Izrada full stack web aplikacije "Game Developer Forum" primjenom Spring Boot i ReactJS

Rumštajn, Mauricio

Undergraduate thesis / Završni rad

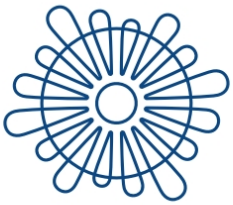
2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zadar / Sveučilište u Zadru**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:162:118513

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-16**



Sveučilište u Zadru
Universitas Studiorum
Jadertina | 1396 | 2002 |

Repository / Repozitorij:

University of Zadar Institutional Repository

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

Sveučilište u Zadru

Stručni prijediplomski studij Informacijske tehnologije

Mauricio Rumštajn

Izrada full stack web aplikacije “Game Developer Forum” primjenom Spring Boot i ReactJS

Završni rad

Zadar, 2023.

Sveučilište u Zadru

Stručni prijediplomski studij Informacijske tehnologije

IZRADA FULL-STACK WEB APLIKACIJE “GAME DEVELOPER FORUM” PRIMJENOM SPRING BOOT I REACTJS

Završni rad

Student:
Mauricio Rumštajn

Mentor:
doc. dr. sc. Ante Panjkota

Zadar, 2023.

Izjava o akademskoj čestitosti

Ja, **Mauricio Rumštajn,** ovime izjavljujem da je moj **završni** rad pod naslovom **Izrada full-stack web aplikacije „Game Developer Forum“ primjenom Spring Boot i ReactJS** rezultat mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na izvore i radove navedene u bilješkama i popisu literature. Ni jedan dio mojega rada nije napisan na nedopušten način, odnosno nije prepisan iz necitiranih radova i ne krši bilo čija autorska prava.

Izjavljujem da ni jedan dio ovoga rada nije iskorišten u kojem drugom radu pri bilo kojoj drugoj visokoškolskoj, znanstvenoj, obrazovnoj ili inoj ustanovi.

Sadržaj mojega rada u potpunosti odgovara sadržaju obranjenoga i nakon obrane uređenoga rada.

Zadar, 5. rujna 2023.

SADRŽAJ:

1 UVOD 1

2 TEORIJSKE OSNOVE 2

2.1 RAZVOJ WEB APLIKACIJA PRIMJENOM PARADIGME „POTPUNOG STOGA“ (ENG. FULL STACK DEVELOPMENT) 2
2.2 BACKEND 2
2.3 BAZA PODATAKA 3
2.4 FRONTEND 5

3 OPIS PONUĐENOG RJEŠENJA 7

3.1 ARHITEKTURA APLIKACIJE 7
3.2 KONCEPTUALNI OPIS RJEŠENJA 9
3.3 KORISNIČKE ULOGE I PRAVA 10
3.4 MODEL BAZE PODATAKA 11
3.5 ADMINISTRACIJSKO I KORISNIČKO SUČELJE 13
3.6 SIGURNOSNI ASPEKTI RJEŠENJA 15

4 RAZRADA RAZVOJA APLIKACIJE 15

4.1 BAZA PODATAKA 15
4.1.1 TIP BAZE PODATAKA 15
4.1.2 UPRAVLJANJE VERZIJAMA MODELA BAZE PODATAKA 16
4.2 SERVERSKI DIO 18
4.2.1 PODATKOVNI SLOJ 18
4.2.2 SERVISNI SLOJ 20
4.2.3 WEB SLOJ 21
4.2.4 DEPENDENCY INJECTION OBRAZAC I IoC KONTEJNER 23
4.2.5 SIGURNOST 24
4.2.6 UPRAVLJANJE GREŠKAMA NA SERVERSKOJ STRANI 26
4.2.7 CHAT SERVER 28
4.2.8 DOCKER 30
4.3 IZRADA KORISNIČKOG SUČELJA (FRONTEND) 31
4.3.1 RAZVOJNI OKVIRI ZA KLIJENTSKI DIO APLIKACIJE GAME DEVELOPER FORUM 31
4.3.2 CSS RAZVOJNI OKVIR 32
4.3.3 POVEZIVANJE KLIJENTSKE I SERVERSKE STRANE 34
4.3.4 OMATANJE ODGOVORA 34
4.3.5 UPRAVLJANJE GREŠKAMA NA KLIJENTSKOJ STRANI 36
4.3.6 SIGURNOST NA KLIJENTSKOJ STRANI 36
4.3.7 NAVIGACIJA 37

5 UPRAVLJANJE VERZIJAMA PROJEKTA 38

6 RASPRAVA 39

6.1 SKALABILNOST 39
6.2 MOGUĆE NADOGRADNJE 40

7 ZAKLJUČAK 41

8 LITERATURA 42

9 POPIS SLIKA I TABLICA 48

SAŽETAK

Ovaj rad prikazuje izradu web aplikacije „Game Developer Forum“ putem razvojne paradigme punog stoga koristeći Spring Boot za serversku stranu i ReactJS za klijentsku stranu. Predstavljeno rješenje predstavlja platformu za sinkronu i asinkronu komunikaciju između stvaratelja video igara. Aplikativno rješenje donosi mogućnost asinkrone komunikacije između korisnika putem tzv. soba za raspravu te sinkronu komunikaciju u pravom vremenu putem chat stranice. Prednost Game Developer Forum aplikacije, u odnosu na ostala aplikativna rješenja koja rješavaju sličan problem, je ta da je aplikacija, unatoč tome što se može koristiti kao opći forum, prilagođena stvarateljima igara. Neka unaprjeđenja koja su odmah vidljiva s razvojne strane su: bolje upravljanje greškama na serverskoj i klijentskoj strani, veći stupanj razrađenosti korisničkog sučelja, optimiziranje operacija čitanja i pisanja, mogućnost produljena sesije nakon isteka i poboljšanje sigurnosnog aspekta aplikacije na način da nije izložena suvremenim prijetnjama.

Ključne riječi: Internet forum, full-stack, Spring Boot, ReactJS

POPIS KORIŠTENIH KRATICA

DBMS	Database Management System
SQL	Structured Query Language
JSON	JavaScript Object Notation
YML	YAML Ain't Markup Language, prethodno Yet Another Markup Language
XML	Extensible Markup Language
IO	Input/Output
JPA	Jakarta Persistance API, prethodno Java Persistance API
JDBC	Java Database Connectivity
ORM	Object Relational Mapping
API	Application Programming Interface
OOP	Object-oriented Programming
CRUD	Create, Update, Delete
JWT	JSON Web Token
CQRS	Command and Query Responsibismolity Segregation
POJO	Plain Old Java Object
HTTP	Hypertext Transfer Protocol
IRC	Internet Relay Chat
DOM	Document Object Model
MUI	Material UI
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
DTO	**Data** Transfer Object
DI	Dependency Injection
IOC	Inversion of Control
JRE	Java Runtime Environment
CORS	Cross-Origin Resource Sharing
W3C	World Wide Web Consortium
IMS	Information Management System (IBM)

VOD Versant Object Database

1 UVOD

U novije vrijeme postoji mnoštvo aplikacija razvijenih primjenom različitih tehnologija. Od mobilnih i desktop aplikacija koje su napravljene za specifični operacijski sustav, pa sve do web aplikacija koje se mogu pokrenuti na svakom sustavu koji ima internetski preglednik. Jedna vrlo zanimljiva karakteristika web aplikacije je ta da se njihov kod piše samo jednom, a može se pokrenuti na bilo kojem sustavu koji podržava potrebne web standarde. Kada se govori o web aplikacijama, većinu vremena se misli na SaaS. SaaS je kratica za *Software as a Service,* što znači da se softver nudi kao servis kojem korisnici mogu pristupiti preko svog internetskog preglednika.

Cilj ovog rada je izraditi potpuno funkcionalnu full-stack web aplikaciju pod nazivom Game Developer Forum (primjenom Spring Boot i ReactJS tehnologija) te detaljno opisati faze izrade uz primjenu navedenih tehnologija. U radu su opisani sastavni dijelovi aplikacije, čemu služe te je detaljno objašnjen proces njihova razvoja. Radom su obuhvaćeni i strukturni i izvedbeni problemi koji su se pojavili prilikom izrade. Sastavni dijelovi aplikacije su serverski dio, baza podataka i klijentski dio te njihovi brojni poddijelovi. Također su opisani primjeri dobrih praksi uz primjenu alata koji olakšavaju i ubrzavaju razvoj ponuđenog rješenja. Za izradu serverske strane koristi se Spring Boot u kombinaciji s nekoliko vanjskih modula. Kao baza podataka koristi se PostgreSQL te ReactJS za izradu klijentske strane (grafičkog korisničkog sučelja) u kombinaciji s razvojnim okvirom za gotove komponente i nekoliko vanjskih modula. Svrha Game developer forum aplikacije je izrada SaaS aplikacije dostupne svima koja će omogućiti stvaranje zajednice za stvaratelje video igara i na taj način potaknuti raspravu i suradnju na projektima.

U prvom poglavlju ukratko je objašnjen razvoj softvera primjenom paradigme punog stoga (eng. Full Stack) i osnovnih pojmova kao što su serverska (eng. Server-side) i klijentska strana (eng. Client-side) te baza podataka. Druga cjelina donosi opis ponuđenog rješenja s pogledom na arhitekturu aplikacije, opis korisničkih uloga i prava, okvirni pregled modela baze podataka i korisničko sučelje. U istoj cjelini je fokus i na razdvajanju administracijskog sučelja od sučelja standardnog korisnika, a definirani su i sigurnosni aspekti aplikacije. Treće poglavlje se bavi izradom aplikacije, što uključuje detaljno opisan proces razvoja serverske strane, odabir baze podataka i izrada klijentske strane. Slijedi cjelina vezana uz verzioniranje projekta s Git-om uz navođenje prednosti takvog pristupa. Predzadnja cjelina je diskusija u kojoj je naglasak stavljen na skalabilnosti i moguća unaprjeđenja aplikacije. Rad završava zaključkom.

2 TEORIJSKE OSNOVE

U ovoj cjelini su opisani osnovni pojmovi koji su potrebni za razumijevanje ovog rada. Cjelina sadrži opis full-stack paradigme razvoja softvera i objašnjava princip rada server-klijent modela.

2.1 RAZVOJ WEB APLIKACIJA PRIMJENOM PARADIGME „POTPUNOG STOGA“ (ENG. FULL STACK DEVELOPMENT)

Paradigma potpunog stoga (često se koristi engleski naziv full-stack) je paradigma razvoja softvera koja se zasniva na izradi obje strane web aplikacije. Sastoji se od serverske strane (backend) i klijentske strane (frontend). Prilikom izrade aplikacija korištenjem paradigme punog stoga, stvaratelj aplikacije će se susresti s većim brojem klijentskih i serverskih tehnologija koje su potrebne za izradu i povezivanje potrebnih dijelova aplikacije. Popularni naziv za skup tehnologija koje se često koriste zajedno za izradu serverske ili klijentske strane je tehnološki stog (eng. Tech stack). Primjeri nekih popularnih tehnoloških stogova su LAMP i MEAN stogovi. LAMP je akronim za Linux operacijski sustav, Apache web server, MySQL bazu podataka i PHP programski jezik.[1]. Koristi se za izradu web aplikacija. MEAN stog se sastoji od MongoDB baze podataka, NodeJS web servera, ExpressJS koji je NodeJS razvojni okvir i Angular frontend frameworka.[2]. Postoje varijacije MEAN stoga kao npr. MERN koji je sličan MEAN stogu, ali koristi ReactJS biblioteku umjesto Angular frameworka.[3].

2.2 BACKEND

Serverska strana (eng. Backend) sadrži svu logiku web aplikacije i time predstavlja srž aplikacije. Spring je popularni open source alat za izradu tzv. enterprise web aplikacija, mikro servisa, bez serverskih (eng. serverless) aplikacija, itd. koristeći Java programski jezik.[4]. Donosi veliki broj funkcionalnosti u igru, kao što su IoC kontejner, Spring MVC web razvojni okvir i još mnogo toga. Jedna od najpopularnijih funkcionalnosti je Springov IoC[1] kontejner koji omogućuje izradu objekata na način da ne ovise o svojim zavisnostima. To postiže na način da objekti sami definiraju svoje zavisnosti dok će kontejner obaviti ostalo.[5]. Spring sam po sebi zahtijeva ponešto ekstenzivnije konfiguriranje za obavljanje određenih stvari. Ovdje dolazimo do Spring Boota. Spring Boot je alat koji olakšava razvijanje Spring aplikacija. Glavna funkcionalnost Spring Boota je automatska konfiguracija Spring aplikacija što znači da se brine za većinu potrebne konfiguracije bibliotekama. Na taj način programer može usmjeriti svoju pozornost i vrijeme na razvijanje same aplikacije umjesto konfiguracije Springa. Spring Boot također olakšava pakiranje Spring aplikacija u samostalne pakete koji se mogu lako pokrenuti. Glavna razlika između Springa i Spring Boota je ta što je Spring gradivni element kod izrade aplikacija, a Spring Boot je dodatak koji olakšava izradu Spring aplikacija. U slijedećoj tablici vidimo ključne razlike između Spring i Spring Boot.

[1] IoC – Inversion of Control container

Tablica 1 Ključne razlike između Spring i Spring Boot [6]

No.	Spring	Spring Boot
1.	Spring is an open-source lightweight framework widely used to develop enterprise applications.	Spring Boot is built on top of the conventional spring framework, widely used to develop REST APIs.
2.	The most important feature of the Spring Framework is dependency injection.	The most important feature of the Spring Boot is Autoconfiguration.
3.	It helps to create a loosely coupled application.	It helps to create a stand-alone application.
4.	To run the Spring application, we need to set the server explicitly.	Spring Boot provides embedded servers such as Tomcat and Jetty etc.
5.	To run the Spring application, a deployment descriptor is required.	There is no requirement for a deployment descriptor.
6.	To create a Spring application, the developers write lots of code.	It reduces the lines of code.
7.	It doesn't provide support for the in-memory database.	It provides support for the in-memory database such as H2.
8.	Developers need to write boilerplate code for smaller tasks.	In Spring Boot, there is reduction in boilerplate code.
9.	Developers have to define dependencies manually in the pom.xml file.	pom.xml file internally handles the required dependencies.

2.3 BAZA PODATAKA

Postoje razne vrste baza podataka koje se koriste u različitim situacijama prema projektnim specifikacijama. Ovdje se navodi samo nekoliko tipova. Relacijske baze spremaju podatke u tablicama i temelje se na vezama između tablica, te su usko povezane sa strukturnim upitnim jezicima (SQL). Kao primjer SQL baze podataka možemo navesti MySQL, PostgreSQL ili SQLite. Baze orijentirane prema dokumentima podatke spremaju u dokumente pisane u JSON, YML, XML i sličnim jezicima. MongoDB je izvrstan primjer baze orijentirane prema dokumentima. Graph baze su baze podataka koje čuvaju podatke u tzv. čvorovima (eng. Node).[7]. Rubovi koji ih

povezuju definiraju relaciju između čvorova.[7]. Primjer takve baze je Amazon Neptune.[7]. Relacijske baze su vertikalno skalabilne dok su baze orijentirane prema dokumentima horizontalno skalabilne.

Tablica 2 Usporedba relacijskih i ne relacijskih baza podataka [8]

Features	Non-Relational	Relational
Availability	High	High
Horizontal Scaling	High	Low
Vertical Scaling	High	High
Data Storage	Optimized for huge data volumes	Medium to large data
Performance	High	Low To Medium
Reliability	Medium	High (Acid)
Complexity	Low	Medium (Joins)
Flexibility	High	Low (Strict-Schema)
Suitability	Suitable For OLAP and OLTP	Suitable For OLTP

Tablica 3 Ključne razlike između relacijske baze, baze orijentirane prema dokumentima i Graf baze
Izvor podataka: [9]

	Relacijske baze	**Baze orijentirane prema dokumentima**	**Graf baze**
Model podataka	Tablica	Dokument	Graf
Shema	Fiksna	Varijabilna	Varijabilna
Skaliranje	Vertikalno	Horizontalno/Vertikalno	Horizontalno/Vertikalno

2.4 FRONTEND

Klijentska strana predstavlja ono što korisnik vidi, tj. grafičko korisničko sučelje koje omogućuje komunikaciju korisnika sa serverskom stranom na pristupačan način. Spaja se na serversku stranu, dohvaća i šalje podatke od i do servera te iste prezentira korisniku.

Ono što je kritično kako bi web aplikacija funkcionirala je logika i spremanje podataka. Dakle, za neku web aplikaciju su nam potrebni samo serverska strana i baza podataka. Korisničko sučelje je koristan dodatak. Serverska strana može raditi bez lijepo dizajniranog sučelja, ali korisnici vjerojatno neće biti zadovoljni kada saznaju da moraju koristiti aplikaciju na način da sami izrađuju i šalju HTTP zahtjeve preko terminala ili nekog alata. Prema tome, glavna funkcija grafičkog korisničkog sučelja je komunikacija sa serverskom stranom i prikazivanje podataka vraćenih od strane servera.

Klasični način izrade grafičkih korisničkih sučelja na web-u (bez korištenja frontend razvojnih okvira) je pisanje „čistog“ HTML-a, CSS-a i JavaScript-a. Takva način rada je vrlo neefikasan, spor, može doći do više grešaka pri izradi logike i generalno će zahtijevati više resursa u slučaju većeg projekta. Aplikacija nije modularna, u smislu da dijelovi aplikacije nisu ponovno iskoristivi, što dovodi do dupliciranja koda.

Za izradu korisničkog web sučelja danas se koriste tzv. razvojni okviri u klijentskom dijelu web aplikacije (eng. frontend web development frameworks). To su alati koji omogućuju brzu i efikasnu izradu web sučelja. Većina tih razvojnih okvira danas koristi sustav komponenti što omogućuje stvaranje samostalnih dijelova korisničkog sučelja koji ne ovise o drugim dijelovima i mogu se ponovno iskoristiti. Na ovaj način omogućuju smanjenje dupliciranog koda i poboljšanje čitljivosti koda. Frontend razvojni okvir je alat koji stvaratelji grafičkih korisničkih sučelja koriste kako bi na efikasan i modularan način izradili sučelje. Apstrahiraju većinu stvari koje su se prije morale raditi same ručno. To rade na puno efikasniji način nego što bi to napravili sami.

Primjeri nekih popularnih frontend razvojnih okvira su VueJS i Angular. ReactJS se često ubraja u ovaj popis, ali on je tehnički gledano biblioteka, a ne razvojni okvir. Jedna od njegovih najvažnijih karakteristika je virtualni DOM o kojem će se detaljnije pričati u poglavlju 4.3.1 Frontend razvojni okvir, koji govori o izradi klijentske strane koristeći ReactJS.[10]. Angular razvojni okvir svoju popularnost dobiva zbog dobrih performansi i čitljivosti koda.[10]. VueJS je razvojni okvir koji je popularan zato što sadrži određene funkcionalnosti koje se nalaze u Angular i ReactJS.[10].

> "However, what we do know is that Vue has the best of both worlds- two-way data binding like Angular and flexibility in code like React."[10].

Naravno, navedeni alati nisu jedini izbor, ali su jako popularni u zajednici te se često koriste za enterprise level aplikacije. Većina frontend frameworka se bazira na sustavu komponenti. Stvaratelj sučelja može sam definirati svoje komponente i logiku vezanu za tu komponentu. Pod

komponentom misli se na izolirani dio koda koji sam po sebi ima sposobnost funkcionirati neovisno o okolini u kojoj se upotrebljava. Odnosno, može se ponovno iskoristiti u drugim dijelovima aplikacije za koje nije originalno napravljena. Time se dobiva određena razina modularnosti i smanjuje se dupliciranje koda. Stvaratelj korisničkog sučelja može sam definirati komponente koje rade prema njihovim potrebama ili može preuzeti već gotove komponente koje su drugi developeri napravili.

ReactJS je frontend biblioteka koja služi za efikasniju izradu web stranica tj. grafičkih korisničkih sučelja baziranih na web tehnologijama. Izumljena je od strane Meta 2013. godine te zajedno sa Angular i Vue.js predstavlja jedan od najpopularnijih izbora za razvoj web aplikacija danas. Bazira se na modularnom sustavu komponenata koje korisnik može sam kreirati i ponovno iskoristiti u svojoj aplikaciji [11]. Time smanjuje ukupno vrijeme potrebno za razvoj aplikacije i smanjuje količinu dupliciranog koda. Osim ReactJS biblioteke koja omogućuje efikasnost pri izradi sučelja, kao baza se koristi Tiller razvojni okvir koji donosi već gotove komponente s profesionalnim izgledom. Većina koda na frontend je napisana u Typescript jeziku zbog veće kontrole nad tipovima podataka. TailwindCSS je CSS razvojni okvir koji je korišten u kombinaciji s Tiller razvojnim okvirom za izradu ove aplikacije.

Osim frameworka za izradu grafičkih korisničkih sučelja poput ReactJS, također postoje framework-ovi koji se baziraju na istim, i stvaratelju nude već gotove komponente za određeni frontend razvojni okvir. Primjer takvih alata su MUI (Material UI) koji sadrži veliki broj predefiniranih komponenti baziranih na ReactJS kao što su navigacijski elementi (navbar), gumbovi, liste, kartice i slično. Još jedan primjer je Tiller razvojni okvir koji se koristi za komponente sučelja u Game Developer Forum aplikaciji.

Kasnije u radu govori se o React virtualnom DOM-u zbog čega je potrebno razumjeti što je ustvari DOM. DOM ili Document Object Model je reprezentacija HTML dokumenta u memoriji.[12]. Navedenoj reprezentaciji se može pristupiti putem koda koristeći DOM API internetski preglednik. DOM je standardizirani način pristupa elementima web dokumenta i neovisan je o programskom jeziku.[12]. Kada bismo razvijali web aplikaciju tj. web stranicu klasičnim putem bez korištenja bilo kakvog frontend frameworka, morali bismo sami direktno modificirati DOM kako bismo manipulirali s elementima web stranice. Kao što je već rečeno, ovo može dovesti do neefikasnosti.

CSS je skraćenica za Cascading Style Sheet. Koristi se na gotovo svakoj web stranici na svijetu u nekom obliku. Radi se o posebnom jeziku koji se koristi za modificiranje izgleda web stranice tj. načina na koji se elementi web stranice prikazuju na ekranu korisnika. Jezik se sastoji od brojnih predefiniranih atributa koji se mogu primijeniti na elemente web stranice koristeći tzv. selektore. Selektor je dio CSS koda koji predstavlja filter za elemente i označava da će se navedeni atributi primijeniti na element koji odgovara tom filteru. Selektor se može dodatno proširiti koristeći pseudo klase što se odnosi na riječi koje se pridodaju selektorima kako bismo omogućili dodatno uvjetovanje prema stanju elementa.[13]. CSS specifikacija je podržana od strane svih popularnih web preglednika u svijetu. Međutim treba obratiti pozornost na to da neki atributi nisu

podržani na svim verzijama svih preglednika. Podržanost nekog atributa se može lako provjeriti kroz tablice kompatibilnosti koje su javno dostupne na web stranicama W3C.

Kao što postoji razvojni okvir za izradu grafičkih korisničkih sučelja i razvojni okvir za gotove komponente koje se koriste na tim sučeljima, također postoji i razvojni okvir za CSS. Omogućuje nam brzo i efikasno modificiranje izgleda stranice i njezinih elemenata tako što dodaje određenu razinu apstrakcije. Korisnik se ne mora brinuti za detalje već svoj fokus može preusmjeriti na ono što je bitno. CSS razvojni okvir će uglavnom sadržavati predefinirane klase (selektore) za često korištena pravila pr. poravnanje elemenata horizontalno ili vertikalno.

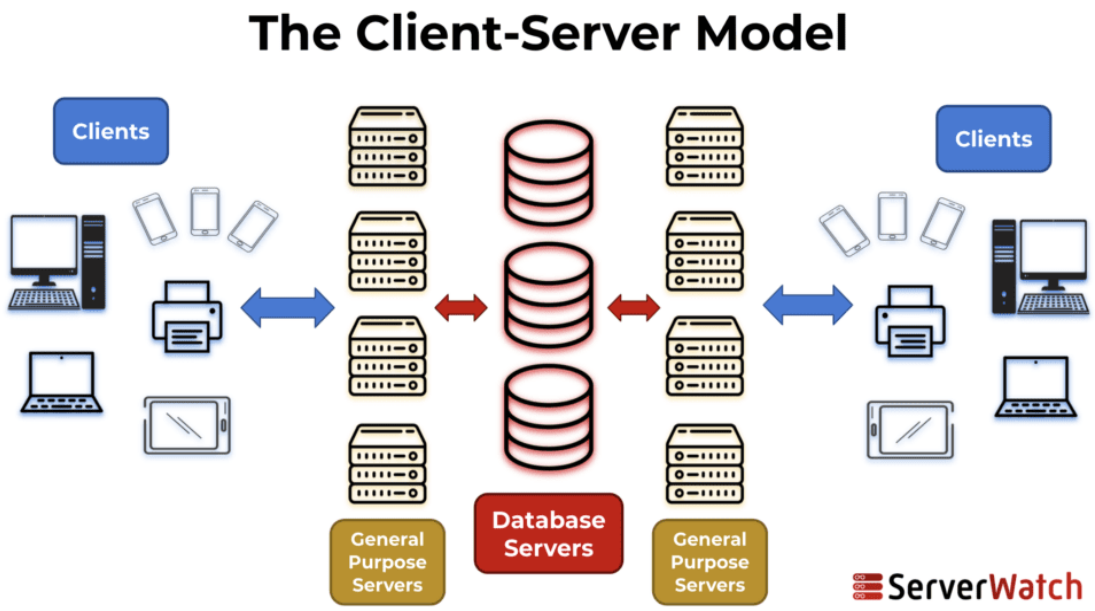
3 OPIS PONUĐENOG RJEŠENJA

Ova cjelina opisuje konkretno ponuđeno rješenje i objašnjava stavke koje su potrebne za razumijevanje daljnjih dijelova ovog rada. Daje se pregled arhitekture aplikacije, koncept rješenja, korisničke uloge, model baze podataka, način na koji odvajamo administracijsko sučelje od korisničkog i sigurnosne aspekte aplikacije.

3.1 ARHITEKTURA APLIKACIJE

U ovom dijelu opisuje se i objašnjava način na koji je napravljena aplikacija tj. od kojih dijelova se sastoji i kako su međusobno povezani.

Full stack aplikacija "Game Developer Forum" se, kao i slične aplikacije, sastoji od tri ključna dijela i njihovih pod dijelova. Tri glavna dijela su serverska strana, baza podataka i klijentska strana tj. grafičko korisničko sučelje. Tehnički gledano svaki od navedenih dijelova je zasebna cjelina koja može funkcionirati neovisno o ostalim cjelinama. Niti jedna cjelina ne donosi korist sama za sebe već mora postojati komunikacija između njih.



Slika 1 Princip organizacije serversko-klijentske arhitekture [15]

Na slici 1 vidljiv je primjer server-klijent modela. Ovaj model je centraliziran u smislu da postoji centralni server na koji se spaja jedan ili više klijentskih uređaja[14]. Na slici se također vide tri prethodno navedene cjeline. To su klijentska strana koja konzumira ponuđeni servis, serverska strana koja spaja bazu podataka i klijentsku stranu, i baza podataka u koju se podaci spremaju, iz nje čitaju ili ažuriraju. U slučaju Game Developer Forum aplikacije, serverska strana ima direktan pristup bazi podataka. Moglo bi se reći da server opće namjene predstavlja filter koji ograničava čitanje i pisanje podataka prema nekim parametrima (npr. ulozi korisnika).

Na serverskoj strani pronalazi se sva logika (servisi) potrebna za rad aplikacije. Ovdje se zaprimaju HTTP zahtjevi od strane klijenta koji se zatim procesiraju pokretanjem neke logike (npr. pisanje u bazu ili čitanje iz baze podataka) nakon čega se šalje povratni odgovor. Na serverskoj strani također se pronalaze sigurnosni mehanizmi koji sprječavaju pokretanje neovlaštenih akcija od strane neautoriziranih korisnika. Ti sigurnosni mehanizmi su bazirani na konceptu uloga i prava što je objašnjeno u cjelini 3.3 Korisničke uloge i prava. Svaka uloga ima prava za izvršavanje određenih akcija i dohvat određenih podataka. Dakle serverska strana je kritični dio aplikacije koji povezuje bazu podataka i klijentsku stranu na siguran način.

Klijentski dio je sve ono što korisnik vidi i s čime vodi interakciju. Međutim, korisničko sučelje je samo dodatak koji olakšava komunikaciju sa serverskom stranom i vizualizira vraćene podatke kako bi olakšao korištenje aplikacije korisniku. Ono što je kritično kako bi web aplikacija funkcionirala je logika i spremanje podataka, što znači da su za jednu web aplikaciju potrebni samo serverska strana i baza podataka. Serverska strana može raditi bez lijepo dizajniranog sučelja, ali korisnici vjerojatno neće biti zadovoljni ako saznaju da moraju koristiti aplikaciju na način da sami

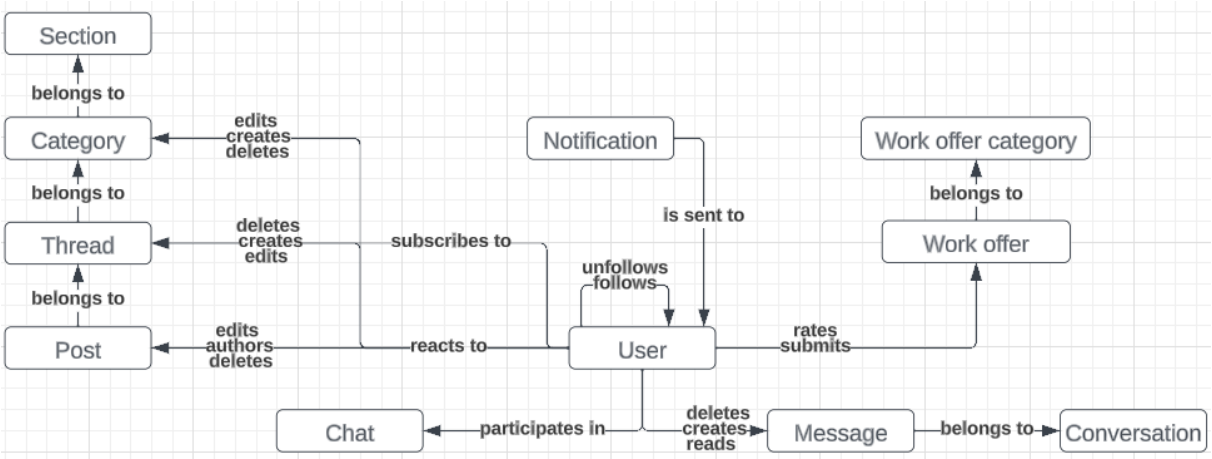
izrađuju i šalju HTTP zahtjeve preko terminala ili nekog alata. Sigurnosni mehanizmi su donekle odraženi i na klijentskoj strani, ali na obrnut način. Serverska strana je ta koja ne dopušta neovlašteno pokretanje akcija i time čuva podatke, a zadaća klijentske strane je da ne dopusti korisniku slanje zahtjeva za pokretanje takve akcije na prvom mjestu. Prevencija pokretanja neovlaštenih akcija od strane korisnika na frontend se odrađuje na način da se provjere uloga i prava trenutno prijavljenog korisnika i ograničava pristup kontrolama na korisničkom sučelju. U pravilu korisnik može zaobići ova ograničenja, ali neće se dogoditi ništa loše ako to i napravi iz razloga što serverska strana neće dopustiti zlonamjernu akciju. Cilj sigurnosnih mehanizama na klijentskoj strani je sprječavanje slanja zahtjeva za koje znamo da neće proći na serverskoj strani. Ograničavanjem kontrola na sučelju daje se korisniku do znanja da nema dovoljno prava za pokretanje neke akcije i smanjuje se promet prema serverskoj strani.

Tablica 4 Prikaz troslojne arhitekture aplikacije

Sloj	Komponente	Funkcionalnosti
Serverska strana (backend)	Server opće upotrebe	Sadrži logiku aplikacije, filtrira upite i brine se o sigurnosti
Baza podataka	Server baze podataka	Spremanje, ažuriranje i čitanje podataka po zahtjevu server opće upotrebe
Klijentska strana (frontend)	Grafičko korisničko sučelje	Komunikacija s serverskom stranom

3.2 KONCEPTUALNI OPIS RJEŠENJA

Ideja iza Game developer forum aplikacije je izrada forum servisa koji će služiti kao platforma za interakciju između programera, 2D i 3D umjetnika, producenta muzike i zvučnih efekata, i ostalih profesija koje spadaju u izradu video igara. Aplikacija je bazirana na konceptu internetskog foruma što znači da se komunikacija većinom događa preko asinkronog modela komunikacije, ali također ima mogućnost sinkrone komunikacije u obliku chat kanala. Cilj aplikacije je omogućiti komunikaciju i dijeljenje sadržaja između stvaratelja video igara na jednostavan i brz način. Aplikacija je strukturirana na način koji omogućuje lako pronalaženje tema i sadržaja koji je relevantan za korisnika aplikacije. U nastavku vidimo blokovski dijagram aplikacije koji okvirno prikazuje sve funkcije aplikacije



Slika 2 Konceptualni dijagram Game Developer Forum aplikacije
Izrađeno prema [47]

Glavni dio aplikacije predstavljaju sobe za raspravu gdje korisnici mogu objavljivati sadržaj i podijeliti ga s ostalim korisnicima. Sadržaj se može filtrirati po različitim parametrima kako bi korisnici mogli lako pronaći ono što žele. Korisnici imaju mogućnost uređivanja i brisanja većine sadržaja kojeg objave u aplikaciji. Uz to mogu i ocjenjivati određene tipove sadržaja kako bi ostalim korisnicima dali do znanja da je neki sadržaj dobre ili loše kvalitete. Aplikacija također ima sinkronu stranu u obliku stranice za dopisivanje u pravom vremenu (eng. Real time chat). Ova funkcija je inspirirana klasičnim IRC kanalima koji su prije bili jako popularni na web stranicama. Kreiranje i ocjenjivanje sadržaja je dozvoljeno samo prijavljenim korisnicima. Naravno, ponekad korisnik želi privatno razgovarati s nekom osobom zbog čega je uvedena i stranica za direktno privatno dopisivanje. Stranica za privatno dopisivanje koristi asinkroni način komunikacije kao i sobe za raspravu. Obavijesti su uvedene u aplikaciju na pasivan način preko zasebnog prozora. Svaki korisnik dobiva obavijesti o različitim aktivnostima kao što su aktivnosti drugih korisnika, pretplate i poruke. Korisnici imaju mogućnost dodati opis na svoj profil u obliku kratke biografije kako bi se ukratko predstavili ostalim korisnicima. Prijavljeni korisnici se mogu međusobno pratiti kako bi dobili obavijest o aktivnostima praćenog korisnika (nove objave).

3.3 KORISNIČKE ULOGE I PRAVA

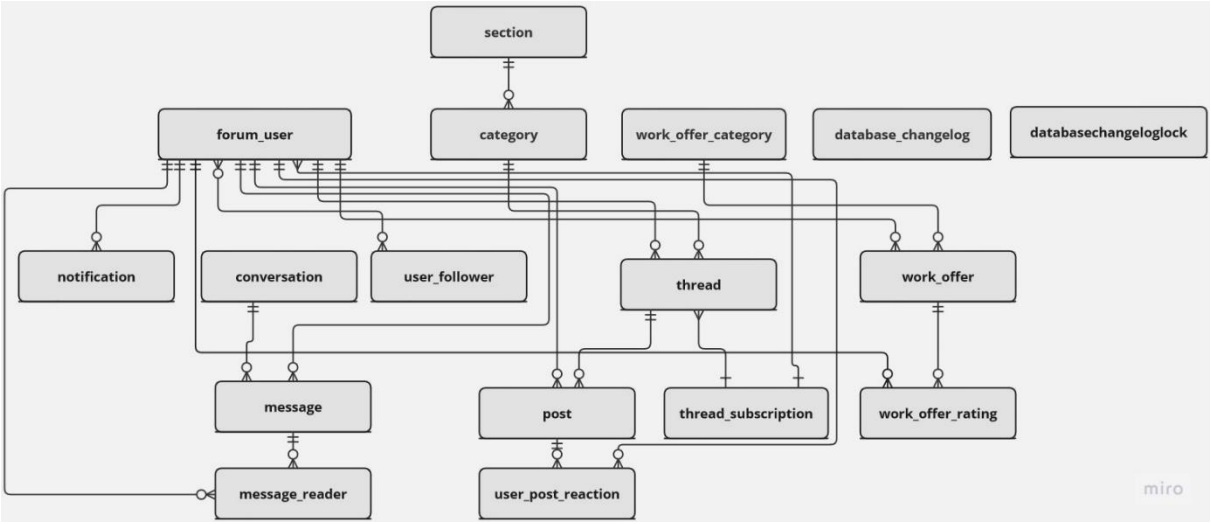
Kao što je već spomenuto ranije, sigurnost aplikacije se bazira na ulogama korisnika koje im daju određena prava. Postoje dvije uloge u aplikaciji, a to su korisnik i administrator. Korisnik je zadana uloga svakog novog korisnika i omogućuje mu standardna prava u aplikaciji. Korisnici imaju prava dohvaćati sadržaj koji se nalazi u kategorijama i sobama za raspravljanje, uređivati i brisati vlastiti sadržaj, stvarati novi sadržaj u dozvoljenim mjestima, itd. Administratori imaju više razine ovlasti. Od korisnika s administratorskom ulogom se očekuje da će održavati red na platformi, što znači da treba imati prava koja to omogućuju. Administrator, isto kao korisnik, može uređivati i brisati svoj sadržaj, ali i sadržaj ostalih korisnika. Također ima pristup stvaranju sadržaja u kanalu za novosti što standardni korisnici ne mogu raditi. U tablici 5 prikazane su korisničke uloge i razine prava koju pojedina kategorija korisnika ima u aplikaciji.

Tablica 5 Prikaz korisničkih uloga i razina autorizacije

Uloga	Razina prava
Administrator	• pregled sadržaja • stvaranje objava u forum odjeljku • stvaranje objava u odjeljku za novosti • uređivanje i brisanje vlastitih objava i objava ostalih korisnika • stvaranje reakcija na objave • ocjenjivanje ponuda za posao • stvaranje privatnih poruka • praćenje korisnika
Registrirani korisnik	• pregled sadržaja • stvaranje objava u forum odjeljku • uređivanje i brisanje vlastitih objava • stvaranje reakcija na objave • ocjenjivanje ponuda za posao • stvaranje privatnih poruka • praćenje korisnika
Neregistrirani korisnik	• pregled sadržaja

3.4 MODEL BAZE PODATAKA

Podaci koji se spremaju u sklopu Game Developer Forum aplikacije su većinom relacijske prirode zbog čega je za ovaj projekt korištena relacijska baza podataka PostgreSQL. Za upravljanje verzijama baze podataka korišten je Liquibase alat o čemu će se detaljno govoriti u nastavku rada. Model se sastoji od tablica koje su međusobno povezane različitim vezama. Na slici 3 prikazan je ERD dijagram baze podataka Game Developer Forum aplikacije uz napomenu kako su polja skrivena zbog preglednosti.



Slika 3 ERD dijagram modela baze podataka Game Developer Forum aplikacije

Odjeljak je bazni kontejner za sav sadržaj. Postoje dva odjeljka - odjeljak za korisničke objave (forum) i odjeljak za novosti koji smije modificirati samo administrator. Odjeljak može, ali i ne mora sadržavati jednu ili više kategorija. Svaka kategorija zatim može sadržavati jednu ili više soba za raspravu. Soba za raspravu opet može sadržavati jednu ili veći broj korisničkih objava. Dio modela koji smo sada opisali je zaslužan za glavnu funkcionalnost aplikacije, a to je asinkrona komunikacija kroz sobe za raspravu. Nakon stvaranja objave u sobi za raspravu korisnik se automatski „pretplaćuje“ na tu sobu za raspravu kako bi dobio obavijesti o odgovorima drugih korisnika. Stvara se novi redak u tablici ***thread_subscription*** koji sadrži broj korisnika i sobe za raspravu. Svaki redak predstavlja jednu pretplatu.

Korisnici mogu ostaviti reakcije na objave u obliku „sviđa mi se“ ili „ne sviđa mi se“ reakcija. Nakon klika na reakciju stvara se redak u tablici ***user_post_reaction***. Jedna reakcija je vezana na samo jednu objavu i samo jednog korisnika, a jedna objava može imati nula ili više reakcija. Svaki korisnik može ostaviti samo jednu reakciju na određenoj objavi.

Obavijesti služe kako bismo korisnicima dali do znanja da se dogodilo nešto što zahtjeva njihovu pažnju pr. da su primili direktnu poruku ili da je netko od praćenih korisnika objavio novi sadržaj. Obavijest može biti usmjerena prema samo jednom korisniku. Isto tako jedan korisnik može primiti mnoštvo obavijesti.

Sustav privatnih poruka se sastoji od ukupno 4 tablice - korisnik, poruka, rasprava i čitatelj poruke. Poruka može biti usmjerena samo jednom korisniku i pripadati samo jednoj raspravi. Rasprava stvara kontekst u kojem se razgovor između dva korisnika održava tj. spaja poruke i korisnike u jednu cjelinu zbog lakšeg filtriranja kod dohvata poruka. Kada korisnik pročita poruku stvara se novi redak u tablici ***message_reader*** koji obilježava da je određeni korisnik pročitao određenu poruku. Tablica ***message_reader*** može biti povezana sa samo jednom porukom i samo jednim korisnikom.

Tablica za praćenje korisnika (***user_follower***) sadrži korisnika koji pokreće proces praćenja i ciljanog korisnika kojeg želimo pratiti. Više korisnika može pratiti korisnika A, i korisnik A može pratiti više korisnika što znači da se radi o many-to-many vezi.

Ponude za posao su također kategorizirane u određene kategorije. Jedna kategorija može, ali i ne mora sadržavati jednu ili više ponuda za posao. Jedna ponuda za posao može biti dio samo jedne kategorije. Svaka ponuda se može ocijeniti od strane korisnika. Ocjene se spremaju u tablicu ***work_offer_rating*** tako što se povezuju ponuda i korisnik koji je unio ocjenu. Svaki korisnik može ocijeniti određenu ponudu samo jednom.

Tablice ***databasechangelog*** i ***databasechangeloglock*** su generirane od strane Liquibase alata i služe za upravljanje verzijama modela baze podataka.

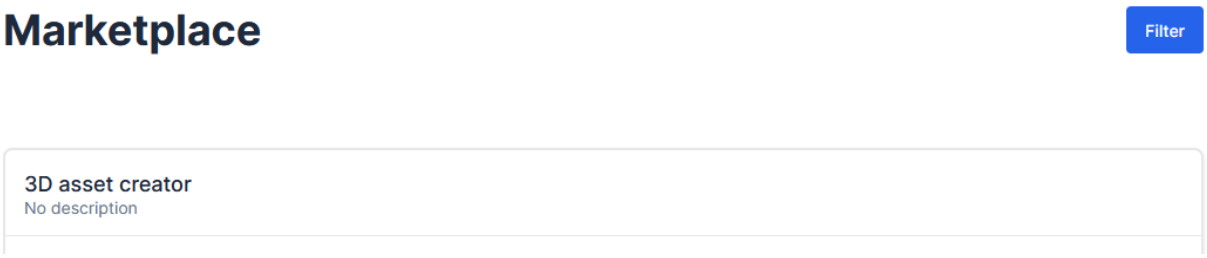
Iako se ne tiče modela baze direktno, važno je napomenuti funkciju „mekog brisanja“ koja se ostvaruje preko modela baze. Meko brisanje znači da će entitet ostati u bazi ali će ga aplikacija preskočiti tijekom čitanja podataka iz baze. To će napraviti ako je određeno polje u retku (npr. deleted) postavljeno na određenu vrijednost.

3.5 ADMINISTRACIJSKO I KORISNIČKO SUČELJE

Svi korisnici koji pristupaju aplikaciji će dobiti istu instancu aplikacije. Postavlja se pitanje kako odvojiti administrativni dio od dijela standardnog korisnika koji ne bi trebao imati pristup administrativnim alatima. Naravno, izrada dvije verzije sučelja, jednu za korisnika, a drugu za administratora nije dobra opcija zbog većeg troška i kompliciranije izvedbe. Ovo se može odraditi na puno elegantniji način koristeći jednu verziju sučelja za sve korisnike. Razdvajanje profila prema ulogama u Game Developer Forum aplikaciji se radi na način da će klijentska strana prvo dohvatiti podatke u trenutno prijavljenom korisniku i ovisno o njihovoj ulozi ograničiti kontrole na korisničkom sučelju. Na ovaj način moramo odraditi sitne provjere na klijentskoj strani, ali dobili smo sučelje koje se adaptira prema ulozi korisnika.

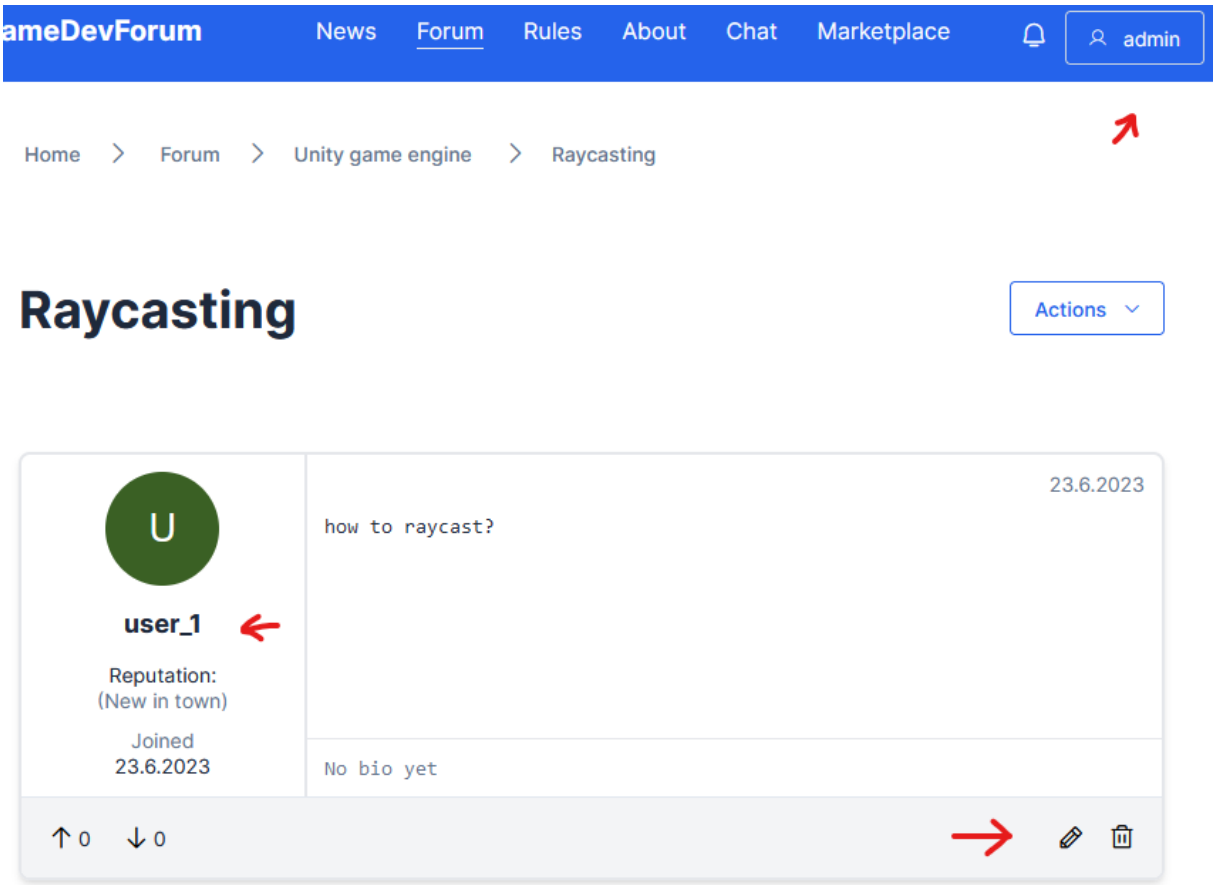
Primjer situacije gdje je potrebno ograničiti kontrole nad korisničkim sučeljem možemo vidjeti na ekranu za pregled kategorija poslova. Kategorije za poslove može kreirati i uređivati samo korisnik s administratorskom ulogom. Zbog toga je potrebno osigurati da te mogućnosti nisu prezentirane ostalim korisnicima. Slijedeće dvije slike prikazuju primjer ograničavanja kontrola nad korisničkim sučeljem ovisno o ulozi korisnika.

Slika 4 Kontrole kategorija poslova kada je korisnik administrator



Slika 5 Kontrole kategorija poslova kada korisnik nije administrator

Na slici 4 vidimo sve moguće kontrole kada je korisnik prijavljen s administratorskim računom, dok na slici 5 vidimo mogućnosti standardnog korisnika s standardnim pravima. Sučelje se prilagodilo korisniku na način da korisnik može inicirati samo one radnje za koje ima potrebna prava. Administrator je taj koji održava red na platformi. Iz tog razloga administrator mora imati mogućnost manipulacije sadržajem drugih korisnika. Na slijedećem primjeru prikazano na slici 6 vidljivo je da administrator može urediti ili obrisati objave drugih korisnika.



Slika 6 Primjer admin kontrola nad objavama ostalih korisnika

3.6 SIGURNOSNI ASPEKTI RJEŠENJA

Ovo poglavlje ukratko opisuje korišteni sigurnosni mehanizam koji osigurava autorizaciju korisnika sa serverskom stranom. Korišteni sigurnosni mehanizam se bazira na JWT tokenima. JWT token je standard za sigurni prijenos informacija koji se bazira na kodiranim JSON objektima.[16]. Detaljnije će se proći JWT tokene u poglavlju 4.2.5 Sigurnost, gdje se govori o sigurnosti na serverskoj strani. Tokeni se generiraju na serverskoj strani i služe za autorizaciju akcija pokrenutih od strane klijenta. Prije dobivanja prvog tokena, klijent se mora autenticirati sa serverskom stranom tako što će priložiti svoje korisničke detalje. Server zatim odgovara s novim JWT tokenom koji će klijent priložiti u svakom budućem zahtjevu. Token se privremeno sprema u lokalno spremište internetskog preglednika kako se korisnik ne bi morao ponovno prijavljivati u sustav za svaki zahtjev. Jedan token obično traje sat vremena nakon čega se mora ponovno generirati. Naravno, ova vrijednost se može podesiti na serverskoj strani aplikacije preko konfiguracijske datoteke. Ukratko token predstavlja korisnika i sva prava koja posjeduje.

4 RAZRADA RAZVOJA APLIKACIJE

Ovim poglavljem detaljnije je opisan sam proces i postupak izrade full stack web aplikacije Game Developer Forum. Cjelina počinje s odabirom tipa i konkretne implementacije baze podataka i upravljanje njezinim verzijama. Nakon toga slijedi opis izrade serverskog dijela aplikacije, sigurnost, izradu chat servera i konačno izradu korisničkog sučelja na klijentskoj strani.

4.1 BAZA PODATAKA

U točki 3.4 prikazan je model baze podataka u vidu ERD dijagrama, a ovdje se detaljnije opisuje proces odabira tipa baze podataka i konkretne implementacije. U cjelini se objašnjava i zašto se za Game Developer Forum koristi relacijska umjesto ne relacijska baza podataka, zatim čemu služi sustav za upravljanje bazom podataka i kako funkcionira upravljanje verzijama baze podataka.

4.1.1 TIP BAZE PODATAKA

Prije samog početka razvoja aplikacije potrebno je donijeti važnu odluku - koji tip baze koristiti. Prethodno spomenuti projekt koji je služio kao inspiracija za Game Developer Forum aplikaciju je koristio bazu podataka orijentiranu prema dokumentima zvanu MongoDB. MongoDB je jedna od najpopularnijih baza podataka današnjice koja omogućuje lako spremanje i čitanje podataka iz kolekcija tzv. dokumenata. Dokument u MongoDB bazi predstavlja jedan JSON objekt. Kolekcija je skup koji se sastoji od većeg broja dokumenata. Jedna MongoDB baza može imati više takvih kolekcija. Kada bismo povukli paralelu s nekom relacijskom bazom, tablica u relacijskim bazama bi predstavljala jednu MongoDB kolekciju a redak u tablici jedan dokument u

kolekciji. Međutim, za ovaj projekt je korišten relacijski tip baze podataka iz jednostavnog razloga što se u aplikaciji pojavljuju podaci koji imaju određeni stupanj međusobne povezanosti. Ovu povezanost je puno lakše prikazati u relacijskoj bazi nego u nekoj ne relacijskoj bazi. Nakon odabira tipa baze podataka, bilo je potrebno još odabrati konkretnu implementaciju baze.

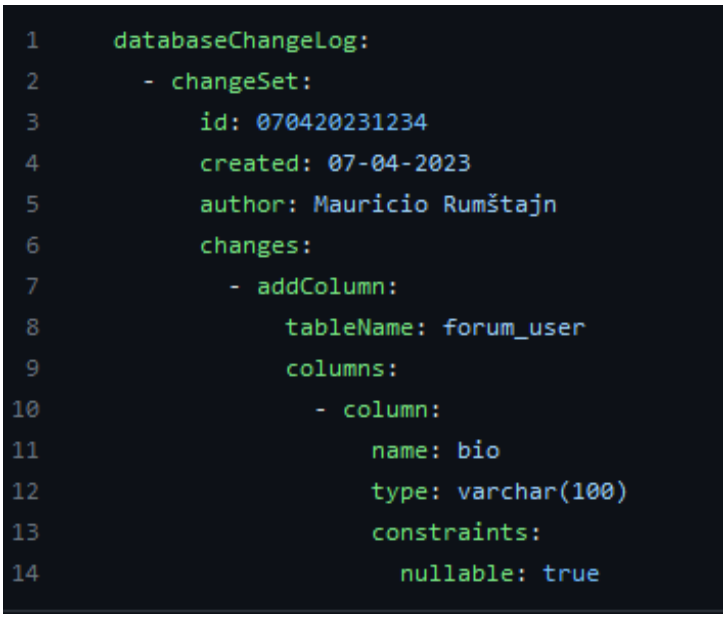
> "Sustav upravljanja relacijskom bazom podataka (eng. Relational Data Base System - RDBMS) skup je programa i mogućnosti koje IT timovima i drugima omogućuju stvaranje, ažuriranje, administriranje i drugu interakciju s relacijskom bazom podataka".[17].

Neki poznati RDBMS su MySQL, SQLite, PostgreSQL, itd. U početku razvitka modela baze podataka dobra je ideja koristiti in-memory bazu podataka koja samo privremeno pohranjuje podatke u memoriji i briše ih nakon gašenja. Jedna implementacija relacijske baze podataka koja ima tu karakteristiku je H2. H2 je lagana implementacija relacijske baze koja omogućuje brzo mijenjanje modela baze prema potrebi, što se u početku dosta često događa. Nakon uspješne implementacije osnovnog modela baze, u redu je prebaciti se na finalnu implementaciju baze koju ćemo koristiti na produkcijskoj okolini. H2 također može spremati podatke na disk ako se tako konfigurira. U ovom slučaju se koristi PostgreSQL, ali bilo koja baza koju smo spomenuli u prethodnom dijelu ima dovoljno funkcija za potrebe ovog projekta.

4.1.2 UPRAVLJANJE VERZIJAMA MODELA BAZE PODATAKA

Pod modelom (relacijske) baze podataka podrazumijevaju se tablice, stupci, ograničenja i veze koje međusobno spajaju tablice. Dakle, model opisuje kompletnu sliku baze i on se konstanto mijenja zbog novih zahtjeva i dorada. Možda ćemo trebati promijeniti tip podatka, dodati neko ograničenje, dodati novi stupac ili čak pobrisati cijelu tablicu. Bilo bi najbolje kada bismo mogli pratiti i zapisivati sve promjene koje se događaju nad modelom baze kako bi imali uvid u to što se promijenilo i kada. Srećom postoje mnogi alati koji omogućuju upravljanje verzijama modela baze podataka koji rade na sličan princip kao što Git alat koji omogućuje upravljanje verzijama koda općenito. Primjeri popularnih alata za upravljanje verzijama modela baze podataka su Flyway i Liquibase alati. Navedeni alati rade na osnovi skripti koje sadrže upute za izvršavanje modifikacija modela baze. Glavna prednost upravljanja verzijama modela baze je to što znamo koje smo naredbe izvršili i u kojem redoslijedu. To omogućuje vraćanje modela u željenu verziju ako je to potrebno (eng. rollback). U nastavku vidljiv je primjer jedne Liquibase[2] skripte koja dodaje novi stupac u tablicu „forum_user“ i definira njegov tip podatka i ograničenja.

[2] https://www.liquibase.org/



Slika 7 Primjer Liquibase skripte

Liquibase alat, koji se također koristi u ovom projektu, podržava više formata za pisanje skripti. Prikazani primjer je pisan u YAML (ili YML) formatu, ali alat također podržava XML, JSON i SQL. Skripta započinje s glavnim objektom koji označava datoteku kao skriptu za upravljanje verzijama. Nakon glavnog objekta slijedi objekt za grupaciju promjena u kojem se nalazi lista pod objekata. Svaki pod objekt predstavlja zasebnu promjenu u bazi i ima svoje opcije koje ovise o tipu promjene. Skripte se najčešće pohranjuju u ***resources*** direktorij kada se radi o Spring Boot aplikacijama. Jedan od načina imenovanja changelog skripti je po verziji izdanja.[18]. Radi se na način da se brojka verzije koristi kao ime datoteke sa prefiksom „changelog-“. Dakle, prva skripta će se zvati „changelog-1.0“ ili „database-changelog-1.0“.[18]. Na ovaj način osigurava se točan redoslijed datoteka u projektu prilikom pregleda i može se jednostavnije identificirati datoteku s pogreškom ako se njezino ime pojavi u logovima aplikacije. Pri pokretanju aplikacije, Liquibase će izvršiti sve nove skripte koje pronađe u konfiguriranom direktoriju. Prvim pokretanjem aplikacije s Liquibase alatom u bazi se stvara nova tablica pod imenom „databasechangelog“. Ova tablica se automatski generira, ako već ne postoji, te sadrži informacije o već izvršenim skriptama.[19]. Nakon svakog pokretanja aplikacije, Liquibase provjeri da li se pojavila nova skripta za upravljanje verzijama u definiranom direktoriju, izvršava novu skriptu i u navedenu tablicu u bazi zapisuje informaciju da je skripta izvršena. Na ovaj način se sprječava duplo izvršavanje već postojećih skripti i sami administrator baze podataka ima uvid u izvršene skripte.

4.2 SERVERSKI DIO

Kao što je već spomenuto, uz serverski dio koristi se nekoliko vanjskih modula koji pomažu u obavljanju specifičnih zadaća. Nrich[3] je alat baziran na Spring Boot razvojnom okviru čiji je cilj olakšati razvoj Java aplikacija [20]. Podijeljen je na nekoliko modula koji služe za različite stvari. Game Developer Forum aplikacija koristi **nrich-serarch** modul za lako pretraživanje baze podataka i paginaciju podataka u repozitorijskom sloju. Omogućuje filtriranje podataka po jednom ili više polja objekta. Uz navedeno pruža dinamičku paginaciju i sortiranje podataka. Na serverskoj strani koristi se već opisani Liquibase alat za upravljanje verzijama modela baze podataka. Korišten je u kombinaciji s PostgreSQL bazom podataka za praćenje izmjena nad modelom podataka korištenjem skripti napisanih u YAML formatu. Java WebSockets biblioteka je korištena za izradu chat servera na serverskoj strani. Pruža mogućnost podizanja servera koji ima sposobnost komunikacije preko web socketa, detekciju spajanja i od spajanja klijenta, primanje i slanje poruka. Od manjih modula se još koriste Lombok za anotacije koje smanjuju količinu „boilerplate koda“[4], model mapper za mapiranje objekata u druge objekte i Java JWT za generiranje i provjeru JSON web tokena. Ranije smo naveli da se za bazu podataka koristi PostgreSQL baza koja se pokreće korištenjem Docker alata za virtualizaciju. Zbog lakšeg razumijevanja, serverski dio je podijeljen na nekoliko slojeva. To su podatkovni ili repozitorijski sloj, servisni sloj i web ili kontrolorski sloj o kojima će se više govoriti u nastavku rada .

4.2.1 PODATKOVNI SLOJ

Većina web aplikacije ima potrebu pohraniti neke podatke. To je zadaća podatkovnog sloja. Spring Boot aplikacije se dijele na nekoliko sastavnih slojeva. To su prezentacijski, servisni, „persistance“, i sloj baze podataka.[21]. Međutim, ovo poglavlje se neće referencirati na općenite slojeve Spring Boot aplikacija već na slojeve na koje je Game Developer Forum aplikacija podijeljena zbog kategorizacije odgovornosti sastavnih dijelova i boljeg razumijevanja.

Podatkovni sloj služi kao poveznica između baze podataka i servisnog sloja serverske strane te se brine o IO (input/output) operacijama nad bazom. Podatkovni sloj tzv. repozitorijski sloj sastoji se od repozitorija koji su definirani za svaki entitet. Svaki tip repozitorija je Java sučelje koje proširuje JPARepository. To je sučelje od kojeg se nasljeđuju metode za IO operacije. Te metode uključuju pronalazak po ID broju, spremanje i brisanje. Metode kao što su dohvat po identifikatoru ili dohvat svih elemenata, većinu vremena izgledaju jako slično za različite entitete. Kako bih izbjegli pisanje sličnog koda na više mjesta možemo koristiti Spring Data JPA. Spring Dana JPA je modul koji automatski definira implementacije osnovnih IO operacija za nas.[22]. Moguće je definirati vlastite metode kada je potrebno detaljnije filtriranje entiteta pr. filtriranje objekata po određenom polju. Tada developer može definirati metode prateći posebnu konvenciju

[3] https://croz.net/news/nrich/

[4] „Boilerplate kod“ je naziv koji koristimo kako bi opisali kod koji se često ponavlja (pr. getter i setter metode)

imenovanja prema kojoj Spring Data JPA izrađuje implementaciju za izvršavanje željene promjene ili dohvata podataka iz baze podataka.[23].

JPA (Jakarta Persistance API, prethodno Java Persistance API) je samo standard ili podloga koja definira pravila za upravljanje s vezanim podatcima čiji vijek trajanja prelazi vijek aplikacije.[24]. To znači da ne sadrži konkretnu logiku već sadrži konvencije koje navode implementatore. Hibernate je jedna od mogućih implementacija JPA i ujedno i implementacija koja se koristi u ovoj aplikaciji. Hibernate je ORM (Object relational mapping) alat koji se koristi u Game Developer Forum aplikaciji za mapiranje ili prevođenje podataka iz baze podataka u odgovarajuće Java objekte zbog lakšeg manipuliranja s podatcima.

U kontekstu ove aplikacije i općenito Spring Boot aplikacija, entitet je predstavljen Java klasom koja sadrži sva polja koja se očekuju u tablici u bazi podataka. Entitet klase se anotiraju sa @Entity anotacijom koja potječe iz JPA i označava klasu kao entitet koji će se čitati i zapisivati u bazu. Promjena polja objekta entiteta će u pravilu uzrokovati istu promjenu u retku tablice iz koje entitet potječe unutar baze podataka. Hibernate podržava klasične OOP koncepte kao što su nasljeđivanje i polimorfizam što znači da se mogu modelirati vrlo kompleksne veze između entiteta.[25]. Koristeći navedene koncepte može se predstaviti vrlo kompleksan model baze s Java klasama. Entitet klase je moguće dodatno konfigurirati raznim anotacijama. Primjer takvih anotacija su @Where anotacija u kombinaciji s @SQLDelete anotacijom. Koristeći ove dvije anotacije može se implementirati "meko brisanje" iz baze. Pojam „meko brisanje" znači da entitet ostaje u bazi ali se smatra izbrisanim. Dodaje se novi stupac u tablicu (npr. „deleted") koji je tipa boolean. Koristimo @Where anotaciju kako bi JPA implementaciji dali do znanja da ne treba učitavati taj redak iz tablice ako je vrijednost polja postavljena na istinito stanje, tj. ako je entitet „meko izbrisan". Vrijednost polja se postavlja tako što se s @SQLDelete anotacijom nadjača funkcionalnost brisanja entiteta u JPA implementaciji. Umjesto kompletnog brisanja entiteta izvesti će se navedena SQL naredba koja će na primjer postaviti vrijednost „deleted" polja na istinito stanje.

```
@Entity
@Getter
@Setter
@JsonIgnoreProperties(value = {"content"}, allowGetters = true)
@SQLDelete(sql = "UPDATE message SET deleted = true WHERE id = ?")
public class Message {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String content;

    private LocalDateTime creationDateTime;
```

Slika 8 Primjer nadjačavanja metode brisanja

Naravno, podatkovni sloj se prvo mora povezati s bazom podataka kako bi mogao izvesti IO operacije. Taj problem rješava još jedan vrlo popularan alat pod nazivom JDBC (kratica za Java Database Connectivity). JDBC je također kao JPA, samo API ili skup konvencija koji ne sadržava konkretnu logiku već se mora implementirati. JDBC API se koristi za izvršavanje operacija nad bazom podataka koristeći Java programski jezik na način da se zahtjevi za podacima iz baze pretvore u naredbe koje sustav za upravljanje bazom razumije.[26]. Korištenjem JDBC API-a postiže se standardizacija povezivanja i interakcije s bazom podataka. Svaka implementacija prati isti obrazac po kojem definira metode za interakciju s bazom podataka preko koda, samo to odrađuje na svoj način, nešto slično obrascu ponašanja strategija (eng. Strategy Pattern). Postoji jedno sučelje koje ima više implementacija između kojih možemo birati. Postoji mnogo implementacija JDBC API-a u obliku drivera za pojedine tipove baza podataka.[26]. Na primjer, u ovom projektu se koristi JDBC API s PostgreSQL driver implementacijom koja ima konkretnu logiku potrebnu za obavljanje IO operacija nad Postgre bazom.

4.2.2 SERVISNI SLOJ

Ovaj sloj sadrži svu poslovnu logiku potrebnu za procesiranje zahtjeva i vraćanje informacija klijentu. U servisnom sloju pronalazimo definicije i implementacije servisa čija je zadaća izvršavati poslovnu logiku. Svaki servis je svoja logička cjelina koja obavlja neku zadaću. Standardna praksa je da se definira i implementira servis za svaki entitet s kojim korisnik može manipulirati na neki način. Naravno, servis ne mora biti vezan za neki entitet već može služiti kao pomoćni servis za agregaciju ili generiranja podataka. Servis i servisne metode se definiraju tako da se kreira Java sučelje koje sadrži definiciju metoda potrebnih za taj servis i konkretnu implementaciju tog sučelja. Sučelje predstavlja nacrt servisa i definira potrebni skup mogućnosti. Servisne metode se u većini slučajeva svode na CRUD operacije (eng. Create, Read, Update, and Delete - stvori, pročitaj, promjeni i izbriši). Konkretna implementacija servisa koristi prethodno definirane repozitorije ili već postojeće servise kako bi izvela IO operacije nad podatcima u bazi podataka.

Sama implementacija servisa nije dovoljna da bi se on mogao koristiti. Klasa se mora nekako registrirati kao kandidat za automatsko detektiranje od strane Spring Boot-a. Ovdje dolazimo do @Component anotacije. Ova anotacija signalizira Spring Boot-u da je klasa u pitanju kandidat za automatsko detektiranje što znači da će se moći koristiti u ostatku aplikacije kada je skenirana. Međutim, postoji specifičnija anotacija koja je napravljena upravo zbog servisnih klasa. Radi se o @Service anotaciji koja je proširenje već spomenute @Component anotacije. Servisna anotacija, osim toga što označava klasu kao komponentu koja se može otkriti od strane Spring Boot-a, također označava klasu kao servis koji spada pod servisni sloj.

Do problema dolazi kada servisi trebaju koristiti jedni druge. Jedan od najčešćih problema se javlja kada imamo dva servisa, pr. servis A i servis B koji međusobno ovise jedan o drugom. Servis A kompozicijski sadrži servis B, a servis B kompozicijski sadrži servis A. Stvara se ciklička petlja. Aplikacija se neće pokrenuti zbog toga što je Spring Boot detektirao potencijalnu petlju pri

skeniranju klasa. Jedno od mogućih rješenja za ovaj problem je korištenje CQRS patterna (kratica za „Command and Query Responsibility Segragation“). Kao što samo ime govori, ovaj pattern se bazira na ideji da se CRUD operacije odvoje na dio s čitanjem i dio s pisanjem.[27]. Dakle, jedan servis dijeli se na dvije definicije i dvije implementacije. Operacije čitanja se definiraju u zasebnom sučelju te se izrađuje zasebna implementacija tog sučelja koja se također anotira sa @Service anotacijom. Isto tako operacije pisanja, uređivanja i brisanja odvajaju se u drugo sučelje i radi se implementacija na isti način. Nakon ovog restrukturiranja koda više ne bi trebalo doći do problema s cikličkom petljom. Još jedno rješenje koje sprječava stvaranje cikličke petlje je anotacija kompozicijske implementacije servisa B u klasi servisa A s @Lazy anotacijom što sprječava trenutno učitavanje anotiranog polja, već će se učitati kada je potrebno.[28]. Poželjno je koristiti prvu opciju umjesto druge zbog čitljivosti koda i negativnog utjecaja @Lazy anotacije na performanse aplikacije.[28]. U projektu je vrlo često korišten CQRS predložak s ciljem razdvajanja operacija čitanja i pisanja u servisima koji trebaju imati mogućnost izvođenja oba tipa operacija. Iznimke gdje nije potrebno koristiti CQRS predložak su servisi koji imaju servisne metode samo jednog tipa tj. izvode samo operacije čitanja ili pisanja ili servisi koji rade nešto sasvim drugo pr. generiraju ili agregiraju podatke.

Osim rješavanja problema s cikličkim zavisnostima, CQRS predložak se još koristi zbog poboljšanja performansi, sigurnosti, korištenja različitih tipova podataka kod čitanja i pisanja i za poboljšanje čitljivosti.[27]. Kao rezultat korištenja ovog obrasca dobivaju se odvojene definicije servisnih metoda za čitanje i pisanje te odvojene implementacije istih.[27] Takav kod je lakše održavati i razumjeti zbog odvojenosti odgovornosti i zbog toga što imaju dva manja servisa za razumjeti u usporedbi s jednim velikim servisom.

Kao što je već spomenuto na početku poglavlja, u servisnom sloju pronalaze se implementacije servisa. Gotovo svaki entitet ima dvije definicije servisa i njihove implementacije. Jedna služi za čitanje dok je druga za pisanje (command i query). Command servisi će uglavnom sadržavati repozitorij kao kompozicijski element. Koristeći repozitorij mogu spremati i modificirati elemente u bazi. Njihov pandan, query servis će raditi upravo suprotno. Također kompozicijski sadrži repozitorij ili druge query servise preko kojih će povlačiti i vratiti informacije pozivatelju.

4.2.3 WEB SLOJ

Zadnji sloj za opisati je takozvani web, kontrolorski ili upravljački sloj. Radi se o sloju koji predstavlja poveznicu tj. most između serverskog dijela i klijentskog korisničkog sučelja. U ovom slučaju ta povezanost nastaje kao rezultat korištenja REST arhitekture. REST ili Representational State Transfer je arhitekturni stil za distribuirane web sustave. [29]. Sadrži konvencije koje bi svaki RESTful API trebao pratiti ako želimo da se smatra REST arhitekturom. Važno je napomenuti kako Game Developer Forum ***nije izrađen prema REST konvencijama, to jest prati ih vrlo labavo i time se ne može smatrati potpunom RESTful aplikacijom***.

Nadalje, POJO (eng. Plain Old Java Object) je pojam koji je potreban za razumijevanje ovog poglavlja. Prema definiciji POJO je klasa koja ima zadani konstruktor bez argumenata, ima

definirana neka polja i pristupne metode (Getter i Setter metode).[30]. Dakle POJO se može definirati kao podatkovni kontejner.

Sastavni elementi web sloja su tzv. web kontrolori. Zadaća kontrolora je primanje zahtjeva od strane klijenta prema određenom filteru rute, mapiranje (preslikavanje) dolazećih podataka u predviđene klase i procesiranje zahtjeva. Pod procesiranjem podrazumijeva se predaja samog zahtjeva servisu koji je odgovoran za primljeni tip zahtjeva, mapiranje odgovora u izlazni tip podatka i slanje istog nazad klijentu s određenim HTTP statusom. Jedan kontrolor može sadržavati više ruta. Ruta ili krajnja točka predstavlja softverski definiran ulaz na serversku stranu. Svakoj krajnjoj točki se može definirati filter u obliku URL putanje (ruta ili eng. route) s kojom se putanja zahtjeva mora poklopiti kako bi zahtjev bio primljen na tu krajnju točku. Samo one točke čiji filter odgovara zahtjevu će primiti nadolazeći zahtjev. Klijent se spaja na krajnju točku i šalje zahtjev formatiran u HTTP standardu. Zahtjev će sadržavati zaglavlje koje sadrži polja s predviđenim vrijednostima (npr. Authorization polja za autorizaciju ili Content-Type za tip sadržaja) i glavni sadržaj (ako tip zahtjeva podržava sadržaj i ako ga server očekuje).

Zahtjev se prima na krajnjoj točki i dobiveni podatci se mapiraju u predviđene klase. Mapiranje se u slučaju Game Developer Forum aplikacije odrađuje preko ModelMapper klase koja je dio povezanog vanjskog modula. Mapiranje (prevođenje objekta u objekt) se radi prema sličnosti imena polja u objektima. Rezultat mapiranja je objekt koji svojim poljima predstavlja instancu primljenog klijentskog zahtijeva. Nakon mapiranja objekta slijedi procesiranje zahtjeva. Ovaj proces se ne odvija u kontrolorskom sloju već u servisnom sloju. Na ovaj način smo odvojili odgovornosti kontrolora od odgovornosti servisa što, kao i kod CQRS patterna, čini kod lakšim za održavanje i čitanje. Mapirani zahtjev se predaje odgovornom servisu u obliku POJO (Plain Old Java Object) objekta čija polja odgovaraju poljima primljenog HTTP zahtijeva. Servis obrađuje zahtjev preko svoje poslovne logike i vraća rezultat (ako je potrebno vratiti rezultat) kontroloru. U pravilu bismo mogli klijentu poslati „sirov“ entitet koji se vrača iz servisnog sloja, ali to nije dobra praksa iz razloga što ne želimo klijentu uvijek poslati potpuni entitet. Ponekad ne želimo klijentskoj strani poslati sva polja koja se nalaze u entitetu nego samo ona polja koja su relevantna za odrađenu akciju. Upravo zbog tog razloga izvodimo mapiranje u još jedan POJO koji sadrži samo ona polja koja želimo vratiti. Prilikom mapiranje dolazi do odbacivanja svih ostalih polja koja nisu definirana u objektu za odgovor. Objekt koji se koristi za slanje odgovora se još naziva i DTO ili Data Transfer Object. Dobra praksa je napisati jedan DTO za svaki zahtjev i odgovor koji će server primiti ili poslati. Na taj način imamo evidenciju svih mogućih zahtjeva i odgovora koje server može zaprimiti ili poslati. Mapirani objekt koji predstavlja odgovor se zatim serializira u JSON format preko Jackson modula te se šalje nazad klijentu u obliku „sirovog“ JSON-a (običnog teksta).[31]. Jackson je alat koji se koristi u Spring Boot aplikacijama za manipulaciju podataka u JSON formatu. Klijent će znati kako interpretirati dobiveni odgovor zbog toga što će serverska strana postaviti polje „Content-Type“ u zaglavlja odgovora na vrijednost „application/json“.[32]. To će reći klijentu da se radi o sadržaju napisanom u JSON formatu te da ga treba tako interpretirati.[32].

U konačnici, zaključuje se kako je web sloj poveznica između serverske i klijentske strane te služi kao posrednik između klijenta i servisnog sloja na serverskoj strani. Uz to još prevodi podatke iz ulaznog u izlazni format.

4.2.4 DEPENDENCY INJECTION OBRAZAC I IoC KONTEJNER

Dependency injection (ili DI) je obrazac pisanja koda na način da klase ne ovise o svojim zavisnostima.[33]. Mehanizam se bazira na „ubrizgavanju“ zavisnosti u instancirani objekt. Zavisnosti se mogu „ubrizgati“ u objekt na više načina. Korištenjem konstruktora, setter metoda i polja klase. Klase napisane koristeći ovaj obrazac mogu funkcionirati kao izolirana cjelina koja dobiva sve što joj je potrebno za rad izvana. Naravno, opisani proces se ne izvodi sam od sebe već je potrebno koristiti neki alat koji podržava dependency injection. Primjeri takvih alata uključuju Weld, Dagger, i alat o koji je već spomenut, Spring odnosno Spring Boot.[33]. Kako bi se razdvojile zavisnosti između klasa, može se definirati sučelja te koristiti ta sučelja kompozicijski umjesto originalnih zavisnosti.[33]. Klasi kojoj je potrebna implementacija tog sučelja može se izvana proslijediti (ili „ubrizgati“) željena implementacija. Može se primijetiti kako se DI ne razlikuje puno od jednog dobro poznatog obrasca ponašanja zvanog Strategy obrazac (predloška) ponašanja. Strategy obrazac ponašanja koristi sučelja kako bi apstrahirao algoritme ili ponašanja koja se mogu koristiti. Definira se sučelje koje će svojim metodama stvoriti nacrt željenog ponašanja. Imati će i nekoliko implementacija tog sučelja između kojih se može birati. DI funkcionira na vrlo sličan princip. Razlika je u tome što strategy pattern apstrahira ponašanje dok je DI mehanizam za stvaranje objekata.

IoC kontejner (skraćenica za Inversion of Control) odnosi se na mehanizam koji predaje kontrolu toka aplikacije nekom vanjskom elementu. U ovom slučaju to je implementacija IoC kontejnera koja dolazi sa Spring Boot alatom. Cijeli koncept služi kako bi se mogle definirati klase koje ne izrađuju svoje zavisnosti direktno nego ih samo definiraju, a IoC kontejner odrađuje ostalo.[34]. Na ovaj način dobivene su klase koje nisu ovisne o određenim zavisnostima već se mogu koristiti na više načina i to koristeći bilo koje kompatibilne zavisnosti. Implementacija IoC kontejnera je zaslužna za instanciranje objekata tako što će im proslijediti potrebne argumente odnosno zavisnosti nakon njihova kreiranja.[34]. IoC kontejner može se shvatiti kao kutiju s alatom koji je potreban za neki rad. Prilikom instanciranja objekata IoC kontejner će detektirati da su tom objektu potrebne određene zavisnosti i potražiti će ih u svojoj kutiji s alatom. U nastavku poglavlja govori se o konfiguraciji kontejnera i jednoj od najbitnijih klasa u Spring Boot-u, a to je Bean klasa.

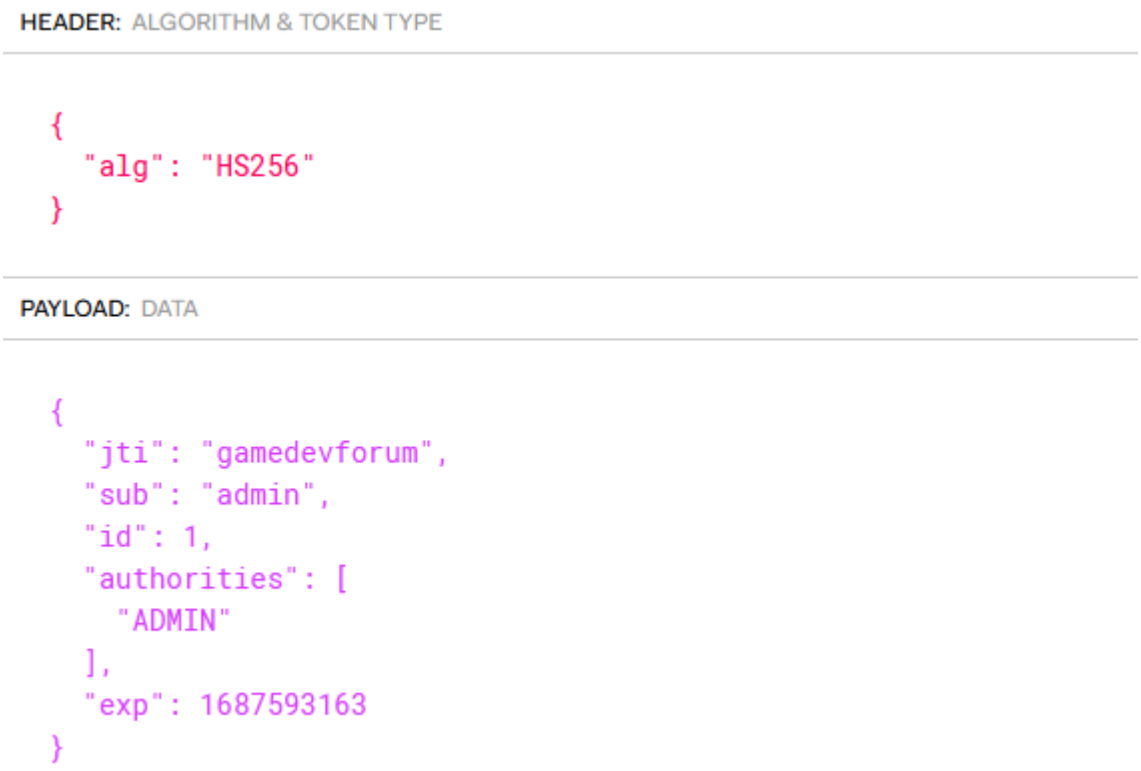
Jedno od prvih pitanja svakog programera koji je prvi put probao koristiti Spring Boot je vezano za Bean. Bean je objekt s kojim Spring Boot upravlja i koristi ih kako bi izgradio druge objekte koji ovise o njima.[35]. Ranije je spomenuto kako IoC kontejner možemo shvatiti kao kutiju alata. Bean u tom slučaju predstavlja jedan komad alata. Potrebno je konfigurirati definicije Bean objekata kako bi IoC kontejner znao s kojim objektima treba upravljati. Sve Bean definicije u Game Developer Forum aplikaciji su definirane u obliku anotacija u kodu, ali podržani su i drugi načini konfiguracija (na primjer XML).

4.2.5 SIGURNOST

Svaka web aplikacija koja korisniku nudi mogućnost objave i manipulacije sadržaja treba imati određene sigurnosne mjere kako bi se spriječilo pokretanje operacija od strane neautoriziranih korisnika. Tu razinu sigurnosti je moguće postići na mnogo načina, ali u ovom poglavlju je pojašnjeno kako je to napravljeno za Game Developer Forum aplikaciju korištenjem JWT token metode.

Prije nego bude govora o konkretnim implementacijama sigurnosnih metoda, kratko će se prikazati osnovna sigurnosna arhitektura svake Spring Boot aplikacije. Ono što prethodi dolasku zahtjeva koji je poslan od strane klijenta do servera je prolazak kroz niz sigurnosnih filtera koji se nalaze u tzv. lancu sigurnosnih filtera.[36]. Lanac filtera sadrži mnoštvo filtera koji mogu modificirati zahtjev, proslijediti ga dalje u lanac ili odlučiti da će sami procesirati zahtjev. Nakon što je zahtjev prošao kroz lanac filtera, dolazi do servera. Pod serverom misli se na implementaciju Java servera (koje Spring Boot koristi u pozadini za procesiranje HTTP zahtijeva i slanje odgovora). Moguće je definirani vlastite filtere proširenjem određenih klasa. Upravo to je i napravljeno u ovoj aplikaciji u obliku JWT filtera. Filter provjerava postoji li polje „Authorization“ u zaglavlju zahtjeva. Ako postoji vrijednost će se pročitati i token će se validirati preko digitalnog potpisa izdavača. Server može provjeriti potpis zato što ga je on izdao i ima tajni ključ koji se koristio prilikom potpisivanja tokena. Ako je token valjan, korisnik se bilježi kao autenticiran u tzv. sigurnosnom kontekstu trenutnog zahtjeva. Sigurnosni kontekst služi za spremanje autenticiranog korisnika i pristup tom podatku za vrijeme trajanja trenutnog zahtijeva kako bismo mogli provjeriti da li je korisnik doista autenticiran i omogućiti mu pristup zaštićenom sadržaju ovisno o ulozi. U slučaju da token nije valjan ili nije priložen, sigurnosni kontekst će ostati prazan i server neće dopustiti pristup zaštićenom sadržaju već će vratiti HTTP status 401 (Unauthorized). Naravno, ovo je jako površan pogled na sigurnosnu arhitekturu Spring Boot-a, ali sadrži sve informacije potrebne za razumijevanje nastavka ovog poglavlja.

JWT ili JSON web token je popularni i široko korišten sigurnosti standard koji služi za siguran prijenos informacija.[16]. Standard funkcionira na bazi kodiranih JSON objekata koji se prenose preko mreže.[16]. Njihov sadržaj se može provjeriti zbog toga što su digitalno potpisani od strane izdavača tokena.[16]. Token se može potpisati na različite načine pr. korištenjem RSA algoritma s privatnim i javnim ključem ili HMAC algoritma ako ga želimo potpisati koristeći jednostavnu tajnu riječ.[16]. U slučaju HMAC algoritma tajna riječ se navodi kao „secret“.[16]. Game Developer Forum aplikacija koristi HS256 (HMAC + SHA256) algoritam za potpisivanje tokena s tajnom riječi koju zna samo serverska strana aplikacije kao što je prikazano na slici 9.



Slika 9 Primjer dekodiranog Json Web Tokena

U nastavku se opisuje struktura JWT tokena prema RFC 7519 standardu. Sadržaj JWT tokena se sastoji od tri glavna dijela. Prvi dio je zaglavlje tokena koje služi za spremanje meta podataka o samom tokenu. U zaglavlju nalazimo dva polja. Prvo polje je tip tokena, što će u ovom slučaju biti „JWT“. Drugo polje je naziv algoritma s kojim se potpisuje token što će biti „HS256“. Nakon zaglavlja dolazi tijelo tokena. Ovaj dio sadrži glavni sadržaj koji se koristi za autorizaciju (u ovom slučaju se koristi za autorizaciju, ali može se koristiti i za ostale svrhe). Podatci za autorizaciju se nazivaju tvrdnje (eng. claims). Tvrdnja predstavlja informaciju vezanu za entitet koji se pokušava autorizirati. Tvrdnja može biti da entitet ima određenu ulogu u sustavu koja mu omogućuje pristupanje i manipuliranje određenog sadržaja. Postoji nekoliko ugrađenih tvrdnji koje se postavljaju zbog dobre prakse. Pri generiranju tokena u aplikaciji, postavljena su dva važna polja. To su entitet koji se autorizira i vrijeme isteka tokena. Entitet koji se autorizira se postavlja na vrijednost korisničkog imena. Vrijeme isteka je obično sat vremena što je i korišteno u ovom projektu. Nakon isteka tokena potrebno je zatražiti novi na isti način kako je korisnik prvi put zatražio token.[16].

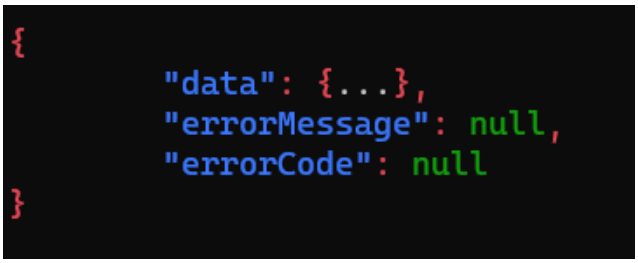
Sada kada imamo osnovno razumijevanje JWT tokena možemo prijeći na sami proces autorizacije. Prije dobivanja prvog tokena, korisnik prvo mora proći kroz autentikacijski proces kako bi server utvrdio identitet korisnika. U slučaju Game Developer Forum aplikacije taj proces se odvija preko zasebne pristupne točke na serveru. Klijent šalje svoje korisničko ime i lozinku na predefiniranu rutu (uobičajeno / login) nakon čega server provjerava valjanost unesenih podataka.

Proces provjere identiteta se vrši tako da se iz baze pročita redak s tim korisničkim imenom, unesena lozinka se raspršuje (eng. hashing) određenim algoritmom za raspršivanje i dobivena vrijednost se usporedi sa već postojećom vrijednosti lozinke koja je unesena pri registraciji računa. Ako se vrijednosti poklapaju, server će generirati novi JWT token s korisničkim ID-om, postaviti ulogu koja je spremljena u bazi te digitalno potpisati i vratiti JWT token klijentu na korištenje. Nakon toga klijent će morati priložiti dobiveni JWT token u svakom budućem HTTP zahtjevu prema serveru tako što će ga poslati u „Authorization“ polju u zaglavlju zahtjeva s prefiksom „Bearer“.

Raspršivanje (eng. hashing) je proces pretvorbe podatka (npr. stringa) u drugi oblik korištenjem jednosmjernih matematičkih funkcija. Dakle, jednom raspršena vrijednost se više ne može dešifrirati i pročitati. Raspršivanje lozinki se bazira na ovoj karakteristici. Aplikacija koja podržava stvaranje korisničkih računa se mora na neki način uvjeriti da je korisnik koji se pokušava prijaviti u svoj korisnički račun uistinu vlasnik tog računa. To ćemo znati tako što će se unesena lozinka provjeriti. Međutim jedan od glavnih problema kod sigurnosti je pohranjivanje korisničkih lozinki. Pri registraciji svog računa, korisnik unosi željenu lozinku. Pitanje je gdje i kako će se ta lozinka sigurno pohraniti. Najlogičnija solucija je spremiti lozinku u bazu kao što spremamo i druge tipove podataka. Problem je u tome što je lozinka vrlo osjetljivi podatak koji nije sigurno spremiti u bazu u tekstualnom obliku zbog mogućih napada na bazu. U slučaju da je napadač uspješan u zaobilaženju sigurnosnih mjera i uspije pristupiti podatcima u bazi, moći će vidjeti lozinku i iskoristiti ju za prijavu na taj račun. Iz ovog razloga koristimo raspršivanje lozinki. Nakon uspješne registracije novog računa, lozinka se raspršuje korištenjem jedne od dostupnih implementacija PasswordEncoder sučelja. U ovom slučaju to je BCryptPasswordEncoder. Lozinka se raspršuje i nečitljivi rezultat se sprema u bazu. Razlika je što sada napadač koji uspije pristupiti bazi ne zna koja je točna lozinka. U slučaju da se napadač pokuša prijaviti u račun koristeći raspršenu vrijednost iz baze, sustav neće dopustiti prijavu zato što se pri provjeri unesene lozinke ta raspršena vrijednost ponovno raspršuje što znači da neće biti jednaka točnoj vrijednosti. BCryptPasswordEncoder ima zadanu jačinu enkripcije od 10, ali preporučeno je podesiti jačinu tako da validacija lozinke traje 1 sekundu u slučaju da traje kraće s trenutnom vrijednošću.[37].

4.2.6 UPRAVLJANJE GREŠKAMA NA SERVERSKOJ STRANI

Skoro svaka operacija na serverskoj strani zahtjeva neke parametre. Ti parametri mogu biti ID brojevi resursa, datum i vrijeme kreiranja, broj ocjena, itd. Iz tog razloga važno je imati sistem validacije zahtjeva kako bismo mogli provjeriti valjanost zahtjeva i vratiti točnu poruku korisniku u slučaju da nije valjan. Game Developer Forum aplikacija ima standardizirani način odgovaranja na klijentske zahtjeve. Pod standardiziranim se misli na strukturu odgovora. Svaki HTTP odgovor je strukturiran na način da uvijek imamo ista polja u korijenskom JSON objektu. Na ovaj način klijent lako može provjeriti status operacije koju je pozvao. Struktura odgovora je prikazana u slijedećem dijelu koda na slikama 10 i 11.



Slika 10 Primjer serverskog odgovora za ispravan zahtjev

Slika 11 Primjer serverskog odgovora za neispravan zahtjev

U primjeru se vide dvije vrste odgovora. Slika 10 prikazuje odgovor kada je zahtjev validan i željena akcija je uspješno izvršena. U tom slučaju polje „data“ će biti popunjeno s povratnim podatkom, a ostala dva polja koja su vezana za grešku će biti prazna tj. postavljena na null vrijednost. U drugom slučaju na slici 11 vidljivo je da je došlo do greške kod izvođenja akcije. Razlog tomu može biti neispravan odgovor ili neuspješno izvršavanje akcije. Tada će polje za povratni podatak biti prazno, a polja za grešku i kod greške će imati vrijednost. Na ovaj način klijent lako može provjeriti ako se dogodila greška pri pokretanju neke akcije i može obavijestiti korisnika o grešci.

Mapiranje svakog odgovora u ovaj format nije nešto što je poželjno raditi ručno. Srećom Spring omogućuje odrađivanje ovog procesa automatski za svaki odgovor. Moguće je konfigurirati presretanje odgovora prije nego se pošalje klijentu, što je i napravljeno u ResponseAdvice klasi. Ovu funkcionalnost nam omogućava sučelje ResponseBodyAdvice koje sadrži metodu beforeBodyWrite.[38]. Navedena metoda može poslužiti kako bismo modificirali izlazni objekt koji će se serializirati i poslati klijentu. Prije slanja svakog odgovora, nadjačana metoda u ResponseAdvice klasi će mapirati odgovor u ranije spomenuti format podatkovnim poljem i poljima koji opisuju grešku. To će napraviti tako što će provjeriti radi li se o odgovoru koji sadrži grešku i ovisno o tome će popuniti odgovarajuća polja u izlaznom formatu.

Sve očekivane greške u aplikaciji imaju svoju klasu kako bi se mogle razlikovati različite vrste grešaka. Konfigurirani su detektori grešaka u AppliationAdvice klasi koji će „slušati“ dok ne primijete da je bačena određena greška. Ako se to dogodi vratit će odgovarajući zajedno s objektom koji predstavlja grešku. Taj privremeni objekt će se mapirati u ranije spomenuti format odgovora prije nego se pošalje klijentu. Dakle, služi nam samo kao privremeni podatkovni kontejner.

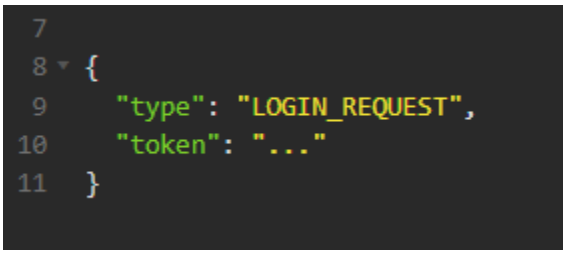
4.2.7 CHAT SERVER

Chat server je najzanimljiviji dio aplikacije. Chat u realnom vremenu u stilu je klasičnih IRC servera. Inspiracija za ovu funkciju dolazi od starije web stranice koja je također izrađena za jednu zajednicu koja se bavi izradom softvera. Game Developer Forum je prvenstveno osmišljen kao platforma za asinkronu komunikaciju. Chat je skroz novi koncept u usporedbi s dosadašnjim funkcijama platforme. Ova funkcija je dodana iz razloga što će članovi moći izabrati način komunikacije. Mogu birati između sporog asinkronog načina ili mogu komunicirati preko jednog sinkronog kanala. Upravljanje sinkronom komunikacijom je uvijek izazov, posebno kada servis koristi veći broj ljudi. Potrebno je voditi računa o više stvari. Potrebno je bilježiti stanje svih klijenata u svakom trenutku kako bismo znali jesu li već spojeni korisnici još uvijek spojeni ili je veza prekinuta. Obično se ovo izvodi pomoću metode otkucaja (eng. Heartbeating). Metoda otkucaja se bazira na periodičkom slanju signala između klijenta i servera kako bi se utvrdilo da veza nije prekinuta i da se klijent nije odspojio. Nakon određenog vremena server zahtijeva da mu klijent pošalje signal koji označava da je klijent aktivno spojen i veza nije prekinuta. Ako klijent ne odgovori u zadanom vremenu, server će prekinuti vezu i maknuti klijenta sa popisa spojenih.

U ovom slučaju nije bilo potrebno implementirati tu funkcionalnost iz razloga što alat koji je korišten već ima sličnu funkciju detektiranja prekida veze između krajnjih točaka (eng. sockets). Najveći izazov kod razvijanja chat kanala je bilo razvijanje servera za pravovremenu komunikaciju i definiranje protokola (tj. formata podataka) koji će se koristiti za komunikaciju između servera i klijenta. Jedan takav server mora imati slijedeće mogućnosti:

- mora podržavati autentikaciju,
- bilježiti tko je spojen na server,
- učiniti listu spojenih korisnika dostupnom ostalim korisnicima,
- distribuirati poruke svim spojenim korisnicima,
- pratiti stanje korisnika,
- detektirati prekid veze bilo kojeg korisnika.

Protokol za komunikaciju je vrlo kritičan element ovog sistema jer će se na njemu bazirati sva buduća komunikacija. Jedan od zahtjeva za protokol je to da je modularan zbog mogućnosti budućeg proširenja. Baziran je na zahtjev-odgovor obrascu razmjene poruka koje se šalju i primaju u JSON formatu u obliku paketa. U kontekstu ove aplikacije paket je jedna podatkovna jedinica koja služi za razmjenu podataka sa gateway serverom za pravovremenu komunikaciju. Svaki paket sadrži polje koje definira tip paketa. Ostala polja koja će se pojaviti u paketu ovise o tipu paketa. Svaki tip paketa ima svoju svrhu. Na primjer, paket tipa „LOGIN_REQUEST“ služi za zahtijevanje autentikacije. U nastavku je prikazan primjer autentikacijskog paketa i odgovora na zahtjev od strane servera na slici 12 i 13.



Slika 12 Primjer podatkovnog chat paketa za login zahtjev

Slika 13 Primjer podatkovnog chat paketa za login odgovor

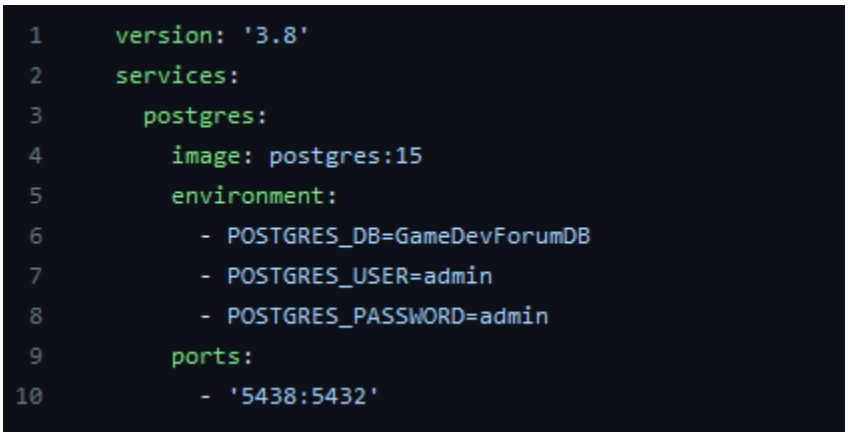
Proces spajanja i slanja poruke na gateway se odvija na slijedeći način. Klijent se želi povezati na gateway server kako bi primao i slao poruke u kanal. Generira podatkovni paket koji je tipa „LOGIN_REQUEST“ i postavlja mu sva potrebna polja. U ovom slučaju to je jedno jedino polje pod nazivom „token“. Vrijednost polja se postavlja na JWT token koji je spremljen u lokalnom spremištu preglednika klijenta. Nakon toga paket se serializira u „sirovi“ JSON format i šalje na gateway server. Server će tada primiti i deserializirati paket u JSON format. Dalje, validirati će paket tako što će pročitati tip paketa i provjeriti postoje li sva potrebna polja za taj tip paketa. U slučaju da paket ne prođe validacijski proces ili nije valjan JSON objekt, šalje se paket tipa „ERROR“ koji sadrži poruku greške. U protivnom, ako validacija uspješno prođe, server šalje odgovor (pogledati primjer naveden na slici 13) klijentu kako bi mu dao do znanja da je autentikacija prošla. Klijent je u ovom trenutku spojen na server i dobivat će sve odaslane poruke od servera. Nakon spajanja, klijent dobiva listu svih trenutno spojenih korisnika. Prilikom spajanja novih korisnika ili odspajanja trenutnih korisnika, šalje se podatkovni paket koji obavještava sve klijente da se korisnik spojio ili odspojio. Klijent koji primi ovaj podatkovni paket može sam promijeniti svoje stanje i maknuti ili dodati korisnika na listu koja se nalazi na sučelju te ispisati poruku o promjeni u kanal. Slanje i primanje poruka radi na isti princip. Prilikom slanja poruke šalje se podatkovni paket posebnog tipa, a poruke se šalju u dodatnom polju. Server prima paket i šalje poruku svim klijentima. Klijent koji je poslao poruku je također uključen iz razloga što na taj način možemo znati da li je poruka doista primljena od strane servera. Ako se poruka pojavi u chat kanalu znači da je server uspješno primio i prenio poruku svim korisnicima. Prilikom prekida veze server šalje obavijest ostalim korisnicima i briše odspojenog klijenta s liste spojenih korisnika.

4.2.8 DOCKER

Ovo poglavlje opisuje pokretanja aplikacije na način da se minimalizira potrebna konfiguracija. Pokretanje aplikacija baziranih na Java programskom jeziku se inače izvodi tako što se kompatibilna i zapakirana aplikacija pozove preko JRE (Java Runtime Environment). Spring Boot je naposljetku samo Java aplikacija i može se pokrenuti klasičnim putem. Međutim, u slučaju Game Developer Forum aplikacije za rad potrebna je i PostgreSQL baza podataka. Dakle, ova aplikacija zahtjeva neku vanjsku zavidnost. Kao i ostale stvari u ovom radu, to se može napraviti na više načina. U ovom slučaju korišten je Docker alat.

Problem kod pokretanja aplikacije je to što su im potrebne određene okoline i vanjske zavisnosti bez kojih ne mogu raditi. U ovom slučaju potreban je JRE bez kojeg se spakirana Java aplikacija ne može pokrenuti. Ovdje dolazimo do Docker alata. Docker je alat koji služi za virtualizaciju aplikacija i upravljanje virtualiziranim aplikacijama. Aplikacije se pakiraju u tzv. kontejnere koji se mogu konfigurirati i izraditi sa željenim zavisnostima. Kontejner je jedinica koja se sastoji od aplikacije i njezinih zavisnosti.[39] Kontejner se instancira prema nacrtu, to jest prema virtualnoj slici koja sadrži sve što je potrebno kako bi aplikacija mogla raditi.[39]. Korisnik sam može izraditi virtualnu sliku prema vlastitoj želji ili može preuzeti jednu od milijun već gotovih slika s javnog Docker repozitorija. U ovom projektu korišten je malo drukčiji način pokretanja koji uključuje docker-compose. Docker compose je alat u sklopu Dockera preko kojeg možemo definirati servise. Servis možemo zamisliti kao jednu logičku cjelinu ili proces koji radi odvojeno od ostalih definiranih procesa. Servis može biti aplikacija kao Game Developer Forum ili baza podataka. Definirana je docker-compose skripta koja će pokrenuti servis s PostgreSQL bazom i otkriti pravilni port kako bi omogućio vanjskim aplikacijama spajanje na bazu podataka. U skripti se koristi već izgrađena slika PostgreSQL baze koja je dostupna preko Docker repozitorija.

Naravno, Docker nije jedino rješenje za ovaj problem. Aplikacija se mogla spakirati u JAR datoteku i pokrenuti na klasični način (pr. preko portabilne verzije JRE). Baza se također mogla instalirati i pokrenuti direktno na operacijskom sustavu umjesto u Docker kontejneru. Također, obje aplikacije bi se trebale postaviti na zasebne servere te spojiti preko mreže. U nastavku na slici 14 vidi se docker compose skripta koja je korištena za pokretanje PostgreSQL baze kao servis. Također, postavljaju se neke varijable i otkriva se potrebni port okolini kako bi se mogli spojiti na bazu.



Slika 14 Docker compose skripta za pokretanje Postgre baze

4.3 IZRADA KORISNIČKOG SUČELJA (FRONTEND)

U ovom poglavlju opisuje se izrada korisničkog web sučelja koristeći ReactJS frontend biblioteku, Tailwind CSS razvojni okvir i još nekoliko pomoćnih alata. Dotaknuti će se i razlozi zbog kojih se danas koriste korisnička razvojna okruženja i gotove komponente, vidjeti koji se programski jezik koristi za pisanje frontend koda, i kratko opisati DOM te ReactJS virtualni DOM.

4.3.1 RAZVOJNI OKVIRI ZA KLIJENTSKI DIO APLIKACIJE GAME DEVELOPER FORUM

U slučaju Game developer foruma koristi se Tiller razvojni okvir koji je nedavno otvoren zajednici kao open source projekt. Tiller je okvir koji donosi veliki broj visoko kvalitetnih komponenti s profesionalnim izgledom. Cilj ovog alata je ubrzati proces izrade web korisničkih sučelja. Baziran je na ReactJS što znači da se može koristiti u bilo kojem ReactJS projektu. Komponente su osmišljene na način da podržavaju, ali zahtijevaju minimalnu konfiguraciju tj. rade „out of the box“. Ranije je spomenuto kako se za razvoj web stranica klasičnim putem koristi JavaScript. Problem kod korištenja JavaScript-a za izradu kompleksnijih aplikacija je to što nema potrebni stupanj strukturiranosti koda u smislu definiranja tipova podataka. Typescript rješava taj problem. Typescript je ekstenzija JavaScript jezika s dodatnim bonusom. Koristeći Typescript možemo pisati JavaScript kod sa strogo definiranim tipovima kao u statički pisanim jezicima. Dodaje potporu za definiranje tipa varijabli, povratnih tipova, tipova definiranih od strane korisnika, sučelja, itd. Naravno, internetski preglednici ne razumiju Typescript kod što znači da se ne može direktno koristiti u finalnoj verziji aplikacije. Typescript kod se na kraju procesa razvoja kompajlira u ekvivalentan JavaScript kod koji internetski preglednik može razumjeti. Kompajler je pri kompiliranju Typescript koda provjerio napisani kod i detektirao potencijalne greške (npr. kod postavljanja vrijednosti varijabli).

Dakle Typescript služi kao pomagalo stvaratelju pri pisanju koda a zatim se pretvara u obični JavaScript kako bi taj kod bio razumljiv internetskom pregledniku. U nastavku će se

pogledati dublje u principe rada u ReactJS koji je korišten prilikom izrade Game developer forum aplikacije zbog svojih brojnih i korisnih funkcionalnosti.

React koristi nešto što se zove virtualni DOM kako bi efikasno izveo promjene elemenata na web stranici. Virtualni DOM je kopija originalnog DOM-a koja se nalazi u memoriji i sinkronizira se s originalnim DOM-om kada je to potrebno. Kada nastane promjena nekog elementa, React će usporediti trenutno stanje virtualnog DOM-a s njegovim prethodnim stanjem i na originalnom DOM-u će promijeniti samo one elemente koji su se doista promijenili umjesto da se cijeli DOM ponovno nacrta na ekran.[40].

Za izradu frontend dijela aplikacije je također korišteno nekoliko vanjskih modula kao što su http-proxy-middleware za preusmjeravanje HTTP zahtjeva do serverske strane, fetch-intercept za presretanje HTTP zahtjeva i odgovora, moment za rad s datumima i vremenom, react-avatar za generiranje korisničkih avatara, react-router za globalnu navigaciju u aplikaciji, formik za validaciju korisničkog unosa i yup za kreiranje sheme za validaciju korisničkog unosa.

4.3.2 CSS RAZVOJNI OKVIR

Game developer forum aplikacije koristi TailwindCSS kao svoj CSS razvojni okvir. U prethodnim poglavljima, kada je bilo riječ o raznim frontend razvojnim okvirima i okvirima za gotove komponente, spomenuto je kako Game developer forum koristi Tiller, razvojni okvir s visoko kvalitetnim redefiniranim komponentama. Tiller kao razvojni okvir interno koristi TailwindCSS kao svoj CSS razvojni okvir, pa je to razlog što ova aplikacija također koristi TailwindCSS. Međutim, isti rezultat se mogao postići s bilo kojim CSS okvirom.

Tailwind omogućava brzo i lako stiliziranje elemenata na web stranici koristeći svoje brojne predefinirane klase. Može se koristiti u projektima koji se rade klasičnim putem (bez frontend frameworka) tako što se klase mogu postaviti u class atribut elemenata a može se koristiti i u kombinaciji s nekim frontend okvirom ili bibliotekom kao što je ReactJS (koristeći className atribut u tom slučaju). U Game Developer Forum aplikaciji se koristi za stiliziranje komponenti, responzivnost ekrana i raspored komponenti po ekranima. Tailwind dolazi s hrpom pomoćnih klasa za pozicioniranje, bojanje pozadine i teksta, responzivnost, detekciju stanja itd. Responzivni dizajn se postiže na način da se klasama doda jedan od predefiniranih prefiksa koji označavaju određene veličine ekrana uređaja. U slučaju da zadane veličine ekrana nisu dovoljne mogu se definirati i vlastite vrijednosti, što je i napravljeno u Game Developer Forum aplikaciji na navigacijskoj traci. U nastavku slijedi jedan primjer responzivnog dizajna iz same aplikacije gdje je kontejner kartica koji zauzima punu širinu svog roditelja dok je veličina ekrana ispod ***md*** prijelomne točke, nakon čega poprima veličinu od jedne trećine svog roditelja.

```
<div className="flex flex-col space-y-10 w-full md:w-1/3">
    <StatisticCard
        statisticName="Popular topics"
        statistics={topCategories.map((category) => {
            return {
            name: category.title,
            value:
                topCategoriesDetails.length > 0
                ? `${
                    topCategoriesDetails.filter(
                        (details) => details.categoryId === category.id
                    )[0].threadCount
                    } threads`
                : `0 threads`,
            clickable: true,
            redirectLink: `/forum/${category.id}`,
            };
        })}
    />
    ...
</div>
```

Slika 15 Primjer responzivnog dizajna koristeći TailwindCSS

Responzivni dizajn je važan iz razloga što osigurava da će naša web stranica izgledati dobro na svim veličinama ekrana. Naravno kako bismo održali preglednost stranice na svim ekranima mora doći do nekih promjena u izgledu. Kontejneri koji su na većim ekranima orijentirani horizontalno će najvjerojatnije biti orijentirani vertikalno na manjim ekranima zbog manje slobodnog mjesta na x osi. To je u redu zbog toga što web preglednici imaju mogućnost listanja. To jest možemo pomicati svoje vidno polje (viewport) po y osi i time nadoknaditi izgubljeni prostor na x osi. U primjeru se vidi kako se može postaviti zadana veličina i veličina koja će nadjačati zadanu kada je potrebno. Klasa w-full postavlja veličinu elementa na punu veličinu svog roditelja sve dok se ne aktivira prijelomna točka md. Nakon aktivacije prijelomne točke veličina elementa će se promijeniti u w-1/3, tj. jednu trećinu roditelja. Osim predefiniranih vrijednosti za prijelomne točke, TailwindCSS podržava konfiguraciju i proširenje svojih zadanih vrijednosti te dodavanje novih korisnički definiranih vrijednosti za boje, prijelomne točke, itd. Dodavanje novih prijelomnih točaka je vrlo korisna opcija koja je poslužila prilikom razvitka Game Developer Foruma. U nekim slučajevima predefinirane točke jednostavno nisu dovoljno precizne.

U konačnici zaključujemo kako se korištenjem CSS frameworka može brže i lakše dobiti željeni izgled web stranice i određeni stupanj responzivnosti. Često korištene vrijednosti atributa su izdvojene u zasebne klase koje korisnik može iskoristiti na elementima što čini razvoj bržim. Također sav CSS kod se nalazi na istom mjestu kao HTM kod što pridonosi jednostavnosti razvoja.

4.3.3 POVEZIVANJE KLIJENTSKE I SERVERSKE STRANE

Nakon izrade grafičkog korisničkog sučelja potrebno je spojiti isto na serversku stranu kako bi korisnik mogao komunicirati s raznim servisima koji su mu na raspolaganju. Kao i kod ostalih stvari postoji više načina na koje se ovo može izvesti. Na raspolaganju su dostupni razni alati kao što je Axios koji je klijent za slanje HTTP zahtjeva. Unatoč tome što je puno lakše koristiti alat za slanje HTTP zahtjeva, Game Developer Forum koristi ugrađenu fetch funkciju JavaScript jezika čija je zadaća slanje HTTP zahtjeva i vraćanje odgovora u obliku JavaScript obećanja (eng. Promise). Može se reći da je to jako primitivan način slanja HTTP zahtjeva u odnosu na druge opcije, ali postoji direktna kontrola nad svime što je potrebno.

Jedan od najvećih problema koji se pojavio prilikom izrade klijentske strane ove aplikacije je vezan za sigurnosne protokole modernih internetskih preglednika. Ako ste ikada razvijali full-stack aplikaciju, najvjerojatnije ste se već susreli sa CORS. CORS je sigurnosni mehanizam koji je dodan u moderne web preglednike u svrhu sprječavanja Cross-site scripting (XSS) i Cross-site Request Forgery (CSRF) napada. Ovaj mehanizam radi na principu polja u zaglavlju HTTP odgovora i dopušta samo HTTP zahtjeve na bazi istog podrijetla (domena, shema ili port)[41]. Prilikom slanja odgovora od strane servera prema klijentu, server će u zaglavlje odgovora uključiti popis podrijetla iz kojih klijent smije povući neke resurse.[41]. Prilikom razvoja klijentskog dijela došlo je do problema gdje CORS mehanizam nije dopuštao klijentskom dijelu pristup serverskoj strani iz razloga što serverska strana nije u istoj domeni kao i klijentska aplikacija.

Prije rasprave o tome kako riješiti ovaj problem važno je znati da se CORS mehanizam odvija samo u web preglednicima, ne na serverima. Ta karakteristika je sastavni dio rješenja koje je korišteno. Dakle, pregledniku nije dozvoljeno zatražiti resurse iz drugog podrijetla, ali server nema tu zabranu. Može se iskoristiti server kako bi se slali HTTP zahtjevi i primali odgovori. Upravo to je i napravljeno za klijentskoj strani. Konfiguriran je proxy server koji će klijentske zahtjeve poslati do server strane aplikacije.[42]. Ovaj server se ponaša kao posrednik između klijentske strane (korisničkog sučelja) i serverske strane. Prisjetimo se prošlog dijela ovog poglavlja kako bismo shvatili zašto ovaj način radi, a prošli način nije radio. CORS mehanizam dozvoljava HTTP zahtjeve iz istog podrijetla. U pravilu slanje zahtjeva s klijentske strane prema proxy serveru bi trebalo biti dozvoljeno iz razloga što je proxy server podignut na istom mjestu gdje je i klijentska aplikacija. To znači da će zahtjev proći do proxy servera koji nema CORS restrikcije i zato će moći poslati zahtjev do serverske strane bez problema.[42].

4.3.4 OMATANJE ODGOVORA

Svaki zahtjev i svaki odgovor koji aplikacija mora procesirati predstavljen je u obliku Typescript tipa u zasebnoj datoteci. Na taj način možemo mapirati zahtjeve i odgovore u objekte u svrhu lakše manipulacije s podacima. Prisjetimo se kako serverska strana vraća odgovor koji je standardiziran na razini ove aplikacije. Taj format se također mora reflektirati na klijentskoj strani.

Pitanje je kako primijeniti jedan format odgovora na sve moguće odgovore koji se mogu primiti od strane servera. Ovaj problem ima jednostavno rješenje - zbog toga što se koristi Typescript postoji mogućnost definiranja generičkih tipova. Koristeći ovaj mehanizam možemo definirati bazni tip podatka koji će svojom strukturom predstavljati ranije spomenuti format koji se vraća sa serverske strane. Pogledom u BaseResponseWrapper datoteku vidi se da je definiran tip podatka koji ima ista polja kao i serverski format i odgovara s još nekoliko dodatnih pomoćnih polja. Također vidimo da je ovaj tip podatka generičan što znači da se može koristiti u kombinaciji s ostalim tipovima (odgovorima). U nastavku na slici 16 vidi se primjer mapiranja primljenog odgovora u standardizirani format i popunjavanje pomoćnih polja.

```
6  export async function postSearchSectionRequest(
7    request: SearchSectionRequest
8  ): Promise<BaseResponseWrapper<SectionResponse[]>> {
9    const response = await fetch(SECTION_SEARCH_URL, {
10     method: "POST",
11     headers: {
12       "Content-Type": "application/json",
13     },
14     body: JSON.stringify(request),
15   });
16
17   return {
18     ...(await response.json()),
19     status: response.status,
20     isOk: response.ok,
21   } as BaseResponseWrapper<SectionResponse[]>;
22 }
```

Slika 16 Primjer mapiranja serverskog odgovora u bazni odgovor

Prva linija u izjavi povratka čita sva polja koja je server poslao do klijentske strane, dok druga i treća linija spremaju stanje odgovora u pomoćna polja koja će poslužiti za brzu provjeru stanja HTTP odgovora. Također vidimo kako se koristi mehanizam generičkih tipova podataka tako što tipu BaseResponseWrapper dajemo tip SectionResponse, što znači da će bazni objekt sadržavati niz SectionResponse objekata u svom podatkovnom polju ako ne dođe do greške pri izvršavanju željene akcije.

Osim baznog tipa za odgovore postoji još jedan generički tip podatka koji je definiran u svrhu mapiranja odgovara, a to je PageResponseWrapper. Radi se o objektu koji pomaže s mapiranjem paginiranih odgovora. Sadrži polje koje služi za pristup paginiranim podacima i još nekoliko polja koja govore koliko ukupno elemenata postoji, koliko stranica postoji, broj stranice i veličina stranice. Za razliku od BaseResponseWrapper, PageResponseWrapper se ne koristi kao

korijenski objekt za mapiranje odgovora. Namijenjen je da se koristi u kombinaciji s baznim objektom tako što će bazni objekt sadržavati objekt za paginaciju.

4.3.5 UPRAVLJANJE GREŠKAMA NA KLIJENTSKOJ STRANI

Ovo poglavlje je nastavak na prethodno poglavlje gdje se govorilo o mapiranju odgovora u određeni format koristeći mehanizam generičkih tipova podataka. Prisjetimo se poglavlja o upravljanju greškama na serverskoj strani gdje smo rekli da server omata odgovor u određeni standardizirani format te u slučaju da da dođe do greške postavlja vrijednosti za dva polja koja opisuju tu grešku. Ta polja su poruke greške i kod greške. Ova polja se koriste kako bi se moglo precizno odrediti o kojoj vrsti greške se radi i zašto je nastala te obavijestiti korisnika na odgovarajući način. Upravljanje greškama se događa na sličan način na klijentskoj strani. Primjer gdje se ovo koristi je na login stranici. Nakon što korisnik unese korisničko ime i lozinku, klijentska aplikacija šalje zahtjev prema serveru s unesenim podacima koji se zatim provjeravaju. Ako podaci nisu točni server će klijentu vratiti poruku koja sadrži kod greške koji odgovara grešci za krivi unos korisničkih detalja. Klijent zatim zaprima odgovor i provjerava radi li se o grešci ili ne. Provjerava polja i otkriva da je server vratio grešku. Zatim će usporediti kod greške s lokalno definiranim kodovima za greške i korisnika obavijestiti da korisnički detalji nisu točni.

4.3.6 SIGURNOST NA KLIJENTSKOJ STRANI

Sada kada je problem s CORS riješen može se vratiti na slanje zahtjeva i arhitekturu klijentskog dijela. Također će se dotaknuti autorizacija s JWT tokenima, omatanje odgovora i presretanja HTTP prometa. Dobro postavljeni sigurnosni mehanizmi na serverskoj strani su dovoljni kako bi se spriječilo pokretanje neželjenih operacija od strane neautoriziranih korisnika, ali potrebno je isto reflektirati i na klijentskoj strani u smislu da korisnik vidi što smije, a što ne smije napraviti. Ovo je moguće napraviti zbog posebne arhitekture aplikacije koja daje mogućnost autorizacije sa serverom i spremanje informacije o trenutno autoriziranom korisniku. Autentikacija i autorizacija na klijentskoj strani rade na slijedeći način. Korisnik će unijeti korisničko ime i lozinku nakon čega će klijentska strana poslati HTTP zahtjev na predefiniranu rutu na serverskoj strani. Server će zatim provjeriti podatke, generirati JWT token i klijentskoj strani poslati odgovor koji sadrži novo generirani token. Token će se tada spremiti u lokalno spremište web preglednika na klijentskoj strani i koristiti za autorizaciju budućih HTTP zahtjeva.

Sada dolazi ključni dio sistema za autorizaciju, a to je slanje tokena serveru. Kao što je već spomenuto, u svaki zahtjev treba priložiti JWT token kako bi server odobrio zahtjev. Umjesto da na svakom pozivu fetch funkcije priložimo token u zaglavlju, možemo to odraditi na puno efikasniji način s minimalnim dupliciranjem koda. Korištenjem vanjskog alata možemo definirati, to jest registrirati funkcije za presretanje HTTP zahtjeva ili odgovora. Upravo je to i napravljeno u glavnoj komponenti aplikacije. Registrirana je funkcija koje će presresti sve HTTP zahtjeve i priložiti JWT token koji je spremljen u lokalnom spremištu (ako postoji). U slučaju da token ne postoji u lokalnom spremištu, polje u zaglavlju će ostati prazno i server će vratiti prikladni statusni kod (403) što će javiti klijentskoj strani da korisnik nije prijavljen i da ga treba preusmjeriti na

login stranicu. Isto će se dogoditi ako je JWT token istekao. Presretanje HTTP odgovora radi na isti način, ali ta funkcija ima drugu svrhu. Funkcija koja presreće HTTP odgovore služi za detekciju grešaka od strane servera, ne autoriziranih operacija i slično. Registrirana funkcija će zaprimiti odgovor i provjeriti statusni kod kako bi znala da li se dogodila greška. Na primjer, u slučaju da se dogodi interna greška na serverskoj strani i vrati se predefinirani statusni kod za tu grešku (kod 500), korisnik će biti preusmjeren na stranicu koja korisnika upućuje na tu informaciju.

Korisnik mora znati što smije a što ne smije raditi na platformi tj. aplikacija to treba znati kako bi sakrila ili otkrila određene kontrole sučelja ovisno o tome treba li ih korisnik vidjeti. To ovisi u korisničkoj ulozi u sustavu. Svakom korisniku je dodijeljena zadana uloga „USER“ koja ima osnovna prava kao što su otvaranje rasprave, pisanje poruka, uređivanje vlastitog sadržaja, itd. Također postoje korisnici koji imaju „ADMIN“ ulogu. Osim osnovnih prava, oni imaju još neka dodatna prava koja su im potrebna kako bi se održao red na platformi. Dakle, potrebno je na neki način odvojiti perspektivu normalnog korisnika i administratora. U ovoj aplikaciji to radimo koristeći kondicionalno prikazivanje komponenti. Kondicionalno prikazivanje komponenti znači da se komponente mogu sakriti ili prikazati ovisno o nekom uvjetu. Pri inicijalnom učitavaju aplikacije u web pregledniku provjerava se postoji li spremljeni JWT token. Ako postoji, serverskoj strani se šalje HTTP zahtjev s priloženim tokenom na predefiniranu rutu. Svrha ovog zahtjeva je provjera tokena i dobiti informacije o korisniku koji je serializira u tokenu. Server vraća DTO (eng. Data Transfer Object) objekt koji sadrži informacije o korisniku. Klijentska strana će procesirati odgovor i spremiti objekt korisnika koristeći posebnu komponentu zvanu AuthProvider. Zadaća ove komponente je pružiti ostatku aplikacije informaciju o trenutno prijavljenom korisniku te izlaže setter funkciju pomoću koje može postaviti korisnika. To se radi korištenjem ReactJS konteksta. ReactJS kontekst je mehanizam preko kojeg se može izbjeći nepotrebno prenošenje podataka od roditelja do potomaka koji su na dubljoj razini korištenjem prop parametara.[43]. Koristeći kontekst bilo koja komponenta koju omatamo s kontekstom može pristupiti podatcima u kontekstu s bilo koje razine tako što će se „pretplatiti“ na kontekst.[43]. Ekrani koji sadrže elemente koje samo određeni korisnici smiju koristiti će odraditi provjeru autoriteta prije prikaza takvih elemenata. To će odraditi na način da se pretplate na kontekst koji nam nudi AuthProvider komponenta i pristupe korisničkom objektu koji sadrži informacije o trenutkom korisniku. Pročitati će polje koje označava autoritet korisnika (ulogu) i ovisno o tome ima li dovoljna prava, prikazati će ili sakriti komponentu u pitanju.

4.3.7 NAVIGACIJA

Svaka web aplikacija koja se sastoji od većeg broja pod stranica bi trebala imati efikasni način navigacije između raznih pod stranica. Ova funkcionalnost se najčešće postiže dodavanjem navigacijske trake na vrh stranice ili po strani. Nakon postavljanja navigacijske trake potrebno je povezati navigacijske elemente s pod stranicama. To se može odraditi u obliku standardnih HTML linkova, ali u slučaju Game Developer Forum aplikacije korišten je vanjski alat. ReactRouter je vanjski modul koji omogućuje rutiranje između pod stranica na klijentskoj strani.[44]. Osim navigacije, ovaj modul donosi još nekoliko bonus funkcionalnosti. Iako se ne koriste, pomoću ReactRoutera mogu se lako dijeliti podatci između pod stranica pri navigaciji s jedne na drugu

stranicu. Isto tako može se pratiti povijest navigacije korisnika između različitih pod stranica. Korištenjem ovog vanjskom modula može se vrlo jednostavno definirati struktura naše web aplikacije i njezinih brojnih pod stranica. Navigacija na razini aplikacije se konfigurira na način da se u glavnu komponentu aplikacije doda posebna komponenta rutera koja će omotati glavne dijelove aplikacije. Ime komponente ovisi o tome koji ruter želimo koristiti. ReactRouter dolazi s više implementacija rutera koje možemo koristiti. U ovom slučaju korišten je BrowserRouter koji omata ostatak aplikacije. Nakon odabira rutera moramo definirati navigacijske rute za svaku pod stranicu i specificirati putanju za svaku od ruta. Ovaj dio se radi unutar Routes komponente koja dolazi s ReactRouter modulom. Definiramo Routes komponentu koja će služiti kao kontejner za sve naše rute do različitih pod stranica. Unutar Routes komponente definira se ruta za svaku pod stranicu. To radimo tako što ćemo koristiti još jednu komponentu zvanu Route. Route komponenta prima prop (parametar) path i home. Path pop predstavlja putanju do pod stranice, a element prop predstavlja glavnu komponentu pod stranice koja će se prikazati kada se preglednik preusmjeri na navedenu putanju. U slučaju ove aplikacije putanje su relativne u odnosu na podrijetlo što znači da ne moramo pisati punu putanju do određene pod stranice već možemo samo napisati slijedeću destinaciju od podrijetla (/home umjesto http://nekadomena.hr/home). Na slijedećoj slici vidimo primjer definiranja ruta koristeći BrowserRouter i Routes komponente. Prilikom navigacije na ***/home*** rutu, aplikacije će prikazati ***Home*** komponentu na ekranu. Također je moguće dodati i parametre u rutu koji se kasnije mogu pročitati u definiranom elementu kao što je to odrađeno za komponentu sobe za raspravu.

```
<BrowserRouter>
  <Routes>
    <Route path="/home" element={<Home />} />
    <Route path="/news/:threadId" element={<Thread />} />
    <Route path="/forum" element={<Forum />} />
    ...
  </Routes>
</BrowserRouter>
```

Slika 17 Definiranje ruta koristeći BrowserRouter i Routes komponente

5 UPRAVLJANJE VERZIJAMA PROJEKTA

Upravljanje verzijama projekta je proces praćenja ili bilježenja promjena koje nastaju u projektu u svrhu njihove evidencije. Bilježi se sve što se promijenilo, gdje, kada i tko je napravio promjenu. Vrlo je poželjno koristiti neki alat za upravljanje verzijama koda kako bismo imali potpunu evidenciju promjena svih datoteka u slučaju da nastane neka neželjena promjena. Upravljanje verzijama koda nam omogućuje vraćanje cijelog projekta ili samo pojedinih projektnih datoteka u prethodne verzije. Isto tako, pomaže kod rada na projektu gdje imamo više ljudi koji rade promjene istovremeno. Prilikom rada na velikom projektu gdje radi više ljudi jako je teško pa skoro i nemoguće održati sinkronizaciju između svih članova tima bez korištenja alata za upravljanje verzijama.

Game Developer Forum koristi Git alat za upravljanje verzijama koda i GitHub kao poslužitelja udaljenog repozitorija. Platforma se sastoji od dva zasebna GitHub repozitorija. U jednom repozitoriju je pohranjena serverska strana dok je u drugom pohranjena klijentska strana. U početku je postojala samo jedna grana (main) na kojoj su se direktno spremale sve tekuće promjene. Nakon što je sadržaj glavne grane dostigao određenu razinu kvalitete i stabilnosti, sve slijedeće promjene su napravljene u zasebnim granama kako bismo odvojili ono nestabilno od stabilnog dijela i imali bolju grupaciju promjena u slučaju da je potrebno vratiti projekt na prethodnu verziju. Kao primjer možemo uzeti chat. Prije početka razvoja chat funkcionalnosti napravljene su nove grane na oba repozitorija. Sve promjene vezane za chat funkcionalnost su spremane u novo kreirane grane i testirane dok se ne utvrdi razina stabilnosti i kvalitete. Nakon završetka rada na toj funkcionalnosti i prolaska kroz testnu fazu, na GitHub repozitoriju otvoren je zahtjev za spajanje grane u baznu granu. Promjene su zatim spojene u baznu granu. Isti proces je primijenjen za sve kasnije promjene s izuzetkom manjih dorada koje su direktno poslane na baznu granu.

6 RASPRAVA

Ova cjelina nije standardna rasprava već je isključivo usmjerena na problem skalabilnosti i mogućim nadogradnjama koje bi mogle unaprijediti aplikaciju. Kao moguće poboljšanje navedena je funkcionalnost koja omogućuje dijeljenje sadržaja na društvenim mrežama iz razloga što je to očekivana funkcionalnost u aplikacijama u novije vrijeme. Proći će se kroz koncept kako bi se vidjelo na koji način ta funkcija može biti implementirana u aplikaciju.

6.1 SKALABILNOST

Vjerojatno najveći problem na koji svi developeri većih web aplikacija i platforma nailaze jest skalabilnost. Uvijek se postavlja pitanje kako unaprijediti performanse aplikacija i povisiti broj mogućih korisnika u svakom trenutku. Ponekad nije dovoljno unaprijediti hardver na kojem se aplikacija izvršava već sami kod aplikacije. Kod koji je napisan na neefikasan način može dovesti do puno većih poteškoća nego što mislimo. Kao primjer možemo uzeti filtriranje podataka u bazama podataka. Query je filter koji se koristi za pronalazak tj. sužavanje izbora podataka u bazi podataka. Dohvat podataka u kojem se koristi nepravilno napisan query može potrajati duplo duže nego onaj s efikasno napisanim query izrazom. Game Developer Forum nije iznimka navedenom. Skaliranje aplikacije ovog tipa se najčešće vrši na način da se poveća broj servera (čvorova ili eng. node u mreži) koji mogu primiti i procesirati zahtjev i time povećati kapacitet procesiranja zahtjeva na razini platforme. Skup servera na kojima se vrti aplikacija se povezuje na glavni razvodni server koji će zahtjeve preusmjeravati na aplikativne servere u skupini.[45]. Stroj koji odrađuje tu zadaću se zove load balancer.[45]. To je specijalizirani dio opreme koji zahtjeve do aplikativnih servera prosljeđuje po nekoj konfiguriranoj strategiji.[45]. Jedna od mogući strategija prosljeđivanja je da se primljeni zahtjev proslijedi serverima po redu (Round Robin strategija).[45]. Na taj su se način

rasteretili strojevi tako što svatko dobije po jedan zahtjev umjesto da jedan stroj procesira sve zahtjeve. Također, postoje strategije za izvršavanje nadogradnje aplikativnih servera na način da korisnici ne primijete zastoj u servisu. Kako bi se to postiglo dodaje se još jedna skupina aplikativnih servera koja će u slučaju potrebne nadogradnje aplikativnog softvera prve skupine služiti kao rezerva kako bi usluga ostala na raspolaganju korisnicima za vrijeme nadogradnje prve skupine.[46]. S vremenom sva infrastruktura koja koristi instancu stare verzije aplikacije će dobiti novu verziju aplikacije.[46]. Ova strategija se naziva rolling update strategija.

Sličan, ako ne i veći, problem se javlja kod chat servera. Chat server se u slučaju Game Developer Forum-a izvodi kao dio aplikacije u pozadinskom procesu. Međutim, to bi u produkcijskoj okolini bilo vrlo nepoželjno zbog više razloga. Prvotno zbog toga što se ne mogu provesti dorade ili nadogradnje na jednoj aplikaciji bez da se naruši dostupnost druge. Još jedan problem su resursi stroja na kojem je aplikacija pokrenuta, koji se dijele između chat servera i Spring Boot aplikacije. Logovi su spojeni u jedan stream što isto nije poželjno kada se trebaju pregledati ili kada se treba debugirati neku funkcionalnost. Chat server bi trebao biti zasebna aplikacija pokrenuta na zasebnom stroju (stvarnom ili virtualnom). Što se tiče skalabilnosti chat servera, kao što je spomenuto ranije, ovdje dolazimo do istih problema s performansama aplikacije. Uz dovoljan broj povezanih klijenta, resursne potrebe gateway servera mogu lako prijeći one Spring Boot aplikacije iz razloga što gateway mora održavati veze između sebe i svih spojenih klijenta te uz to još i procesirati nadolazeće zahtjeve, a Spring Boot server ne mora održavati vezu na produljene periode vremena.

6.2 MOGUĆE NADOGRADNJE

Osnovna ideja za potencijalnu buduću nadogradnju platforme je u vidu integracije s društvenim mrežama. Većina web aplikacija danas ima neku vrstu integracije s društvenim mrežama. To je najčešće dijeljenje sadržaja s platforme na svoj društveni profil. Možemo reći da je to očekivana funkcionalnost u aplikacijama današnjeg doba. Takva integracija na Game Developer forum aplikaciji bi mogla funkcionirati na način da korisnici mogu dijeliti određeni sadržaj poput objava u sobama za raspravu i ponuda na tržištu (eng. Marketplace). Dijeljenje bi se odvijalo na način da korisnik pritisne gumb za dijeljenje koji se nalazi negdje u podnožju objave ili marketplace ponude nakon čega se preusmjerava na formu za objavu sadržaja na društvenoj mreži. Objava bi sadržavala unaprijed formatiran sadržaj koji korisnik može urediti po želji.

Još jedna ideja za proširenje aplikacije je dodavanje sustava za dijeljenje datoteka poput 3D modela, glazbe i zvukovnih efekata, skripti i slično. Korisnici bi mogli dijeliti datoteke preko objava u sobama za raspravljanje, privatnim porukama, pa čak i u chat kanalu. Naravno ova funkcija zahtjeva puno više planiranja. Potrebno je postaviti i konfigurirati server koji će služiti kao trajno spremište za prenesene datoteke. Jedna bonus funkcija koja bi se isto mogla dodati u sklopu ove funkcionalnosti je skeniranje datoteka u svrhu otkrivanja zlonamjernog sadržaja.

7 ZAKLJUČAK

U ovom radu opisan je kompletni proces izrade full stack web aplikacije Game Developer Forum koja prvenstveno služi za sinkronu i asinkronu komunikaciju između stvaratelja video igara. Aplikacija se sastoji od serverske strane koja je napisana u Java programskom jeziku koristeći Spring Boot alat. Spring Boot je korišten zbog olakšanja razvoja i uštede vremena. Za bazu podataka je izabrana relacijska baza PostgreSQL zbog relacijske prirode podataka koji se pojavljuju u aplikaciji. Baza podataka se pokreće preko Docker alata za virtualizaciju na način da se pokrene docker-compose skripta koja će kreirati servis za bazu i otvoriti potrebne portove. Klijentski dio tj. korisničko sučelje je izrađeno koristeći ReactJS biblioteku u kombinaciji s TailwindCSS okvirom za stiliziranje i Tiller okvirom za gotove komponente. Povezanost tj. razmjena poruka između serverske i klijentske strane je odrađena koristeći JSON datoteke. Klijentska strana šalje zahtjeve na REST API koji tada procesira zahtjev i ovisno o tome koja operacija je zatražena, pokreće stvaranje, modificiranje ili brisanje sadržaja. Chat server koristi web-socket protokol za pravovremenu komunikaciju s klijentskom stranom. Zbog sigurnosti se koriste JWT tokeni koji predstavljaju korisnika i njegova prava. Svakom korisniku su dodijeljena osnovna prava pri registraciji koja mu omogućuju obavljanje osnovnih funkcija kao stvaranje sadržaja. Administratorska uloga koju ima samo administrator omogućuje uređivanje i brisanje sadržaja ostalih korisnika zbog održavanja reda. Korisničko sučelje se automatski prilagođava ulozi korisnika na način da se korisniku prikazuju samo one kontrole kojima može pristupiti dok je ostalo skriveno od pogleda. Verzije projekta su upravljane primjenom Git alat u svrhu evidencije promjena nad kodom. Na kraju rada su dani prijedlozi za skalabilnost aplikacije i moguće unaprjeđenje u obliku integracije s društvenim mrežama. Predloženo je skaliranje aplikacije na način da se poveća količina dostupnih aplikativnih servera i doda zasebni server za chat server aplikaciju. Također su spomenute metode nadogradnje i raspoređivanja zahtjeva po aplikativnim serverima. Dijeljenje na društvenim mrežama je zamišljeno kao dodatni gumb na sučelju koji omogućuje direktno dijeljenje odabranog sadržaja na podržane društvene mreže.

8 LITERATURA

[1] Amazon Web Services, Inc., „What Is A LAMP stack“, *aws.amazon.com* [Online]. Dostupno: https://aws.amazon.com/what-is/lamp-stack [Pristupljeno: 21.6.2023]

[2] MongoDB, Inc., „What Is the MEAN Stack“, *mongodb.com* [Online]. Dostupno: https://www.mongodb.com/mean-stack [Pristupljeno: 21.6.2023]

[3] MongoDB, Inc., „MERN Stack Explained“, *mongodb.com* [Online]. Dostupno: https://www.mongodb.com/mern-stack [Pristupljeno: 21.6.2023]

[4] VMware, Inc., *spring.io* [Online]. Dostupno: https://spring.io/ [Pristupljeno: 20.5.2023]

[5] VMware, Inc., „Introduction to the Spring IoC Container and Beans“ *docs.spring.io* [Online]. Dostupno: https://docs.spring.io/spring-razvojni okvir/reference/core/beans/introduction.html [Pristupljeno: 20.6.2023]

[6] prashant_srivastava, „Difference between Spring and Spring Boot“, *geeksforgeeks.org* [Online]. Dostupno: https://www.geeksforgeeks.org/difference-between-spring-and-spring-boot/ [Pristupljeno: 3.7.2023]

[7] Amazon Web Services, Inc., „What Is a Graph Database*“, aws.amazon.com.* [Online]. Dostupno: https://aws.amazon.com/nosql/graph/ [Pristupljeno: 7.4.2023]

[8] MongoDB, Inc., „Relational vs. Non-Relational Databases“, *mongodb.com* [Online]. Dostupno: https://www.mongodb.com/compare/relational-vs-non-relational-databases [Pristupljeno: 4.7.2023]

[9] M. Schwarzmüller, „SQL vs NoSQL“, *academind.com* [Online]. Dostupno: https://academind.com/tutorials/sql-vs-nosql [Pristupljeno 5.7.2023]

[10] J. Patadiya, „Angular vs React vs Vue – Javascript Razvojni okvir You Need for Your Business“, *radixweb.com* [Online]. Dostupno: https://radixweb.com/blog/angular-vs-react-vs-vue [Pristupljeno: 3.7.2023]

[11] D. Herbert, “What is React.js? (Uses, Examples, & More),” *HubSpot*. 27.5.2022 [Online]. Dostupno: https://blog.hubspot.com/website/react-js. [Pristupljeno: 20.5.2023]

[12] Mozilla, „Introduction to the DOM“, *developer.mozilla.org* [Online]. Dostupno: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction [Pristupljeno: 26.5.2023]

[13] Mozilla, mozilla.org kontributori, „Pseudo-classes“, *developer.mozilla.org* [Online]. Dostupno: https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes [Pristupljeno 28.5.2023]

[14] P.t Hooda, „Comparison – Centralized, Decentralized and Distributed Systems“, *geeksforgeeks.org* [Online]. Dostupno: https://www.geeksforgeeks.org/comparison-centralized-decentralized-and-distributed-systems/ [Pristupljeno: 22.6.2023]

[15] S. Ingalls, „*The Client-Server Model“.* Pristupljeno: 21.6.2023. [Online Slika]. Dostupno: https://www.serverwatch.com/guides/client-server-model/

[16] M. Jones, Microsoft, J. Bradley, Ping identity, N. Sakimura, NRI, „Json Web Token (JWT)“, *Datatracker*, Sviban, 2015 [Online]. Dostupno: https://datatracker.ietf.org/doc/html/rfc7519 [Pristupljeno 25.5.2023]

[17] K. Brush., „RDBMS (relational database management system)“, *TechTarget*. 2019 [Online]. Dostupno: https://www.techtarget.com/searchdatamanagement/definition/RDBMS-relational-database-management-system [Pristupljeno: 21.5.2023]

[18] Liquibase Inc., „Design Your Liquibase Project“, docs.*liquibase.com* [Online]. Dostupno: https://docs.liquibase.com/start/design-liquibase-project.html [Pristupljeno: 3.7.2023]

[19] Liquibase Inc., „DATABASECHANGELOG table“, *docs.liquibase.com* [Online]. Dostupno: https://docs.liquibase.com/concepts/tracking-tables/databasechangelog-table.html [Pristupljeno: 3.7.2023]

[20] M. Folnovic, „Nrich – Simplify your Java Application Development“, *croz.net* 4.2.2022 [Online]. Dostupno: https://croz.net/news/nrich/, [Pristupljeno 6.4.2023]

[21] Maddy, „Spring Boot Architecture“, *Tech With Maddy*, 14.11.2021 [Online]. Dostupno: https://techwithmaddy.com/spring-boot-architecture [Pristupljeno 22.5.2023]

[22] VMware, Inc., „Spring Data JPA“, *spring.io* [Online]. Dostupno: https://spring.io/projects/spring-data-jpa [Pristupljeno: 3.7.2023]

[23] T. Darimont, C. Strobl, M. Paluch, J. Bryant, G. Turnquist, „Spring Data JPA – Reference Documentation“, *docs.spring.io* [Online]. Dostupno: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories.query-methods.details [Pristupljeno 4.7.2023]

[24] M. Tyson, „What is JPA? Introduction to Java Persistance“, *InfoWorld*. 20.5.2022 [Online]. Dostupno: https://www.infoworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html [Pristupljeno: 22.5.2023]

[25] Hibernate, „Idiomatic persistance“, *hibernate.org* [Online]. Dostupno: https://hibernate.org/orm/ [Pristupljeno: 22.5.2023]

[26] R. Rakhe, „Java Database Connectivity (JDBC)“, *Medium*, 31.5.2021 [Online]. Dostupno: https://rajasrakhe20.medium.com/java-database-connectivity-jdbc-7f148ad06286 [Pristupljeno 22.5.2023]

[27] Microsoft, „CQRS pattern“, *Microsoft* [Online]. Dostupno: https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs [Pristupljeno: 23.5.2023]

[28] S. Kargopolov, „@Lazy Annotation in Spring Boot“, *appsdeveloperblog.com* [Online]. Dostupno: https://www.appsdeveloperblog.com/lazy-annotation-in-spring-boot/ [Pristupljeno: 4.7.2023]

[29] R. T. Fielding, „Representational State Transfer (REST)“, *Donald Bren School of Information and Computer Sciences,* 2000 [Online]. Dostupno: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, [Pristupljeno: 24.5.2023]

[30] M. Jha, „POJO Class in Java“, *Scaler*, 9.5.2022, https://www.scaler.com/topics/pojo-class-in-java/, (pristupljeno 24.5.2023)

[31] dotslash_adwitiya, „How Jackson Dana Binding Works in Spring Boot?“, *geeksforgeeks.org* [Online]. Dostupno: https://www.geeksforgeeks.org/how-jackson-data-binding-works-in-spring-boot/ [Pristupljeno: 4.7.2023]

[32] Mozilla, mozilla.org kontributori, „Content-Type“, *developer.mozilla.org* [Online]. Dostupno: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type [Pristupljeno: 4.7.2023]

[33] Thorben, „Design Patterns Explained – Dependency Injection with code examples“, *Stackify*, 3.3.2023 [Online]. Dostupno: https://stackify.com/dependency-injection/ [Pristupljeno: 25.5.2023]

[34] R. Johnson *et al*, „Spring Framework Reference Documentation“, *docs.spring.io* [Online]. Dostupno: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html#beans [Pristupljeno 25.5.2023].

[35] R. Johnson *et al*, „Spring Framework Reference Documentation“, *docs.spring.io* [Online]. Dostupno: https://docs.spring.io/spring-framework/docs/2.5.x/reference/beans.html#beans-definition [Pristupljeno 25.5.2023].

[36] VMware, Inc., „Architecture“ *docs.spring.io* [Online]. Dostupno: https://docs.spring.io/spring-security/reference/servlet/architecture.html [Pristupljeno: 25.5.2023]

[37] VMware, Inc. „Password Storage“, *docs.spring.io* [Online]. Dostupno: https://docs.spring.io/spring-security/reference/features/authentication/password-storage.html#authentication-password-storage-bcrypt [Pristupljeno: 25.5.2023]

[38] A. Balu, „ResponseBodyAdvice<T> Implementation in SpringBoot“, *medium.com* [Online]. Dostupno: https://medium.com/thefreshwrites/responsebodyadvice-t-implementation-springboot-c8ca605542b7 [Pristupljeno: 4.7.2023]

[39] Docker Inc., „Use containers to Build, Share and Run your applications“, *docker.com* [Online]. Dostupno: https://www.docker.com/resources/what-container/ [Pristupljeno: 25.5.2023]

[40] I. Mojeed, „What is the virtual DOM in React?“, *LogRocket*, 16.8.2022 [Online]. Dostupno: https://blog.logrocket.com/virtual-dom-react/ [Pristupljeno: 27.5.2023]

[41] Mozilla, „Cross-Origin Resource Sharing (CORS)“, *developer.mozilla.org* [Online]. Dostupno: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS#examples_of_access_control_scenarios [Pristupljeno: 28.5.2023]

[42] Herman J. Radtke III, „Proxying API Request in Development“, *create-react-app.dev* [Online]. Dostupno: https://create-react-app.dev/docs/proxying-api-requests-in-development/ [Pristupljeno: 3.7.2023]

[43] Meta Open Source, „Passing Data Deeply With Context“, *react.dev* [Online]. Dostupno: https://react.dev/learn/passing-data-deeply-with-context, [Pristupljeno: 28.5.2023]

[44] Remix Software, Inc. „Client Side Routing“, *reactrouter.com* [Online]. Dostupno: https://reactrouter.com/en/main/start/overview [Pristupljeno: 29.5.2023]

[45] Amazon Web Services, Inc., „What Is Load Balancing“, *aws.amazon.com* [Online]. Dostupno: https://aws.amazon.com/what-is/load-balancing/ [Pristupljeno: 4.7.2023]

[46] B. Bost, „Overview of Deployment Options on AWS“, *docs.aws.amazon.com* [Online]. Dostupno: https://docs.aws.amazon.com/pdfs/whitepapers/latest/overview-deployment-options/overview-deployment-options.pdf#rolling-deployments [Pristupljeno: 4.7.2023]

[47] J. Maras, „Client-side web application conceptual model“. Pristupljeno: 3.7.2023. [Online Slika]. Dostupno: https://www.researchgate.net/figure/Client-side-web-application-conceptual-model_fig4_261045393

CREATION OF THE FULL-STACK WEB APPLICATION "GAME DEVELOPER FORUM" USING SPRING BOOT AND REACTJS

SUMMARY

This paper shows the creation of "Game Developer Forum" web applications through the full-stack development paradigm using Spring Boot for the server side and ReactJS for the client side. The solution is a platform for synchronous and asynchronous communication between video game developers. The application solution brings the possibility of asynchronous communication between users through so-called discussion rooms and synchronous communication in real time through the chat page. It also includes a few more bonus features that we will discuss in more detail in later chapters. The advantage of the Game Developer Forum application compared to other application solutions that solve a similar problem is that, despite the fact that the application can be used as a general forum, it is adapted for use by game developers. Of course, the solution is not perfect and there is room for improvement. At the end of the paper, we will go through some possible upgrades that could improve the experience of using the applications.

Keywords: Internet forum, full-stack, Spring Boot, ReactJS

9 POPIS SLIKA I TABLICA

Slika 1 Princip organizacije serversko-klijentske arhitekture [15] 8
Slika 2 Konceptualni dijagram Game Developer Forum aplikacije Izrađeno prema [47] 10
Slika 3 ERD dijagram modela baze podataka Game Developer Forum aplikacije 12
Slika 4 Kontrole kategorija poslova kada je korisnik administrator 13
Slika 5 Kontrole kategorija poslova kada korisnik nije administrator 14
Slika 6 Primjer admin kontrola nad objavama ostalih korisnika 14
Slika 7 Primjer Liquibase skripte 17
Slika 8 Primjer nadjačavanja metode brisanja 19
Slika 9 Primjer dekodiranog Json Web Tokena 25
Slika 10 Primjer serverskog odgovora za ispravan zahtjev 27
Slika 11 Primjer serverskog odgovora za neispravan zahtjev 27
Slika 12 Primjer podatkovnog chat paketa za login zahtjev 29
Slika 13 Primjer podatkovnog chat paketa za login odgovor 29
Slika 14 Docker compose skripta za pokretanje Postgre baze 31
Slika 15 Primjer responzivnog dizajna koristeći TailwindCSS 33
Slika 16 Primjer mapiranja serverskog odgovora u bazni odgovor 35
Slika 17 Definiranje ruta koristeći BrowserRouter i Routes komponente 38

Tablica 1 Ključne razlike između Spring i Spring Boot [6] 3
Tablica 2 Usporedba relacijskih i ne relacijskih baza podataka [8] 4
Tablica 3 Ključne razlike između relacijske baze, baze orijentirane prema dokumentima i Graf baze Izvor podataka: [9] 4
Tablica 4 Prikaz troslojne arhitekture aplikacije 9
Tablica 5 Prikaz korisničkih uloga i razina autorizacije 11