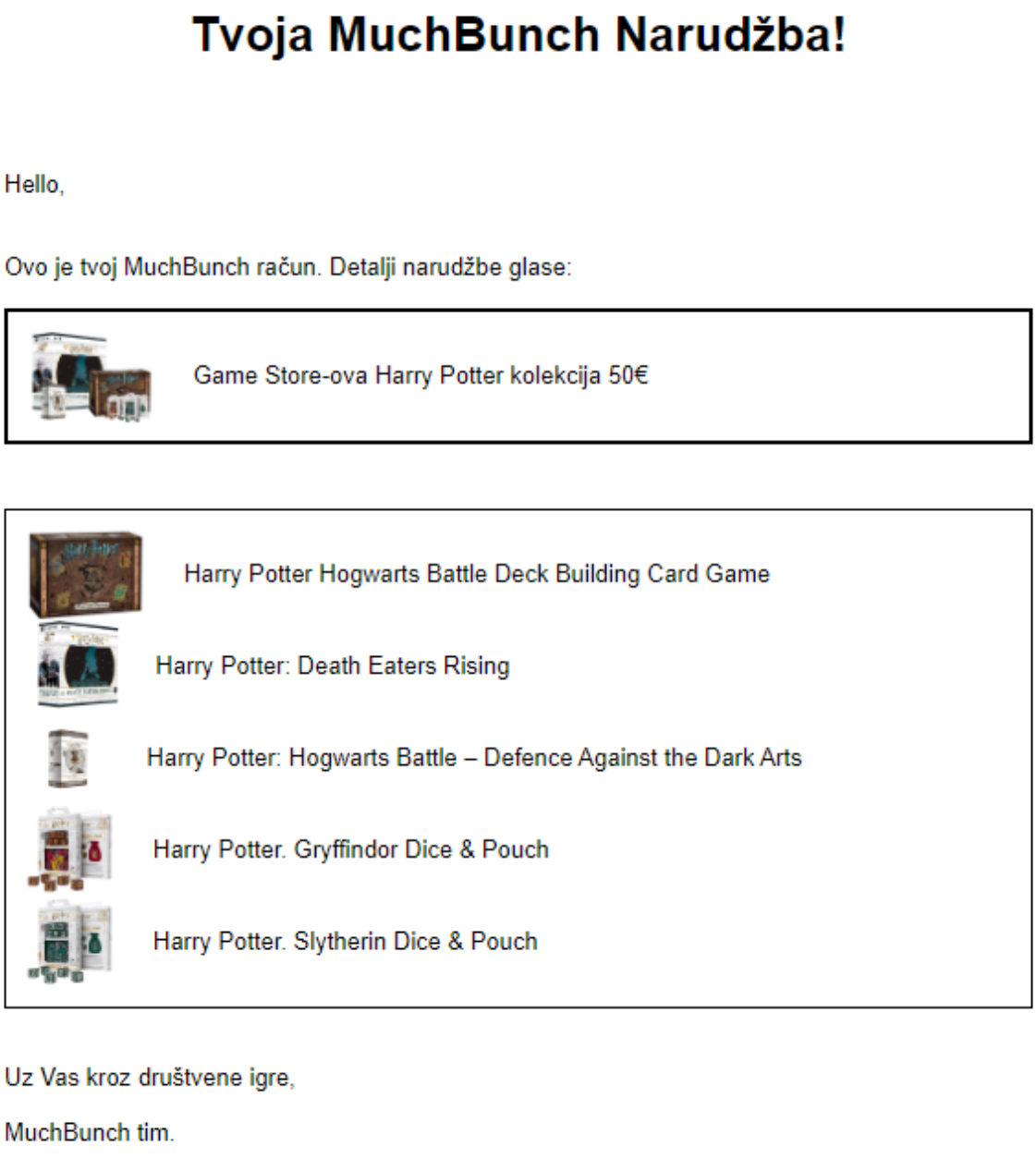
Slika 6.8. *Filtriranje kolekcija*

Kako bi korisnik pobliže pogledao proizvode u kolekciji, klikom na kolekciju otvara se modalni prozor u kojem se nalaze podaci kolekcije. Prikazane su slike kolekcije, kao i slike pojedinačnog proizvoda koje se izmjenjuju svakih par sekundi (slika 6.8.) Korisnik iz modala može naručiti kolekciju, no samo u slučaju da postoji dovoljno proizvoda na zalihi od kolekcije.



Slika 6.9. *Modal za pojedinosti kolekcije*

Korisnik dobiva elektroničku poštu s detaljima svoje narudžbe čim se kreira. Predložak elektroničke pošte nalazi se na slici 6.10.



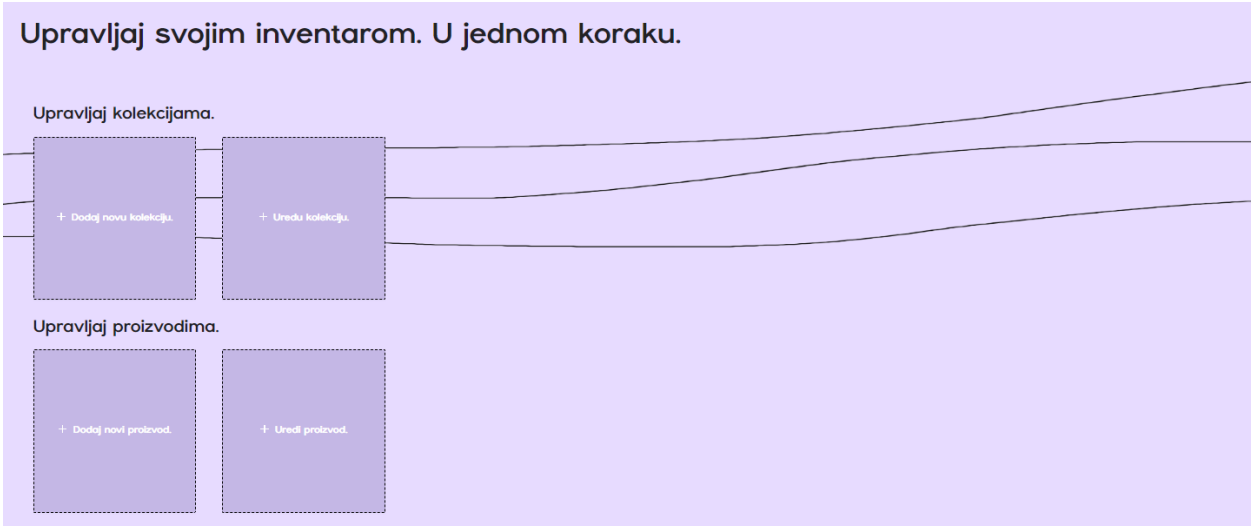
Slika 6.11. *Elektronička pošta računa*

Korisnik odlaskom na njegov profil vidi osnovne informacije o svom korisničkom računu, te prošle narudžbe. Također, postoji kutak za pretplatu gdje stoje pogodnosti pretplate na platformu i akcija za odlaženje s pretplate (slika 6.12.).

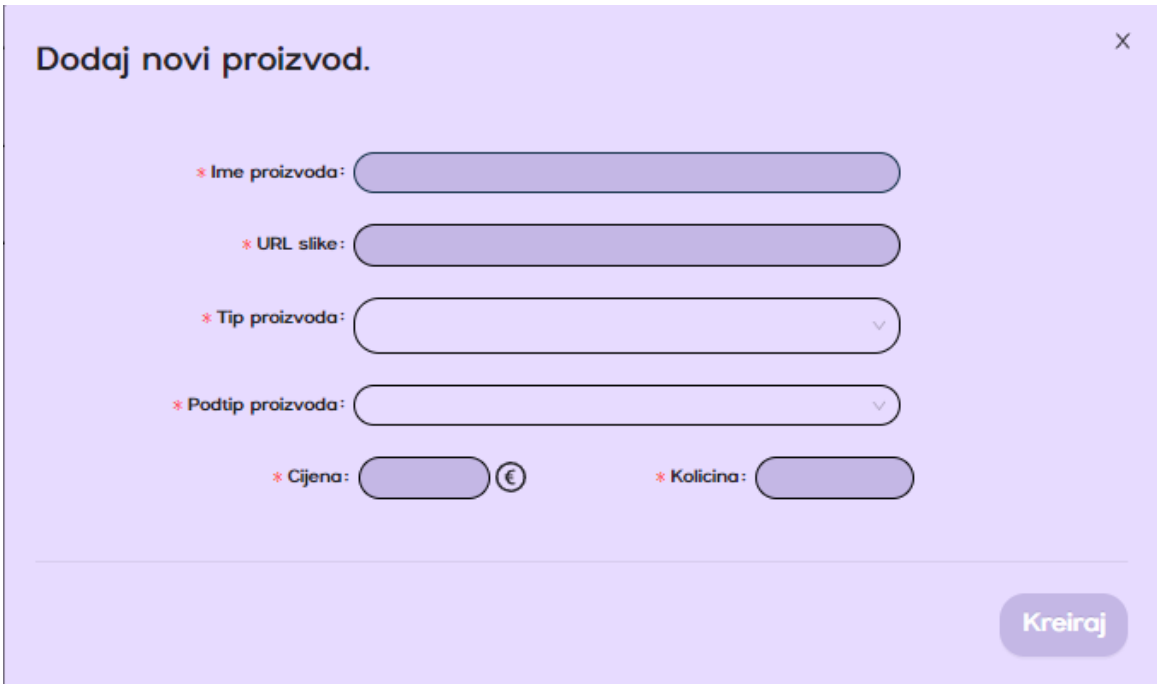
Slika 6.12. *Profil korisnika*

6.2. Korisničko iskustvo tvrtke

Tvrtka kao korisnik platforme posjeduje iste funkcionalnosti kao i krajnji korisnik, uz dodatak kreiranja proizvoda i kolekcija (slika 6.13.). Jedina razlika je što se tvrtka ne registrira na platformu, nego dobiva već spreman i napravljen račun od strane administratora platforme. Sva kreiranja ili uređivanja proizvoda ili kolekcija događaju se u modalnom prozoru (slika 6.14.), te se nakon akcije vraća povratna poruka korisniku o uspješnosti. Kod uređivanja proizvoda, otvara se isti modalni prozor, no popunjen već podacima o proizvodu. Proces je isti kod kreiranja i uređivanja kolekcije.

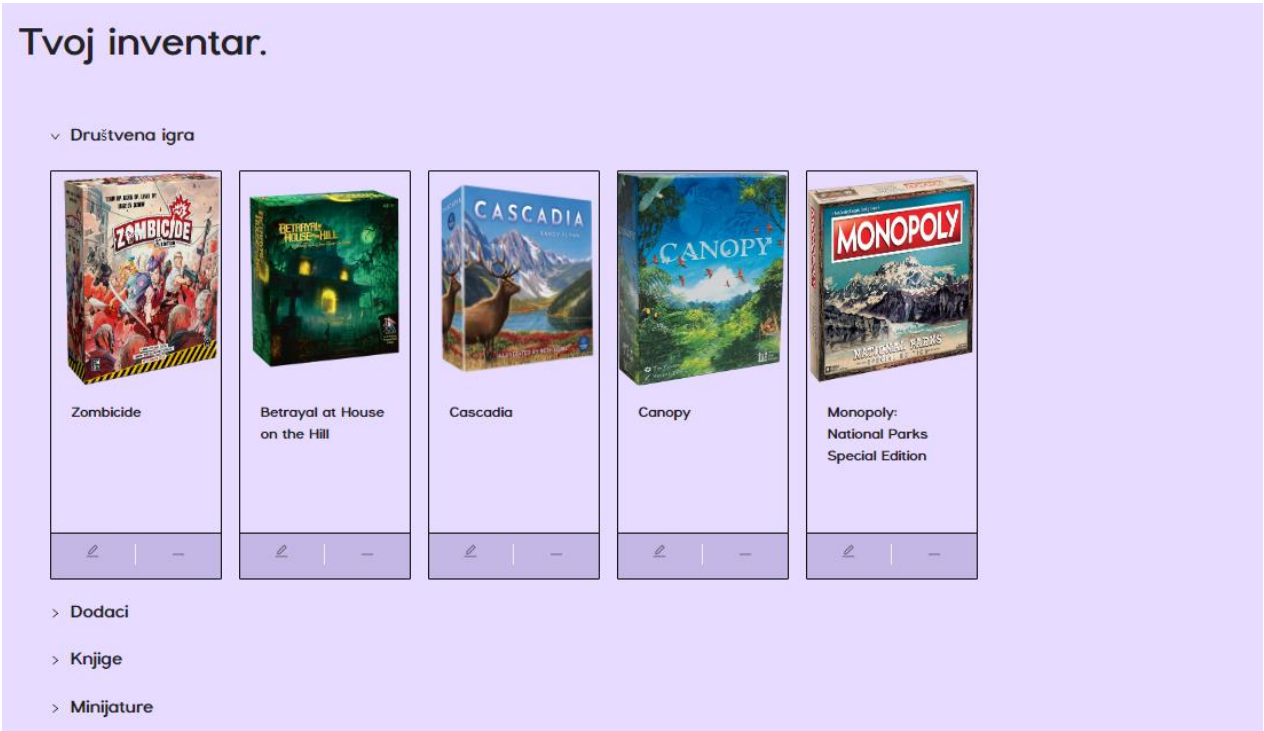


Slika 6.13. *Kutak za upravljanje proizvodima i kolekcijama*



Slika 6.14. *Modalni prozor za kreiranje novog proizvoda*

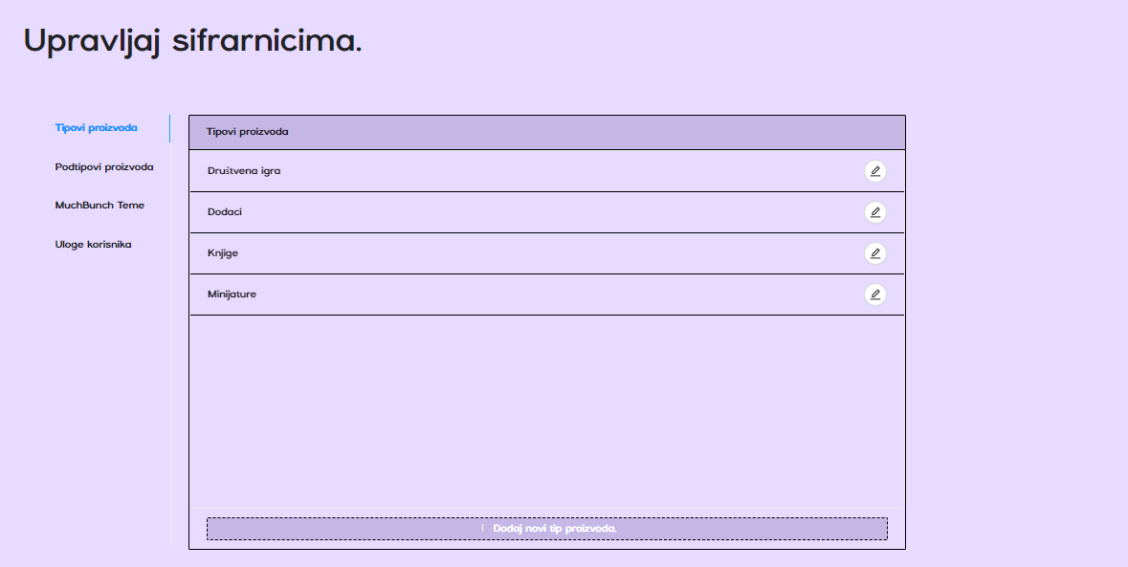
Nakon kreiranja proizvoda i kolekcija, tvrtka može provjeriti njihovo postojanje na stranicama inventara i popisa kolekcija. Od tamo, tvrtka također može urediti proizvod ili kolekciju, uz još funkcionalnost brisanja istog (slika 6.15.)



Slika 6.15. *Popis proizvoda tvrtke*

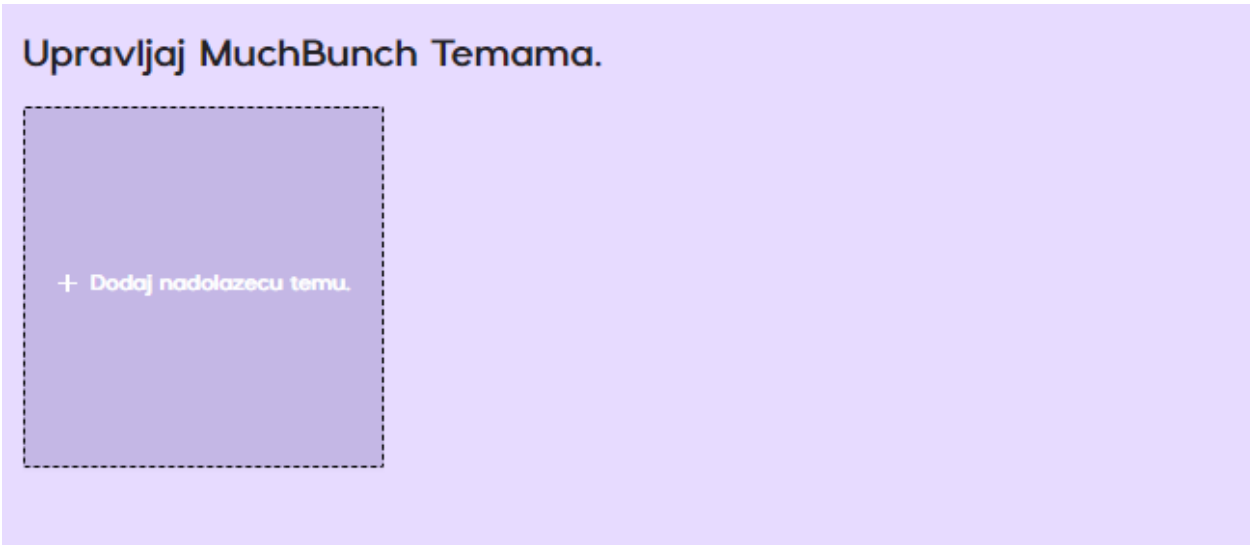
6.3. Administracijsko sučelje

Osim što administrator ima funkcionalnosti obje prethodno spomenute uloge, također može upravljati šifrarnicima na administracijskom sučelju (slika 6.16.) Upravljanje svim proizvodima i kolekcijama tvrtke, administrator prvo odabire tvrtku čijim proizvodima želi upravljati, te onda nastavlja s pristup zatraženom. Omogućen mu je unos novih tipova i podtipova proizvoda, nove teme kolekcije i uloga rola. Na istome mjestu je moguće i urediti ili obrisati neželjene podatke.



Slika 6.16. *Upravljanje šifrarnicima*

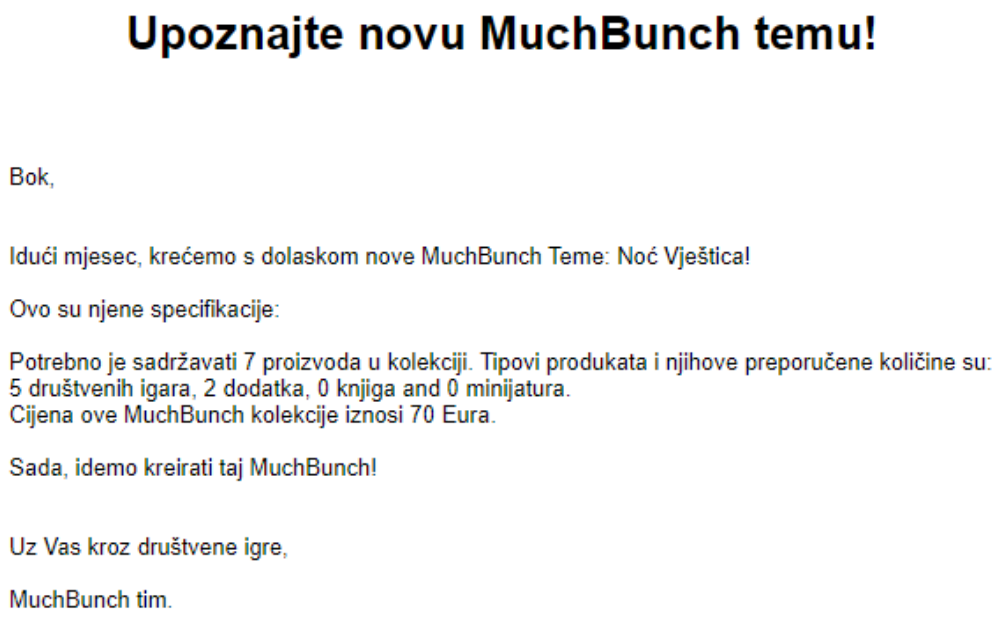
Zatim, u modulu administracije, administrator vidi posebni odjeljak za kreiranje nadolazeće MuchBunch teme (slika 6.17.) Klikom se otvara novi modalni prozor u kojem administrator priprema predložak elektroničke pošte koju će primiti tvrtke (slika 6.18.)



Slika 6.17. *Nadodan kutak za kreiranje nove teme*

Slika 6.18. *Modalni prozor za kreiranje predloška nadolazeće teme*

Naposlijetku, šalje se elektronička pošta tvrtkama, popunjena podacima od strane administratora, te se tvrtke obavještavaju o dolasku nove teme za mjesec dana (slika 6.19.)



Slika 6.19. *Elektronička pošta nadolazeće teme*

7. ZAKLJUČAK

Razvijanje bilo kakve web aplikacije je vrlo zapetljan i složen posao koji zahtijeva široki spektar vještina. Od odabira korisničkog sučelja te sve vezano uz vizualni aspekt aplikacije je posao koji zahtijeva vještine dizajnera, no i dobro poznavanje elemenata sučelja i njihovih interakcija. Bilo je potrebno kreirati ugodan i pojednostavljen tok rada platforme bez zbunjivanja korisnika.

Nadalje, posao implementiranja zahtjeva platforme zahtijeva vještine dviju strana, poslužiteljske i klijentske. Poslužiteljska vodi brigu o arhitekturi projekta, arhitekturi baze podataka te o cijeloj logici nad podacima. Koristeći *.NET Framework* aplikaciju kao poslužiteljsko rješenje, stvorena je sigurna aplikacija koja komunicira s Angular klijentskom aplikacijom. Krenuvši od skice baze podataka, koristi se pristup bazi podataka kroz kod (engl. *code-first approach*) koji omogućuje kreiranje entiteta baze kroz kod, kao i svih veza među entitetima. Biblioteka *Entity Framework* omogućuje spomenuti pristup te je standardna praksa uz korištenje *.NET Framework* okoline. Zatim, koristile su se biblioteke *AutoMapper* radi pojednostavljenja procesa kopiranja podataka iz jednog objekta u drugi, *MailKit* biblioteku zbog mogućnosti rada s električnom poštom putem SMTP-a i biblioteku *FluentValidation* kako bi se definirala pravila validacije podataka. Koristeći sve gore navedene biblioteke, gradila se poslužiteljska aplikacija koja je bila krajnja točka u cijelom radu platforme.

Klijentska strana hvata sve korisnikove radnje i prosljeđuje ih poslužiteljskoj strani na otprije opisan način. Način prijenosa podataka su definirani modelima podijeljenim na dva tipa: BM (*Binding model*) i DTO (*Data Transfer Object*) model. Obje strane kreiraju pravila gdje se događa prijenos podataka, odnosno na koje se API rute šalju pozivi. Klijentska strana kreira servise u kojima su spomenute definicije i koristi ih nadalje u pojedinačnim komponentama. Poslužiteljska strana kreira odgovore na API pozive u kontrolerima te u servisima opisuje što s dobivenim podacima. Time je dan osnovni slijed procesa koji omogućuju rad platforme nazvane MuchBunch.

Iako se proces rada na papiru čini jednostavan, izvedba je puno zamršenija i puna nuspojava biblioteka koje se međusobno suzbijaju te ih je takve potrebno dobro proučiti prije korištenja.

LITERATURA

[1] Kickstarter, https://www.kickstarter.com/ (stranica posjećena 29.06.2023.)

[2] Humble Bundle, https://www.humblebundle.com/ (stranica posjećena 29.06.2023.)

[3] Carta Magica, https://www.cartamagica.hr/ (stranica posjećena 29.06.2023.)

[4] G. Geetha, M. Mittal, K.M. Prasad, J. G. Ponsam, **Interpretation and Analysis of Angular Framework,** IEEE, 2022.

[5] What is Angular, https://angular.io/guide/what-is-angular (stranica posjećena 29.06.2023.)

[6] S. Seshadri, **Angular: Up & Running**, O'Reilly Media, 2018.

[7] G. Fink, I. Flatow, **Single Page Application Development,** Appress, 2014.

[8] T. Thai, H. Q. Lam, **.NET Framework Essentials,** third edition, O'Reilly Media, 2003.

[9] J. Wiley, **Professional ASP.NET MVC 4,** John Wiley & Sons Inc., 2012.

[10] C. Ireland, D. Bowers, M. Newton, K. Waugh, **Understanding Object-Relational Mapping: A Framework Based Approach**, Department of Computing, The Open University Milton Keynes, UK, 2009.

[11] S. Jablonski, I. Petrov, C. Meiler, U. Mayer, **Guide to Web Application and Platform Architectures,** Springer, 2004

[12] R. R. Singh, **Mastering Entity Framework,** Packt Publishing, 2015

[13] S. Ahmed, Q. Mahmood, **An authentication based scheme for applications using JSON web token,** IEEE, 2019

[14] A. Chaturvedi, **Comparison of Different Authentication Techniques and Steps to Implement Robust JWT Authentication,** IEEE, 2022

SAŽETAK

Web platforma za kupovanje kolekcija društvenih igara ima u cilju približiti korisnicima svijet društvenih igara te pomoći pri odabiru istih. Omogućeno je pregledavanje, filtriranje i kupnja kolekcija, čiji račun dolazi korisniku u obliku elektroničke pošte. Također, omogućen je pristup i tvrtkama koje će slagati kolekcije od vlastitih proizvoda koje unose u aplikaciju. Administrator zadaje tvrtkama nadolazeće teme i njene specifikacije kako bi tvrtka mogla kreirati ih. Osim tematskih kolekcija, tvrtka ima mogućnost kreirati vlastite netematske kolekcije po željenim specifikacijama te tako pružiti zabavan sadržaj krajnjim korisnicima. Ugrađen je i sustav pretplate na obavijesti o novim kolekcijama, kako bi se izbjegao slučaj nestanka zaliha proizvoda kolekcija. Platforma je građena u Angular i .NET okruženju, koristeći jezike TypeScript i C#.

Ključne riječi: Angular, .NET Framework, Pristup bazi kodom, Slanje elektroničke pošte, Pretplata, Kolekcije društvenih igara, TypeScript, C#

ABSTRACT

A platform for purchasing board game collections

The web platform for buying board game collections aims to bring the world of board games closer to the end user and help him choose. It is possible to browse, filter and purchase collections, whose bill is sent to the user in the form of an electronic mail. Also, access is provided to companies that will put together collections consisted of their own products entered in the application. The administrator assigns companies upcoming topics and their specifications so that the company can create them. In addition to themed collections, the company has the possibility to create its own non-thematic collections according to the desired specifications and thus provide entertaining content to end users. A subscription system for notifications about new collections is also built in, to avoid product collections being out of stock. The platform is built in Angular and .NET environments, using TypeScript and C# languages.

Keywords: Angular, .NET Framework, Codebase Access, Emailing, Subscription, Board Game Collections, TypeScript, C#

PRILOZI

Prilog 1. Diplomski rad u docx formatu

Prilog 2. Diplomski rad u pdf formatu

Prilog 3. Programski kod web aplikacije