Arhitektura modernih web aplikacija

Cvetko Voćanec, Petar

VELEUČILIŠTE U BJELOVARU

PREDDIPLOMSKI STRUČNI STUDIJ RAČUNARSTVO

ARHITEKTURA MODERNIH WEB APLIKACIJA

Završni rad br. 12/RAČ/2022

Petar Cvetko Voćanec

Bjelovar, listopad 2022.

obrazac ZR - 001

Veleučilište u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Student: **Petar Cvetko Voćanec** JMBAG: **0314020610**

Naslov rada (tema): **Arhitektura modernih web aplikacija**

Područje: **Tehničke znanosti** Polje: **Računarstvo**

Grana: **Programsko inženjerstvo**

Mentor: **Tomislav Adamović, mag. ing. el.** zvanje: **viši predavač**

Članovi Povjerenstva za ocjenjivanje i obranu završnog rada:

1. **Ivan Sekovanić, mag. ing. inf. et comm. techn., predsjednik**
2. **Tomislav Adamović, mag. ing. el., mentor**
3. **Krunoslav Husak, dipl. ing. rač., član**

2. ZADATAK ZAVRŠNOG RADA BROJ: 12/RAČ/2022

U sklopu završnog rada potrebno je:

1. odabrati potrebne resurse i alate za izradu web aplikacije
2. razviti Web aplikaciju upotrebom Docker platforme i Nginx web poslužitelja
3. analizirati ulogu slojeva u troslojnoj arhitekturi web aplikacija
4. Modelirati podatke upotrebom Prisma alata i PostgreSQL baze podataka
5. izraditi komunikacijski sloj primjenom NestJS okvira
6. izraditi prezentacijski sloj primjenom Angular platforme i Tailwind CSS okvira

Datum: 31.08.2022. godine

Mentor: **Tomislav Adamović, mag. ing. el.**

1. UVOD

Aplikacije kojima se pristupa putem internetskog preglednika vrlo su korištena i robusna rješenja za probleme širokog dijela ljudske svakodnevice. Korisnici dnevno provedu sate na takvim aplikacijama obavljajući razne poslovne zadatke poput sastanaka, organizacije poslova i održavanja programskog koda na sustavima za verzioniranje. No, primjena internetskih aplikacija nije rezervirana samo za poslovni dio svakodnevice. Naime, putem internetskih aplikacija velika većina stanovnika dolazi do novih vijesti i spoznaja, razgovara s prijateljima i općenito konzumira raznoliki sadržaj kojeg na internetu svakim danom ima sve više.

Velika potreba za internetskim aplikacijama dovela je do velikog napretka istih. Naime, web aplikacije vremenom postaju arhitekturalno kompleksnije i zahtijevaju sve veći broj alata za ostvarenje dobrog programskog rješenja. Imajući to na umu, mnogi se razvojni programeri web aplikacija sve više koncentriraju na samu arhitekturu aplikacija koje razvijaju.

Potreba za internetskih aplikacijama, njihove prednosti i nedostaci, kao i sve što je potrebno za jednu modernu aplikaciju takve prirode, teme su razrađene u ovom radu.

2. PREDNOSTI I NEDOSTACI WEB APLIKACIJA

2.1. Prednosti internetskih aplikacija

U usporedbi s klasičnim lokalno instaliranim aplikacijama, internetske aplikacije su fleksibilne, lako dostupne, nije ih potrebno instalirati, nemaju potrebe za ažuriranjem i slično. Posljedica toga jest široka rasprostranjenost internetskih aplikacija.

2.1.1. Podrška za velik raspon uređaja

Jedno od ključnih rješenja koje internetske aplikacije nude je korištenje web preglednika za učitavanje web aplikacija. Ovakvo rješenje u velikom je dijelu uklonilo mnoge probleme s kojima su se suočavali programeri klasičnih lokalno instaliranih aplikacija. Iako su programski jezici poput Jave uklonili mnoge probleme koji nastaju zbog različitih kombinacija sklopovlja i operativnih sustava, internetske aplikacije dovele su do toga da današnji programeri web aplikacija ne moraju brinuti na kojoj će se platformi njihovo programsko rješenje pokretati.

Naravno, problem vizualne responzivnosti aplikacije na širokom spektru zaslona još uvijek ostaje jedan od većih problema svake vrste aplikacije koja je namijenjena velikom broju različitih uređaja.

2.1.2. Lakoća isporuke programskog rješenja

Veliki problem klasičnih aplikacija koje je potrebno instalirati je ažuriranje istih. Naime, navedeni problem je vrlo izražen u populaciji ljudi s niskim znanjem o računalima i načinu na koji aplikacije funkcioniraju. Zbog toga mnogi korisnici aplikacija ne znaju kako ih ažurirati pa samim time ne mogu dobiti najnovije sigurnosne zakrpe ili značajke. Ovaj problem vrlo je izražen kod starijih klasičnih aplikacija koje je trebalo ručno ažurirati kad god je bila dostupna nova verzija aplikacije.

Internetske aplikacije u ovom području briljiraju jer je njihovo ažuriranje i inicijalno postavljanje u velikoj većini slučajeva vrlo minimalno što omogućuje primjenu takvih aplikacija kod širokog spektra ljudi, neovisno o njihovim računalnim sposobnostima.

2.2. Nedostaci internetskih aplikacija

Internetske aplikacije nisu rješenje za sve vrste problema te tako imaju svoje nedostatke i ograničenosti. Ti su nedostaci u mnogim slučajevima rješivi dobrim programskim kodom i dobrom organizacijom arhitekture cijele aplikacije.

2.2.1. Sigurnosni problemi

Kako su internetske aplikacije dostupne svima putem interneta, pristup njima je javan te je u mnogim rješenjima potreban pristup aplikaciji putem nekog sustava autentikacije i autorizacije. Uz mnogo rješenja poput JsonWebTokena (*JWT*), sigurnost web aplikacija postaje manji problem.

Važno je paziti na to što neprijavljeni korisnici (tzv. gosti) smiju raditi na aplikaciji te je iznimno važno osigurati da obični korisnici nemaju pristup osjetljivim podacima. Uz to, neophodno je osigurati rute na vizualnom dijelu aplikacije te na dijelu aplikacije koji komunicira s bazom. To se podrazumijeva ako se prati tzv. troslojni model aplikacije unutar kojeg se aplikacija sastoji od tri dijela:

- Vizualni dio (tzv. *frontend*)
- Dio koji je zaslužan za komunikaciju s bazom (tzv. *backend*)
- Baza podataka

2.2.2. Performanse

Vrlo je poznata činjenica da su web rješenja za aplikacije sporija od klasičnih aplikacija kada je potrebno iskoristiti većinu računalnih resursa, tj. većinu radne snage računala koji pokreće aplikaciju.

JavaScript je kao programski jezik fokusiran na lakoću kreiranja. Djelomično i zbog sigurnosnih razloga, internetski preglednici ne dozvoljavaju JavaScript programima pristup memoriji kao što to imaju klasični programski jezici poput jezika C i C++. Programska rješenja u JavaScriptu pokreću se na samo jednoj dretvi što u zadacima koji zahtijevaju snažnu procesorsku moć dovodi do neefikasnosti. No, valja napomenuti kako je JavaScript tu jednu dretvu iskoristio na vrlo dobar način korištenjem raznih praksi kao što su neblokirajući asinkroni pozivi.

Ukratko, ako je potrebna računalna snaga unutar internetske aplikacije, valjalo bi proučiti brzorastući trend zvan WebAssembly koji omogućava kompajliranje mnogih programskih jezika u binarni oblik kojeg razumiju internetski preglednici. Korištenje WebAssembly rješenja našlo je razne primjene, no, još mnogo prepreka preostaje na putu do široke implementacije takvih aplikacijskih rješenja

3. RAZVOJNI CIKLUS APLIKACIJE

Moderne internetske aplikacije zahtijevaju sve preciznije procjene i zahtjeve te sve kompleksnije primjene. U inicijalnim procjenama vrlo bitnu ulogu imaju tzv. poslovni analitičari i određeni članovi programerskog tima. Ovisno o kompleksnosti problema, tim može sadržavati i tzv. arhitekta rješenja koji mora imati široku sliku programskog rješenja za problem koji uz biznis analitičare mora riješiti. Nakon toga arhitekt programskog rješenja zadaje inicijalne smjernice voditeljima timova na projektu te uz njih prati zahtjeve klijenta te nudi okvirna rješenja. Arhitekti programskog rješenja ne bi trebali mnogo sudjelovati u samom programiranju. Oni bi trebali dati naputke kako nešto implementirati.

Nakon procjena i inicijalnog postavljanja smjera u kojem će se arhitektura aplikacije razvijati, voditelji timova raspisuju zadatke sukladno sa zahtjevima koje su biznis analitičari i klijent utvrdili.

Nakon procjena i inicijalnih zadataka, počinje razvoj aplikacije.

4. KORIŠTENI ALATI

Verzioniranje programskog rješenja ostvareno je na platformi GitLab koja unutar svog besplatnog paketa nudi pregršt opcija kao što je postavljanje automatskog pokretanja aplikacije na vlastitom serveru što je iskorišteno prilikom postavljanja aplikacije na produkcijsku okolinu.

Vizualni dio aplikacije napisan je u Googleovom alatu zvanom Angular koji je vrlo robusno rješenje za poslovne aplikacije te samim time vrlo popularan unutar tvrtki koje se bave tzv. B2B (Business to business) modelom rada. Uz Angular, koristi se sveobuhvatan paket komponenti zvan PrimeNG te moderan paket CSS klasa zvan Tailwind CSS koji u modernom svijetu programiranja uzima sve više zamaha. Uz te bitnije pakete, koristi se i paket za prevođenje stranica zvan NGX-Translate te alat za pakiranje cijelog vizualnog dijela aplikacije u efikasnije dijelove zvan Webpack.

Dio aplikacije koji komunicira s bazom napisan je u relativno novom alatu pod nazivom NestJS. Navedeni alat dobiva sve veću pažnju u svijetu web programiranja, naročito u kombinaciji s Angularom. Unutar ovog dijela aplikacije, koriste se koncepti koji u daljnjim fazama razvoja omogućavaju lakšu implementaciju mikroservisne arhitekture.

Za autentikaciju i autorizaciju koristi se besplatan alat zvan Keycloak. Keycloak nudi razna sigurnosna rješenja poput uklanjanja korisnikovih tokena ako je došlo do propusta, postavljanje na vlastiti server i slično. Naravno, te funkcionalnosti nalaze se povrh iznimno dobro implementiranih funkcionalnosti za izdavanje tokena, prijenos i sigurnost istih te još mnogo drugih industrijskih standarda.

Unutar web aplikacije postoje dvije baze podataka. Obje baze podataka su PostgreSQL. Navedena baza podataka osvojila je svijet web programiranja svojom jednostavnošću i mogućnostima. Kroz godine, zajednica koja razvija PostgreSQL implementirala je mnoge koncepte koje su korisnici trebali i isticali na forumima te se tako, među ostalom, PostgreSQL počeo koristiti diljem interneta. Unutar aplikacije, jedna baza podataka služi Keycloaku za skladištenje podataka o korisnicima dok druga baza podataka služi za ostatak aplikacije.

Podizanje aplikacije na produkcijsku okolinu omogućava korištenje Docker kontejnera na privatnom virtualnom serveru zakupljenom od tvrtke Hetzner. Vlastita domena zakupljena je putem platforme zvane Namecheap dok je HTTPS certifikat sigurnog prometa izdan od strane Certbota. Alat zvan Nginx omogućava lakšu orkestraciju i preusmjeravanje prometa na servise unutar aplikacijskog rješenja te uz Namecheap DNS postavke omogućava korištenje poddomena kako bi aplikacijsko rješenje bilo dostupno na serveru koji već pokreće nekoliko aplikacija.

5. VIZUALNI DIO APLIKACIJE

5.1. Uvod u Angular

Preteča Googleovog alata poznatog pod nazivom Angular bila je verzija Angulara napisana u JavaScriptu. Ta prva verzija poznata je pod imenom AngularJS. Prvotna verzija Angulara gotovo u potpunosti je zamijenjena verzijom 2.0 koja je unijela vrlo velike promjene u arhitekturi samog alata. Prvotna verzija vrlo je brzo ostala bez održavanja te su rijetki projekti ostali na njoj. Angular 2.0 objavljen je 14. rujna 2016. godine i od tad predstavlja alat koji je vrlo popularan unutar tvrtki koje razvijaju programska rješenja namijenjena za korištenje unutar velikih tvrtki.

5.2. Usporedba s ostalim alatima

U usporedbi s ostalim popularnim alatima poput Reacta i VueJSa, Angular se ističe na temelju nekoliko specifičnosti poput nametanja stila pisanja programskog koda i same arhitekture aplikacije.

Angular potiče programere na separaciju programskog koda u module. Moduli predstavljaju zasebne cjeline koje se teoretski trebaju moći vrlo jednostavno implementirati u druge projekte. Taj modul može se sastojati od mnogih drugih modula i moduli mogu dijeliti jedni druge koristeći koncept zajedničkih modula. To je velika prednost u usporedbi s ostalim alatima, no znatno povećava donju razinu iskustva potrebnog za ispravno korištenje koncepata unutar Angulara. Moduli omogućuju vrlo visoku čitljivost programskog rješenja i znatno poboljšavaju performanse kako raste veličina aplikacije. Poboljšanja u performansama postižu se konceptom lijenog učitavanja (eng. *lazy loading*). Taj koncept omogućuje da klijent preuzima module i njihov programski kod tek onda kad su mu ti podaci potrebni, primjerice prilikom navigiranja aplikacijom.

Korištenje JavaScripta kao programskog jezika već godinama se izbjegava unutar svijeta Angulara. Naime, Angular nameće korištenje TypeScripta koji je posljednjih godina postao standard unutar industrije zbog svoje robusnosti u usporedbi s JavaScriptom i ostalim prednostima koje nude statički pisani programski jezici. Činjenica da je gotovo nemoguće pisati Angular bez znanja TypeScripta predstavlja još jedan razlog zašto Angular ima vrlo strmu krivulju učenja.

Dok ostali alati fokus postavljaju na jednostavnost kreiranja, Angular se početnicima može činiti kao nepotrebno kompliciran alat te kao takav gubi mnoge početnike koji se rijetko kad vraćaju Angularu nakon što steknu iskustvo. Taj trend naročito je vidljiv prilikom godišnje ankete koju provodi platforma StackOverflow. Unutar te ankete, programeri vrlo često glasaju za Angular kao jedan od najtežih alata za programiranje internetskih aplikacija.

5.3. Važni koncepti unutar Angulara

Komponente predstavljaju bitan dio svake aplikacije napisane pomoću Angulara. Najmanji funkcionalni dio svakog modula predstavlja komponenta. Svaka komponenta može imati podkomponente (eng. *child components*). To omogućava vrlo čitljivo razdvajanje programske logike u manje cjeline. Postoji li, primjerice, zahtjev za komponentu koja prikazuje popis korisničkih kartica i iznad njih statistiku potrošnje, postoji odlična šansa za razdvajanje programske logike. Tako bi tražena komponenta sadržavala dvije manje komponente i na taj bi način, sama komponenta imala znatno manje nepovezanog programskog koda.

Moduli u teoriji predstavljaju odvojene logičke cjeline koje se sastoje od mnogih komponenti, servisa i ostalih dijelova potrebnih za neovisan rad modula. Neovisan rad modula kao takav je vrlo težak za ostvariti te je zbog toga dobra praksa imati zajedničke module koji vrlo često sadrže sve komponente koje su potrebne diljem cijele aplikacije. Taj način korištenja zajedničkih modula usporava početno učitavanje internetske aplikacije, no znatno ubrzava daljnje korištenje iste jer se zajednički moduli ne učitavaju ponovno.

Servisi služe kao klase koje su dostupne većem broju komponenti. Unutar tih klasa mogu se implementirati razne zajedničke funkcionalnosti poput dohvaćanja podataka, rad nad entitetima i slično. Servisi su u pravilu inicijalizirani kao klase koje imaju samo jednu instancu diljem cijele aplikacije. To bi značilo da nakon prvog poziva, svaka iduća komponenta koja pokuša inicijalizirati servis, dobiva već kreiranu instancu servisa na korištenje. Naravno, omogućeno je i definiranje djelokruga servisa, tj. moguće je postaviti da samo komponente unutar određenog modula dijele istu instancu servisa dok ostali moduli inicijaliziraju vlastite instance servisa.

Čuvari ruta (eng. *guards*) vrlo su korisni prilikom implementiranja autorizacije unutar internetske aplikacije. Naime, čuvari ruta pozivaju se prije instanciranja komponente koja se pokazuje na nekoj ruti. Ako čuvari ruta jave da je sve u redu, tek onda se inicijaliziraju komponenta ili modul koje čuvari čuvaju. Jedan od najčešćih primjera je provjera prijavljenosti korisnika i korisničkih prava pristupa.

Presretači prometa (eng. *interceptors*) omogućavaju presretanje HTTP prometa u oba smjera, tj. odlazeće HTTP pozive i njihove odgovore. Vrlo čest primjer je automatski prikaz poruka o uspješnosti HTTP zahtjeva. Primjerice, prilikom promjene nad entitetom, sa sloja koji upravlja bazom podataka moguće je vratiti ključ za poruku koja se onda pomoću presretača prometa prikaže korisniku. Još jedan čest primjer korištenja presretača prometa je provjera statusa HTTP odgovora. Točnije, često se provjerava hoće li odgovor imati status neovlaštenog pristupa s brojem 401. Ako se takav HTTP odgovor pojavi, presretač prometa često korisniku obriše pristupni token i pošalje korisnika na stranicu za prijavu.

Transformatori podataka (eng. *transform pipes*) služe za različite transformacije nad podatkom koji se prikazuje korisniku. Uobičajeno se koriste za jednostavnije radnje oblikovanja teksta unutar HTML koda. Angular nudi razne transformatore za datume, valute i tekst. Primjerice, ako se prikazuje cijena nekog kupljenog artikla, ta je cijena u pravilu neformatirana te je zapisana kao običan broj. Proslijedi li se taj broj u transformator podatka za valutu, Angular će automatski dodati formatiranje za tisućice, decimalna mjesta te prikazati simbol valute. Naravno, moguće je prosljeđivati parametre u transformatore podataka kao i pisati vlastite transformatore.

6. KOMUNIKACIJA S BAZOM PODATAKA

6.1. Uvod u NestJS

NestJS je alat koji pomaže pri pisanju NodeJS aplikacija koje se pokreću na serveru. Prva stabilna verzija ovog alata objavljena je 2017. godine i od tada mu popularnost postepeno raste. S više od milijun tjednih preuzimanja, NestJS sve se više koristi i unutar aplikacija koje zahtijevaju besprijekoran rad na produkcijskoj okolini. Za razliku od ostalih alata za pisanje serverskih aplikacija u NodeJSu, NestJS nameće korištenje TypeScripta i praćenje određenog stila pisanja programskog rješenja koje sintaksno vrlo podsjeća na programsko rješenje pisano unutar Angulara. To je jedan od mnogih razloga zbog kojeg razvojni programeri koji koriste Angular uče i NestJS.

6.2. Usporedba s ostalim alatima

U usporedbi s ostalim alatima kao što su Laravel, Ruby on Rails, Spring i Django, NestJS se ističe time što se piše u TypeScriptu. Početnicima koji su razvijali programska rješenja koja se pokreću unutar preglednika olakšano je učenje alata jer ne moraju učiti novi programski jezik, već samo glavne koncepte koje nudi alat. Ta je prednost još izraženija kod razvojnih programera koji su koristili Angular. U tom će slučaju mnogi koncepti unutar NestJSa novim razvojnim programerima biti vrlo poznati.

Velika prednost koja nastaje prilikom korištenja istog programskog jezika i na dijelu aplikacije koji se pokreće kod klijenta i na dijelu aplikacije koji se pokreće na serveru je korištenje zajedničkog programskog koda. Primjerice, mnoge biblioteke mogu se u isto vrijeme pokretati i na serveru i na klijentu. Mnoge klase i sučelja koja predstavljaju entitete u bazi podataka mogu se jednom napisati i koristiti na više mjesta.

Prilikom objavljivanja aplikacije na produkciju prednost korištenja istog programskog jezika za serverski i klijentov dio aplikacije je da su zahtjevi za pokretanje programskog rješenja gotovo identični. To uvelike olakšava posao razvojnih programera koji moraju održavati aplikaciju pokrenutom, ažuriranom i sigurnom.

6.3. Važni koncepti unutar NestJSa

NestJS sintaksno podsjeća na Angular od kojega je mnoge koncepte preuzeo i implementirao na način vrlo poznat razvojnim programerima serverskih aplikacija. Samim time,

kao i Angular, NestJS temelji se na konceptu modula i odvajanja logičkih cjelina unutar programskog rješenja. Najčešće će modul strukturno podsjeća na tzv. MVC arhitekturu, tj. modul će sadržavati kontroler i servis kojeg poziva kontroler za ostvarivanje raznih funkcionalnosti. Takav pristup bit će vrlo poznat programerima koji su razvijali serverska rješenja u nekim drugim alatima. Ako se od samoga početka dobro razdvaja logika unutar modula, skaliranje i promjene na arhitekturi bit će lakše.

Slično kao u Angularu, servisi implementiraju različite funkcionalnosti, dok kontroleri obrađuju zahtjev na temelju kojeg pozivaju određenu funkcionalnost iz servisa.

NestJS podržava funkcije koje se mogu pozivati prije ili poslije poziva kontroleru ovisno o ruti na koju je došao HTTP zahtjev. To je vrlo sličan koncept presretačima prometa unutar Angulara. Česta primjena ovih funkcija je implementacija filtera za greške, odrađivanje programske logike prilikom greške i oblikovanje odgovora koji sa servera odlaze klijentu. Na taj se način uvelike smanjuje količina koda koji bi se inače morao multiplicirati diljem aplikacije.

Čuvari ruta implementirani su i unutar NestJSa te većinom služe sličnu svrhu kao i na klijentskom dijelu aplikacije, tj. da na svakom HTTP zahtjevu provjeravaju ima li korisnik sva prava potrebna za pristup ruti i logici koja će se izvršiti.

Dekoratori su vrlo popularan koncept kojime se vrlo jednostavno primjenjuju unaprijed definirane programske logike. NestJS ima mnoge ugrađene dekoratore kojima se, primjerice, mogu dohvatiti podaci unutar HTTP zahtjeva, tijelo HTTP zahtjeva, podaci zaglavlja ili bilo koja vlastita funkcionalnost. Ti se dekoratori često koriste unutar parametara funkcija kontrolera kako bi se iščitali podaci iz HTTP zahtjeva i proslijedili u servis koji na temelju tih podataka odrađuje programsku logiku.

Vrijedi napomenuti kako su mnogi napredniji koncepti odlično opisani unutar službene NestJS dokumentacije [1]. Također, službena dokumentacija nudi primjere implementacije raznih biblioteka i alata unutar NestJSa. Primjeri naprednih koncepata koji su ostvareni korištenjem NestJSa i drugih biblioteka su tzv. internetske utičnice (*eng. web sockets*), korištenje alata za olakšano upravljanje bazama podataka poput Prisme, događaji (*eng. events*) poslani sa servera, raspoređivanje zadataka na serveru, korištenje posrednika poruka poput Kafke ili RabbitMQa itd.

7. BAZE PODATAKA

7.1. Baza korisnika aplikacije

Keycloak ima mogućnost korištenja baze podataka unutar radne memorije, no to se nikako ne preporučuje na produkcijskoj okolini. Kako bi podaci bili osigurani i prilikom greške unutar autentikacijskog servisa, Keycloaku se daje zasebna baza podataka. Pristupne podatke za bazu podataka korisnika aplikacije potrebno je dati Keycloaku kako bi znao koju bazu podataka treba koristiti.

Jedan od većih nedostataka ovog pristupa je gubitak funkcionalnosti vanjskog ključa pojedinog entiteta u bazi podataka. Naime, gotovo svaki podatak unutar aplikacije sadrži informacije o korisniku koji ga je kreirao. Nalaze li se korisnici i sami podaci unutar iste baze podataka, vrlo je jednostavno implementirati zaštitu kojom se provjerava postoji li doista taj korisnik u bazi podataka. No, u ovom slučaju, tu logiku treba odvojiti od same baze podataka.

Sloj aplikacije koji je zaslužan za komunikaciju s bazom podataka tako će na svakom HTTP zahtjevu koji zahtjeva token korisnika provjeriti unutar Keycloaka postoji li taj korisnik i validirati mu token. Bude li korisnikov token odbijen iz nekog razloga, HTTP zahtjev neće ni doći do same baze podataka. Iz tog razloga gotovo sigurno je da su korisnikovi podaci na entitetu točni.

Naravno, problem će se pojaviti unutar svake naprednije radnje na samoj bazi podataka. Primjerice, žele li se dodati razne automatske funkcionalnosti prilikom registracije korisnika, nemoguće je koristiti okidače na samoj bazi podataka korisnika jer ona nema pristup bazi podataka ostalih dijelova aplikacije. Naravno, postoje razna rješenja za navedeni problem, no svako takvo rješenje izvan same baze podataka unosi novu kompleksnost unutar aplikacije.

7.2. Baza podataka aplikacije

Sama baza podataka aplikacije u trenutnoj, tj. monolitnoj arhitekturi sadrži sve podatke izuzev podataka o korisnicima. Takav je pristup teoretski najjednostavniji jer razvojni programeri moraju paziti na samo jednu bazu podataka i migracije koje odrade na njoj. Promijeni li se arhitektura aplikacije iz monolitne u mikroservisnu, ova baza podataka bit će rascjepkana u nekoliko baza podataka, po jedna za svaku logičku cjelinu aplikacije.

Postoji nekoliko entiteta koji se spremaju unutar baze podataka. Korištenjem alata pod nazivom Prisma, moguće je vrlo lako upravljati entitetima i kreirati sučelja i klase tih entiteta

koje je moguće koristiti unutar dijela aplikacije koji komunicira s bazom podataka. Prismu je vrlo lako postaviti praćenjem uputstava na službenoj dokumentaciji [2]. Nakon inicijalnog postavljanja alata, jedna od kreiranih datoteka nosit će naziv *schema.prisma*. Unutar te datoteke, moguće je definirati entitete i kreirati migracije potrebne da se ti entiteti preslikaju u bazu podataka.

Prvotna verzija aplikacije sastoji od nekoliko modela. Modeli su sljedeći:

- CreditCard
- CreditCardIssuer
- CreditCardItem
- Exception
- ProfileSettings

Programski kod 7.1: Definiranje modela kreditne kartice

```
model CreditCard {
  id           String  @id @default(uuid())
  userId       String
  name         String  @db.VarChar(255)
  limit        Decimal @db.Decimal(10, 2)
  billingDate  Int

  createdAt    DateTime  @default(now())
  updatedAt    DateTime  @updatedAt
  deletedAt    DateTime?

  limitType    CreditCardLimit

  issuer       CreditCardIssuer @relation(fields: [issuerId], references: [id])
  issuerId     String

  items        CreditCardItem[]

  @@map("credit_cards")
}
```

Model kreditne kartice sastoji se od nekoliko ključnih cjelina. Kao gotovo svaki entitet, sadržava polje *id* koje predstavlja primarni ključ. Koristeći to polje, moguće je povezati ostale entitete na model kreditne kartice i obratno. Dodavanjem dekoratora *@id* i *@default(uuid())*, Prisma će razumjeti da se traži korištenje tog polja kao primarni ključ i da taj identifikator mora biti predstavljen kao tzv. *uuid* (eng. *universally unique identifier*). Tako će navedeni identifikator

biti niz znakova koji se generiraju prilikom unosa u bazu podataka. Polje *userId* predstavlja vanjski ključ koji pokazuje na identifikator korisnika koji je vlasnik te kreditne kartice. Od ostalih ključnih polja, tu su *issuer* i *issuerId* pomoću kojih Prisma razumije da se želi kreirati veza jedan naprema više, što u konkretnom primjeru značilo da jedan izdavač kreditnih kartica može imati više kreditnih kartica dok jedna kreditna kartica može imati samo jednog izdavača. Na samome kraju definiranja entiteta, moguće je definirati redak *@@map("credit_cards")* pomoću kojeg se definira ime tablice za kreirani entitet.

Programski kod 7.2: Definiranje modela izdavača kreditnih kartica

```
model CreditCardIssuer {
  id       String @id @default(uuid())
  name String @db.VarChar(255)
  logo   String @db.Text() // Base64 string

  createdAt      DateTime  @default(now())
  updatedAt     DateTime  @updatedAt
  deletedAt      DateTime?

  creditCards   CreditCard[]

  @@map("credit_card_issuers")
}
```

Model izdavača kreditnih kartica sadrži mnoga polja identičnih uloga kao i model za kreditne kartice. Jedna od razlika je korištenje *@db.Text()* dekoratora kojim Prisma unutar PostgreSQL baze podataka kreira polje tipa *Text*. Unutar PostgreSQL baze podataka, tip *Text* predstavlja znakovno polje neograničene duljine. Takvo je polje iznimno korisno koristiti prilikom spremanja slika u samu bazu podataka u formatu zvanom *base64*. U tom formatu, slika je predstavljena nizom znakova koji se prilikom dohvaćanja mogu dekodirati i oblikovati u sliku. Vrijedi napomenuti da je prilikom korištenja ove vrste polja dobra praksa ograničiti veličinu podataka koji se može upisati u bazu podataka kako netko zlonamjeran ne bi postavljao izrazito velike datoteke i tako usporavao bazu. Unutar modela entiteta, vidljiv je i drugi dio veze između izdavača kreditnih kartica i samih kreditnih kartica. Naime, model izdavača kreditnih kartica sada mora spremati niz objekata kreditnih kartica unutar polja. Tako Prisma u potpunosti razumije da se želi kreirati veza između entiteta u bazi podataka.

Programski kod 7.3: Definiranje modela stavke kreditne kartice

```
model CreditCardItem {
  id                  String   @id @default(uuid())
  userId              String
  name                String   @db.VarChar(255)
  description         String?  @db.Text()
  boughtAt            DateTime
  firstInstalmentDate DateTime
  instalments         Int
  amount              Decimal  @db.Decimal(10, 2)

  createdAt DateTime  @default(now())
  updatedAt DateTime @updatedAt
  deletedAt DateTime?

  card          CreditCard     @relation(fields: [cardId], references: [id])
  cardId        String

  @@map("credit_card_items")
}
```

Stavka kreditne kartice mora biti povezana na sam entitet kreditne kartice i na korisnika koji je kreirao taj entitet. Kao i u primjeru veze između izdavača kreditnih kartica i samih kreditnih kartica, i u ovom primjeru jedna stavka kreditne kartice može biti vezana na samo jednu kreditnu karticu dok jedna kreditna kartica može imati više stavki.

Programski kod 7.4: Definiranje modela iznimke u radu sustava

```
model Exception {
  id          String  @id @default(uuid())
  userId      String
  endpoint    String  @db.Text()
  message     String? @db.Text()
  params      String? @db.Text()
  query       String? @db.Text()

  createdAt   DateTime @default(now())

  @@map("exceptions")
}
```

Vrlo koristan entitet prilikom pojave pogreške u radu sustava je i *Exception* entitet koji unutar baze podataka služi za spremanje podataka o greški koja se dogodila u sustavu. Unutar

baze podataka spremaju se razni podaci koji razvojnom programeru mogu olakšati pronalazak slučaja koji je izazvao grešku. Tako se, primjerice, o grešci spremaju podaci poput poruke greške, parametara koji su došli do sloja za komunikaciju s bazom podataka i same krajnje točke koju je vizualni dio aplikacije pozvao i tako prouzročio pogrešku u radu sustava.

Programski kod 7.5: Definiranje modela korisničkih postavki

```
model ProfileSettings {
  userId          String @unique
  language        String @db.VarChar(3)
  currency        String @db.VarChar(3)

  createdAt       DateTime @default(now())

  @@map("user_profile_settings")
}
```

Kako bi aplikacija bila prilagodljiva korisnikovim potrebama, unutar baze podataka spremaju se i podaci poput valute i jezika aplikacije za pojedinog korisnika. Sloj zaslužan za komunikaciju s bazom podataka, prilikom korisničke registracije, kreira zapis o postavkama koje korisnik kasnije može ažurirati.

7.3. Pokretanje baza podataka

Baze podataka potrebne za rad aplikacije vrlo je lako pokrenuti korištenjem alata Docker Compose. Navedeni alat omogućava definiranje pojedine baze podataka i raznih varijabli okruženja poput pristupnih podataka samoj bazi podataka i postavljanja porta na kojemu je dostupna pojedina baza podataka.

Pomoću Docker Compose alata, vrlo je jednostavno definirati mapu unutar koje će se spremati podaci baze podataka. To omogućava da se i nakon ponovnog pokretanja servisa podaci ne izgube. U isto vrijeme, kako se ti podaci ne nalaze na GitLabu, svaki razvojni programer može raditi s tim bazama podataka što god želi te vrlo lako vratiti verziju baze podataka na neku prethodnu ako dođe do problema. Podrazumijeva se da su sve migracije spremne za primjenu na novoj bazi podataka ako je potrebno vratiti bazu podataka na neku prethodnu verziju.

Vrijedi spomenuti i kako je iznimno korisno definirati varijable okruženja s podacima za pristup bazi podataka. To omogućava da svaki programer aplikacije ima gotovo isto okruženje kao što će biti i na produkcijskom serveru, samo s drugačijim pristupnim podacima. Produkcijski

pristupni podaci definiraju se na samom GitLabu te su navedeni podaci skriveni za sve javne korisnike. To je izuzetno korisno ako je aplikacija otvorenog tipa koda pa je svaki redak programskog rješenja javno dostupan. Pristupni podaci za produkciju nikako ne smiju biti javno dostupni.

Programski kod 7.6: Definiranje servisa za pokretanje baza podataka

```
version: '3'

volumes:
  postgres_data:
    driver: local
  mm-postgres:
    driver: local

services:
  keycloak-postgres:
    image: postgres:14.4
    volumes:
      - postgres_data:/var/lib/postgresql/data
    environment:
      POSTGRES_DB: ${KEYCLOAK_DB_DATABASE}
      POSTGRES_USER: ${KEYCLOAK_DB_USER}
      POSTGRES_PASSWORD: ${KEYCLOAK_DB_PASSWORD}
    ports:
      - 5433:5432
  mm-postgres:
    image: postgres:14.4
    volumes:
      - mm-postgres:/var/lib/postgresql/data
    environment:
      POSTGRES_DB: ${POSTGRES_DB}
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    ports:
      - 5434:5432
```

Varijable označene znakom dolara predstavljaju varijable okruženja. Njih Docker Compose alat automatski pokušava pročitati iz datoteke *.env*. Unutar te datoteke potrebno je definirati pristupne podatke za lokalnu bazu podataka. Ti će pristupni podaci biti drugačiji na produkciji pa je zbog toga moguće navedenu datoteku imati dostupnu unutar sustava za verzioniranje.

7.4. Buduće verzije

7.4.1. Mikroservisna arhitektura

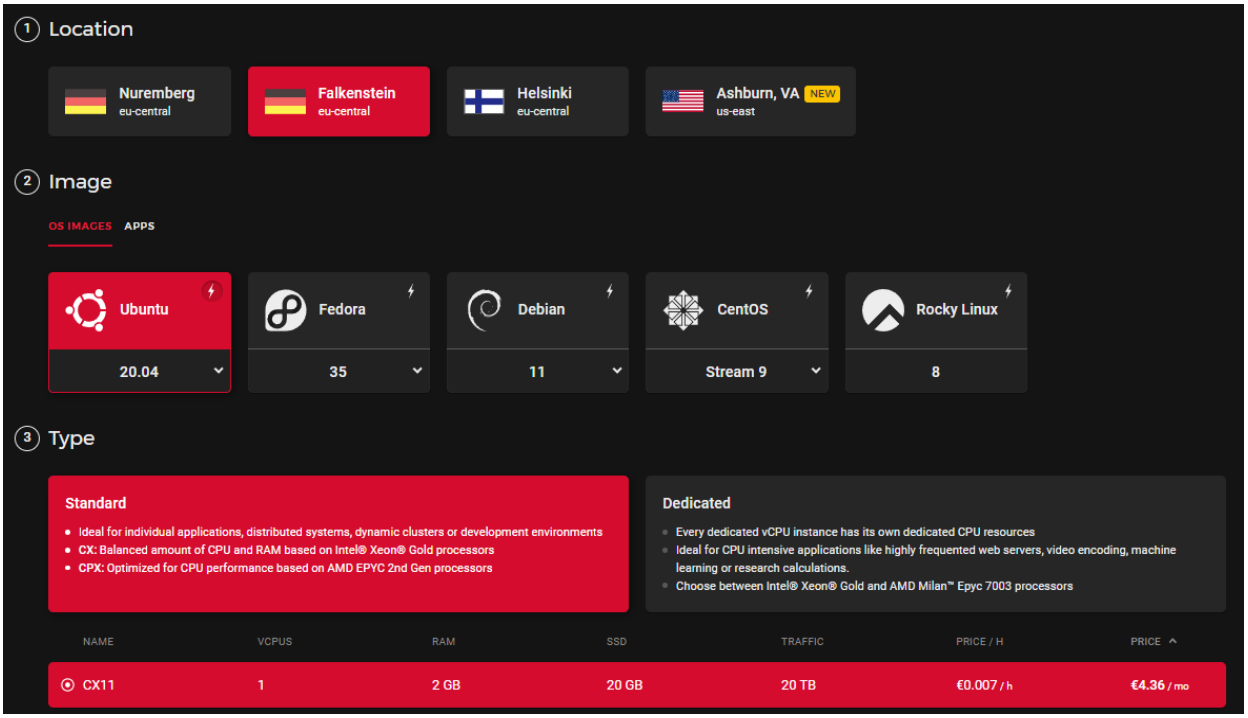
Ako je idući korak razvoja implementirati navedenu aplikaciju prema načelima mikroservisne arhitekture, potrebno je razdvojiti baze podataka ovisno o funkcionalnostima servisa kojem pojedina baza podataka pripada.

Primjerice, ako jedan od novih servisa omogućava praćenje vlastitih investicijskih pozicija, tada taj servis ima vlastitu bazu podataka koja skladišti samo podatke vezane za taj servis. Broj podataka unutar ovakve baze podataka rast će znatno sporije prilikom rasta broj korisnika.

8. POSTAVLJANJE NA PRODUKCIJSKU OKOLINU

8.1. Zakup servera

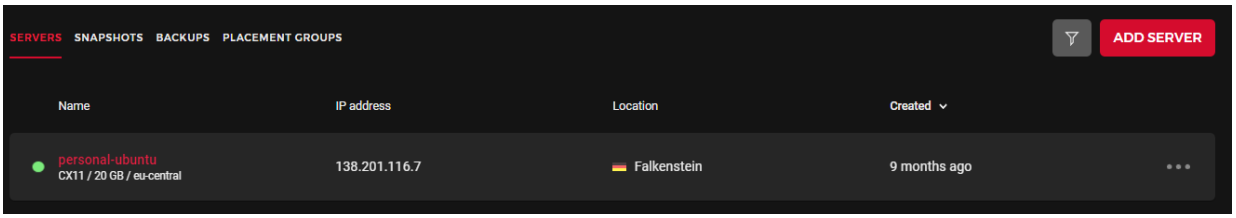
Privatni virtualni server zakupljen je na web stranici tvrtke Hetzner. U ponudi postoje serveri od 4.36 €. U bazičnoj opciji, 25. ožujka 2022. godine, Hetzner nudi jedan vCPU (procesorska jezgra), 2 GB radne memorije, 20 GB trajne memorije te 20 TB prometa na serveru. Navedene specifikacije sasvim su dovoljne za programsko rješenje koje ne zahtijeva puno resursa te u samome početku rada nema tisuće konkurentnih poziva. U slučaju rasta aplikacije, potrebno je postaviti skaliranje aplikacije korištenjem većeg broja jačih servera koji su povezani balanserom opterećenja koji automatski preraspodjeljuje promet na servere ovisno o geolokaciji korisnika i opterećenosti ostalih servera.



Slika 8.1: sučelje za postavljanje servera

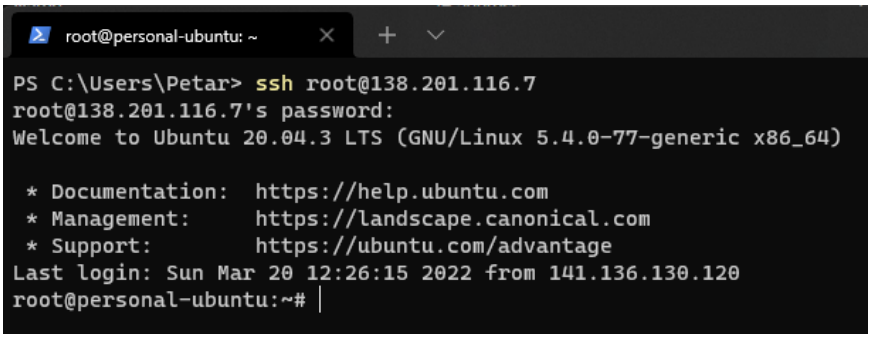
Na sučelju za postavljanje servera, odabire se server koji će biti najbliži većini korisnika i Ubuntu kao operativni sustav. Kao što je i prije navedeno, odabire se bazična opcija specifikacije servera te na samome dnu upisuje naziv servera. Uz osnovne opcije, Hetzner nudi opcije kojima se može postaviti vatrozid ili sigurnosno kopiranje koje će dodatno podići cijenu servera.

Nakon kreiranja servera, na kontrolnoj ploči moguće je vidjeti kreirani server čiji pristupni podaci dolaze putem elektroničke pošte.



Slika 8.2: status zakupljenog servera

Nakon što na mail pristignu pristupni podaci, potrebno je korištenjem alata SSH pristupiti mu. Zadani korisnik je naziva root te mu je inicijalna lozinka aktivna samo prilikom prve prijave kada server zahtijeva promjenu lozinke za root korisnika.



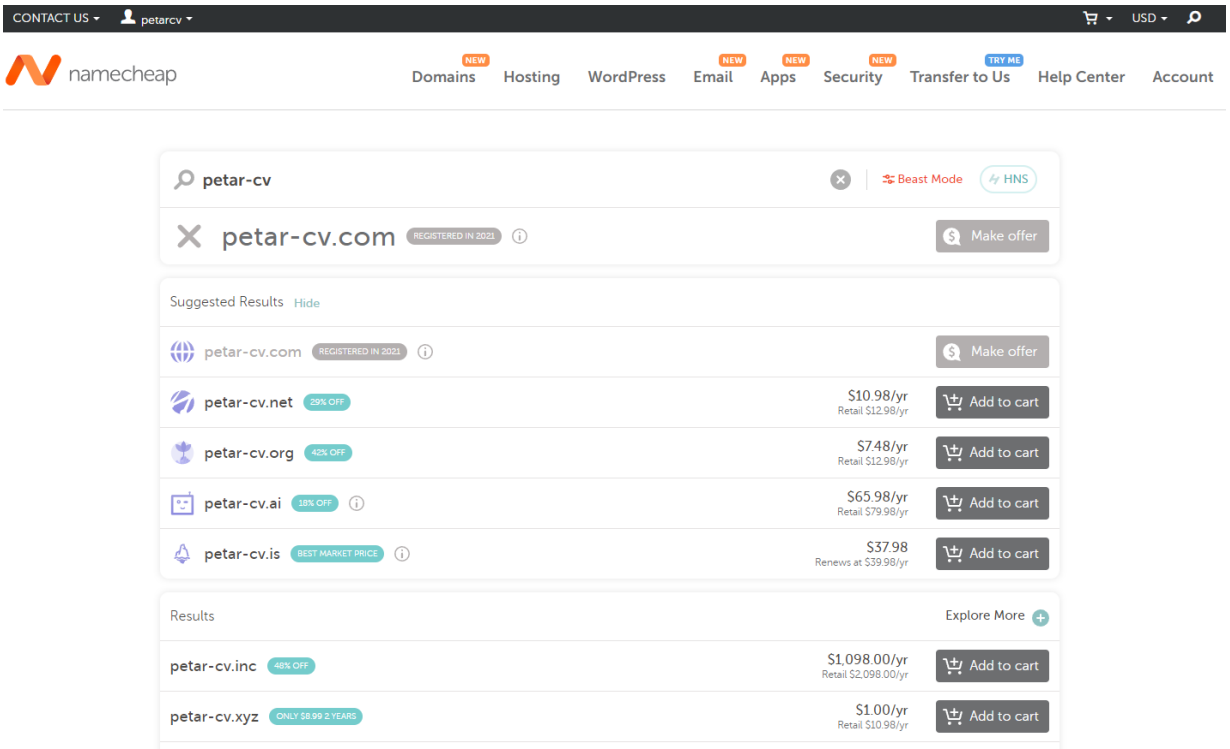
Slika 8.3: postupak prijave na zakupljeni server

Server je inicijalno postavljen vrlo minimalno pa nema mnogo predinstaliranih programa što je svakako prednost. Kako se arhitektura aplikacije zasniva i na korištenju Docker kontejnera, nema potrebe razmišljati što sve treba instalirati kako bi se pokrenula aplikacija na serveru. Docker omogućava kreiranje datoteka koje predstavljaju konfiguraciju na temelju koje se mogu pokretati razne aplikacije s minimalnim dodatnim postavljanjem. Docker kontejner moguće je zamisliti kao vrstu virtualne mašine koja dijeli razne resurse operativnog sustava s mašinom koja ju pokreće. Pokrenuti kontejner ima svoj operativni sustav koji u pravilu zauzima malo memorije jer ne zahtjeva potpunu virtualizaciju kao klasične virtualne mašine. Nastavno tome, jedino što se mora postaviti na serveru su sami Docker i Nginx.

Za dodatne informacije o postavljanju vlastitog servera na platformi Hetzner, potrebno je koristiti službenu dokumentaciju [4].

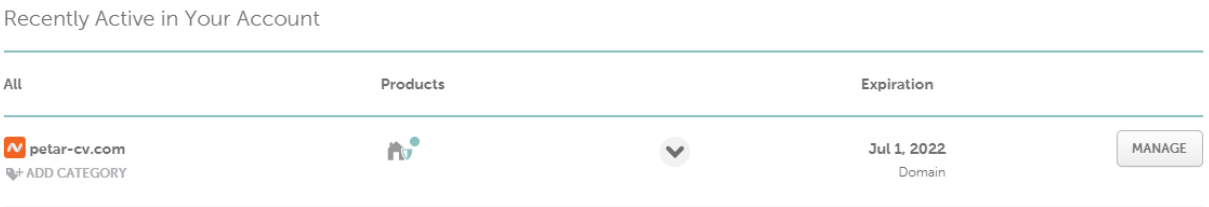
8.2. Kupnja domene

Domena web aplikacije kupljena je na platformi zvanoj Namecheap. Ta platforma nudi laku kupnju domena i pristup naprednim postavkama DNS-a što omogućava izradu poddomena za kupljenu domenu.



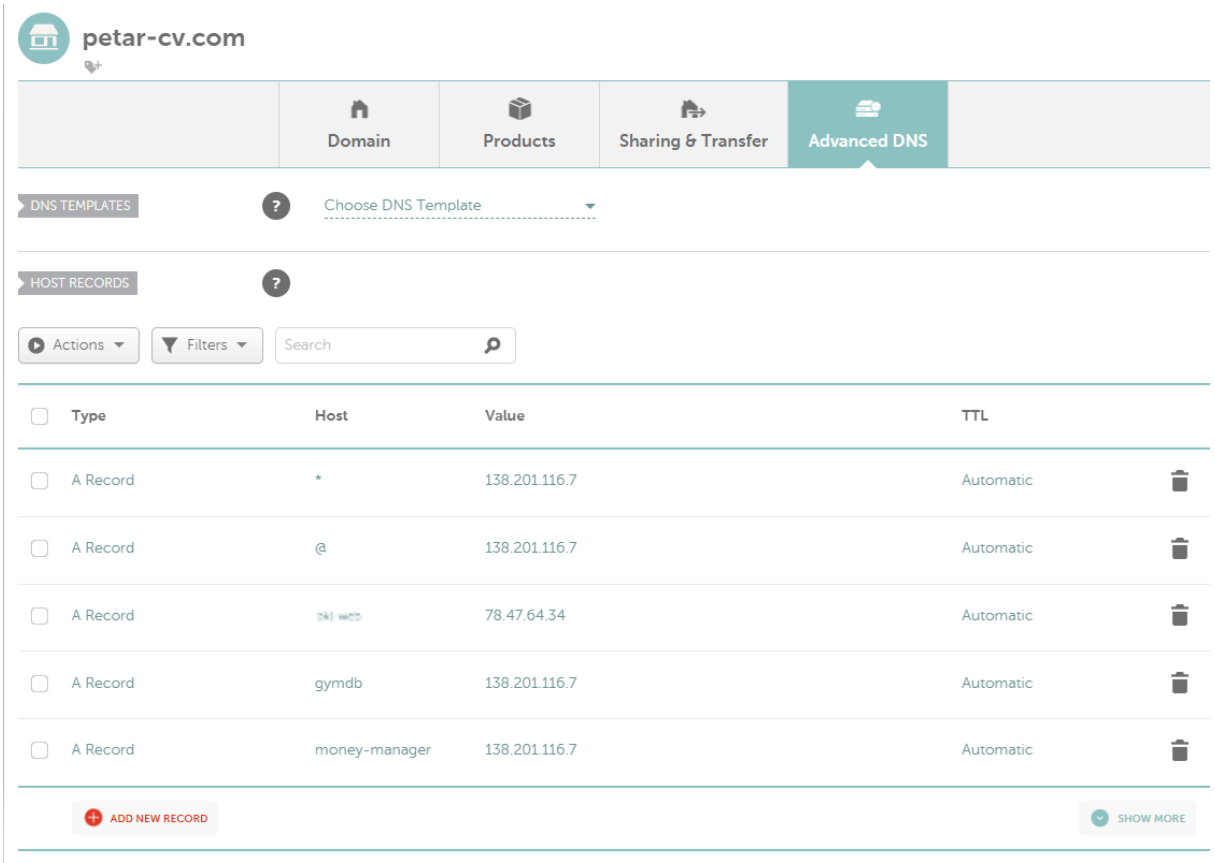
Slika 8.4: potražnja i ponuda kupnje domena

Vlasništvo domene plaća se na godišnjoj bazi. Nakon kupljene domene, moguće je na nadzornoj ploči platforme vidjeti domene u vlasništvu te datume isteka vlasništva kao i gumb koji vodi do postavki vezanih za navedenu domenu.



Slika 8.5: domene u vlasništvu

Pritiskom na gumb koji vodi na dodatne postavke domene, moguće je vidjeti razne postavke od kojih je korištena postavka “Advanced DNS”. Unutar te postavke moguće je dodavati poddomene s kojih se preusmjerava promet u Docker kontejnere koristeći Nginx.



Slika 8.6: napredne DNS postavke domene

Pritiskom na gumb za dodavanje novog zapisa, potrebno je odabrati prvu opciju "A record". Vrijednost "A record" koristi se za povezivanje domene i IP adrese servera. Pod polje "Host" potrebno je upisati naslov poddomene dok polje "Value" predstavlja IP adresu zakupljenog servera. Valja napomenuti kako jedna domena može biti povezana na više servera. Takav primjer vidi se na slici 8.6. gdje jedna poddomena preusmjerava promet na server s drugačijom IP adresom. Konačno, polje "TTL" ostavlja se na automatski pridruženoj vrijednosti.

Nakon spremanja novoupisane poddomene, Namecheap će obavijestiti kako promjena može potrajati. Razlog je što se promjene na DNS-u moraju propagirati kroz čitavi svijet.

8.3. Postavljanje Dockera

8.3.1. Instalacija Docker Enginea

Za početak rada s Dockerom potrebno ga je instalirati na zakupljeni server. Kako je operativni sustav odabranog servera Ubuntu, potrebno je pratiti službena uputstva s web stranice Dockera [3].

Potrebno je ažurirati listu i indekse apt paketa te nakon ažuriranja indeksa, instalirati pakete koji omogućavaju pristup repozitoriju putem HTTPS prometa.

Programski kod 8.1: Instalacija paketa za dobivanje HTTPS certifikata

```
sudo apt-get update

sudo apt-get install ca-certificates curl gnupg lsb-release
```

Potom je potrebno dodati službeni GPG ključ koji pripada Dockeru.

Programski kod 8.2: Dodavanje službenog GPG ključa za Docker

```
sudo mkdir -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

Za povezivanje na repozitorij koji nudi trenutnu stabilnu verziju, potrebno je koristiti naredbu:

Programski kod 8.3: Povezivanje na repozitorij sa zadnjom stabilnom verzijom Dockera

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Za dodavanje pristupa novom repozitoriju, potrebno je ponovno okinuti naredbu za dohvaćanje liste paketa i indeksa te potom instalirati potrebne pakete.

Programski kod 8.4: Instalacija paketa potrebnih za rad Dockera

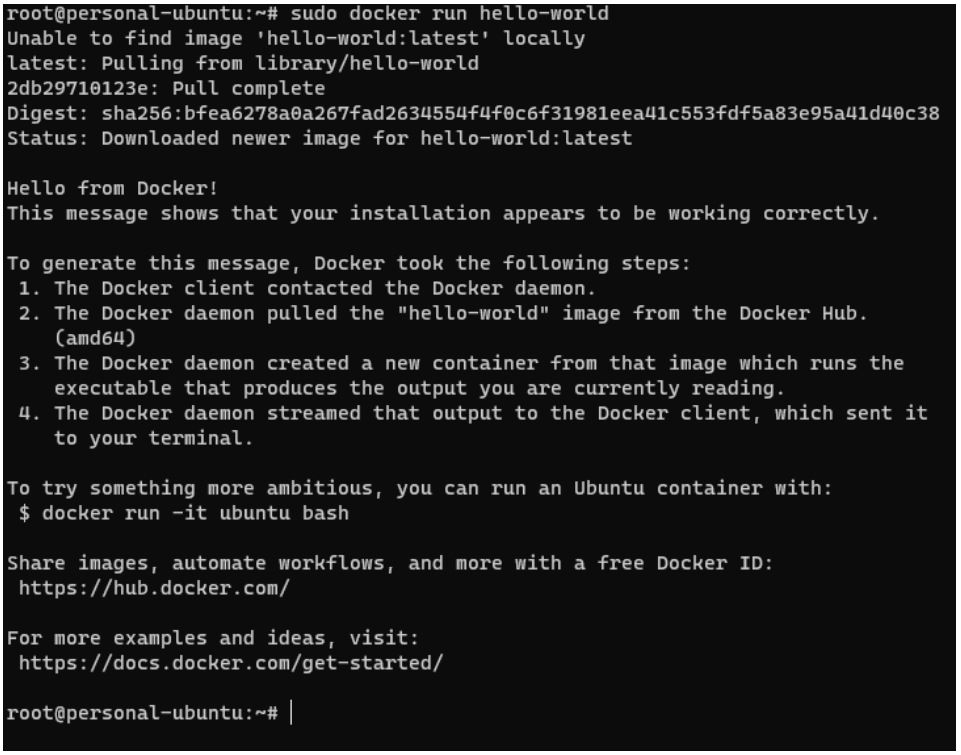
```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

Posljednji korak je isprobavanje rada Dockera. Najjednostavniji način provjere ispravnosti je pokretanjem naredbe koja sa središnjeg Docker servera povlači sliku kontejnera zvanog “hello-world”.

Programski kod 8.5: Testiranje rada Dockera

```
sudo docker run hello-world
```

Ako sve prođe ispravno, u naredbenom retku ispisuje se tekst “Hello from Docker!”.



Slika 8.7: pokretanje testnog kontejnera nakon uspješne instalacije

8.3.2. Instalacija alata Docker Compose

Nakon uspješne instalacije Dockera, može se instalirati alat zvan Docker Compose. Ovaj je alat iznimno koristan u radu s većim brojem kontejnera u isto vrijeme. Docker Compose omogućava da se unutar .yaml datoteke definira koje sve kontejnere je potrebno kreirati, na kojim portovima trebaju biti dostupni i trebaju li navedeni kontejneri biti dio mreže kako promet ne bi mogao dolaziti iz ostalih kontejnera. Naravno, tu su još i mnoge dodatne opcije poput dijeljenja prostora i slično. Glavna prednost ovog alata je što je sve predefinirane kontejnere moguće pokretati i zaustavljati jednom naredbom.

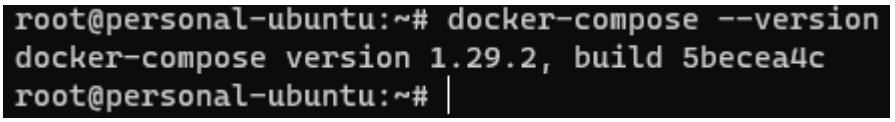
Za instalaciju ovog alata potrebno je dohvatiti zadnju stabilnu verziju i postaviti ovlasti za izvršavanje alata. Nakon toga je moguće pokrenuti naredbu za testiranje ispravnosti instalacije alata.

Programski kod 8.6: Instalacija i testiranje Docker Compose alata

```
sudo curl -L
"https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname
-s)-$(uname -m)" -o /usr/local/bin/docker-compose


sudo chmod +x /usr/local/bin/docker-compose


docker-compose —version
```



Slika 8.8: ispis naredbe kojom se provjerava verzija Docker Compose alata

8.4. Postavljanje Nginxa

Nginx je jedan od najpopularnijih web servera koji nudi razne mogućnosti poput balansera prometa, dodatnog sloja sigurnosti i preusmjeravanja prometa s domena na određene portove servera na kojima se mogu nalaziti Docker kontejneri s aplikacijama.

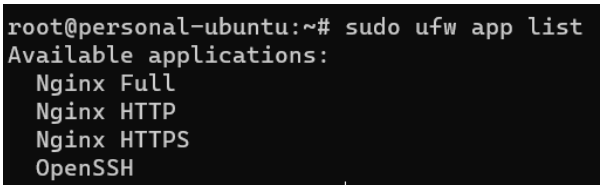
8.4.1. Instalacija Nginxa

Kao i pri instalaciji Dockera, potrebno je ažurirati listu apt paketa i indeksa te instalirati željeni program koristeći naredbe:

Programski kod 8.7: Instalacija Nginxa

```
sudo apt update
sudo apt install nginx
```

Ako je instalacija uspješna, Nginx se registrira kao servis unutar Ubuntuovog vatrozida zvanog ufw.

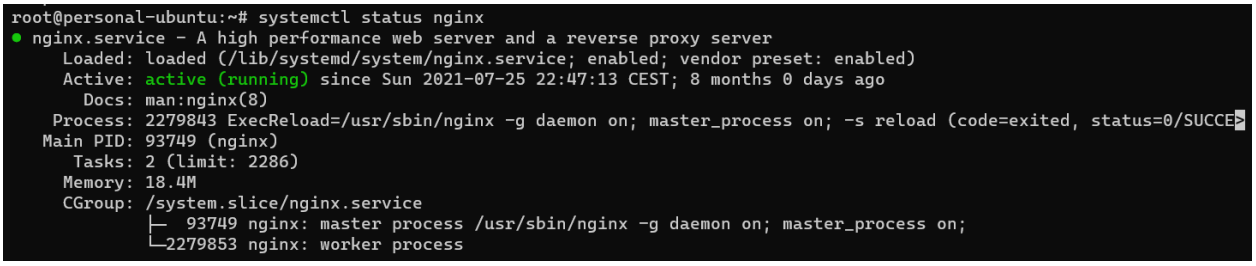


Slika 8.9: prikaz aplikacija koje imaju pristup kroz vatrozid servera

Status procesa moguće je vidjeti naredbom:

Programski kod 8.8: Provjera stanja procesa Nginx

```
systemctl status nginx
```



Slika 8.10: ispis statusa Nginx servisa

Dodatna uputstva dostupna su unutar službene dokumentacije koja je dostupna na službenoj web stranici [5].

8.4.2. Postavljanje poddomene koristeći Nginx

Postavljanje poddomene je vrlo jednostavno te se sastoji od nekoliko koraka. Prvo je potrebno na serveru unutar direktorija */etc/nginx/sites-available* kreirati datoteku čije ime predstavlja potpunu adresu putem koje se želi pristupiti aplikaciji. U ovom slučaju, adresa će biti *money-manager.petar-cv.com* pa će i sama datoteka nositi taj naziv. Unutar datoteke potrebno je upisati neke postavke kao što su port na kojem Nginx prati promet, ime servera te određene postavke za preusmjeravanje prometa.

Vizualni dio aplikacije pokreće se na portu 5000 pa bi trebalo sav promet na poddomenu preusmjeravati na taj port. U nastavku se nalazi primjer inicijalne konfiguracije. Ovu će konfiguraciju unaprijediti alat kojim se omogućava HTTPS certifikat.

Programski kod 8.9: Nginx konfiguracija za aplikaciju

```
server {
  listen 80;
  server_name money-manager.petar-cv.com www.money-manager.petar-cv.com;

  location / {
    proxy_pass http://localhost:4200;
  }
}
```

Nakon postavljanja konfiguracije, potrebno je kreirati poveznicu na datoteku unutar direktorija */etc/nginx/sites-enabled* koristeći naredbu:

Programski kod 8.10: Omogućivanje rada kreirane poddomene

```
sudo ln -s /etc/nginx/sites-available/money-manager.petar-cv.com /etc/nginx/sites-enabled/
```

Konačno, potrebno je ponovno pokrenuti Nginx servis kako bi se promjene primijenile. To je omogućeno naredbom:

Programski kod 8.11: Ponovno pokretanje Nginx servisa

```
sudo systemctl restart nginx
```

Nakon ovog koraka, postavljanje Nginxa trebalo bi biti uspješno te bi se sav promet koji dolazi na adresu *money-manager.petar-cv.com* trebao preusmjeravati na aplikaciju koja je pokrenuta na portu 4200 zakupljenog servera.

8.5. Postavljanje HTTPS certifikata

Koristeći besplatni alat zvan Certbot, moguće je vrlo lako postaviti HTTPS certifikat koji će se sam obnavljati prije isteka. Certbot kreira cronjob posao kojim u određenim vremenskim intervalima obnavlja sve certifikate na serveru. Certbot iznimno dobro radi s alatom Nginx te prepoznaje adrese koje su omogućene, prolazi kroz njihove konfiguracijske datoteke te ih nadopunjuje sigurnosnim postavkama kako bi promet na aplikaciji bio siguran.

8.5.1. Instalacija alata Certbot

Za instalaciju Certbota potrebno je na server instalirati snapd koji omogućava preuzimanje i instalaciju programa koji su unutar kontejnera. Za instalaciju snapda potrebno je pokrenuti naredbu koja će instalirati snapd te naredbu koja će provjeriti podudaraju li se instalirana i posljednja dostupna verzija. Potom je potrebno pokrenuti naredbu koja pokreće instalaciju Certbota te naredbu koja kreira poveznicu na taj alat. Postupak instalacije alata dostupan je na službenoj stranici alata [6].

Programski kod 8.12: Instalacija Certbota

```
sudo apt install snapd
sudo snap install core
sudo snap refresh core
sudo snap install —classic certbot
sudo ln -s /snap/bin/certbot /usr/bin/certbot
```

8.5.2. Kreiranje HTTPS certifikata

Nakon uspješne instalacije Certbota, HTTPS certifikat moguće je kreirati naredbom:

Programski kod 8.13: Korištenje Certbota za izradu HTTPS certifikata

```
sudo certbot —nginx
```

Prilikom pokretanja naredbe za postavljanje certifikata, Certbot će postaviti pitanje za koje je domene potrebno postaviti certifikat. Ostavi li se ta opcija neispunjenom, Certbot će

postaviti certifikate na sve domene i poddomene koje su postavljene putem Nginxa. Nadalje, Certbot će tražiti adresu e-pošte na koju može javiti ako bude problema s certifikatom te treba li sav promet s HTTPa prosljeđivati na HTTPS protokol. Potrebno je izabrati opciju da se prosljeđuje promet te će Certbot proći kroz sve konfiguracijske datoteke i dodati dodatne postavke. Primjer izgleda konfiguracijske datoteke koja se kreira pri postavljanju poddomene nakon kreiranja HTTPS certifikata nalazi se u programskom kodu 8.14.

Programski kod 8.14: Nginx konfiguracija poddomene nakon izdavanja HTTPS certifikata

```
server {
  server_name money-manager.petar-cv.com www.money-manager.petar-cv.com;

   proxy_set_header X-Forwarded-For $proxy_protocol_addr; # To forward the original client's IP address
   proxy_set_header X-Forwarded-Proto $scheme; # to forward the  original protocol (HTTP or HTTPS)
   proxy_set_header Host $host; # to forward the original host requested by the client

  location / {
    proxy_pass http://localhost:4200;
  }

  location /auth {
    proxy_pass http://localhost:8080;
  }

  location /api {
    proxy_pass http://localhost:3333;
  }

   listen 443 ssl; # managed by Certbot
```

```
    ssl_certificate /etc/letsencrypt/live/money-manager.petar-cv.com/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/money-manager.petar-cv.com/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}
```

---

9. BUDUĆNOST APLIKACIJE

9.1. Nove funkcionalnosti

Kako se aplikacija fokusira na osobne financije, postoje mnoge funkcionalnosti koje bi odlično služile korisnicima. Neke od tih funkcionalnosti su praćenje vlastitih kredita i rokova otplate, praćenje investicijskih pozicija, uplata u mirovinske fondove, predviđanje prihoda za tekuću godinu, praćenje prihoda za obrt (knjiga prometa) i slično.

9.2. Arhitektura

9.2.1. Mikroservisna arhitektura

Bitna razlika između trenutne monolitne i mikroservisne arhitekture je način na koji se razdvajaju logičke cjeline aplikacije. Primjerice, ako se želi implementirati funkcionalnost praćenja vlastitih kredita, potrebno je odvojiti svu tu funkcionalnost u zaseban servis. Taj servis imat će vlastitu bazu podataka i vlastiti dio koji komunicira s tom bazom podataka. S ostatkom servisa unutar aplikacije, ovaj će servis komunicirati putem posrednika poruka kao što su alati zvani Kafka ili RabbitMQ.

Vrijedi napomenuti kako je cilj imati što manje međusobne povezanosti između servisa. Da bi se ostvarila ta neovisnost servisa, potrebno je napraviti dobro razdvajanje na temelju logičkih cjelina. U ovom primjeru, to je vrlo lako napraviti jer praćenje troškova kreditnih kartica i praćenje otplate kredita nemaju očitih zajedničkih cjelina. Osim, naravno, servisa za autentikaciju koji je svima zajednički.

Implementacija mikroservisne arhitekture omogućava bolje performanse, otpornost na greške i dodavanje novih funkcionalnosti kako raste veličina same aplikacije.

Prednosti ovakvog pristupa uključuju otpornost sustava na probleme tijekom rada. Primjerice, ako servis za praćenje troškova kreditnih kartica prestane s radom, ostatak aplikacije može nesmetano raditi. Još jedna prednost bila bi da se prilikom skaliranja, baze podataka pune sporije nego kod monolitne arhitekture.

Mikroservisna arhitektura omogućava daljnje napredovanje u samoj arhitekturi. Koncepti poput balansiranja prometa, multipliciranje servera i sličnih, vrlo su popularni unutar ove vrste arhitekture

Veliki nedostatak je održavanje samog servera i svih dijelova aplikacije. Pokretanje nekoliko bazi podataka zahtjeva mnogo više računalne snage u usporedbi s jednom bazom podataka koja sadrži sve podatke. Vrijedi spomenuti i kako s većim brojem servisa opada i jednostavnost razvoja. Programer u ovakvom načinu rada mora moći podići mnoge servise na svom računalu što definitivno povećava cijenu računala za razvoj i povisuje donju granicu iskustva potrebnog za razvoj aplikacija. Drugim riječima, mnogim početnicima u programiranju, ovakav pristup otežava razvoj.

10. ZAKLJUČAK

Internetske aplikacije su proteklih godina osvojile svijet razvojnih programera svojom jednostavnošću, skalabilnošću i lakoći isporuke programskog rješenja. Rast broja korisnika pratio je i nevjerojatan napredak unutar tehnologija koje se koriste za kreiranje internetskih aplikacija. Današnje internetske aplikacije koje svakodnevno koriste milijuni korisnika toliko su kompleksne da na njima radi nekoliko desetaka arhitekata programskih rješenja. Implementacije su sve brže i svakim se danom povećava broj alata koji se koristi. Prije samo desetak godina, internetske aplikacije često su se pokretale na samo jednom serveru te velik dio njih nije niti imao dio aplikacije koji se pokreće na klijentovom računalu. Internetske stranice često su bile statične s vrlo ograničenim funkcionalnostima što je kao rezultat imalo veliku rasprostranjenost aplikacija koje se moraju instalirati na svaki uređaj posebno.

Velik broj kreativnih individualaca zaslužno je za tisuće alata koji su otvorenog tipa programskog rješenja te besplatni za korištenje. Neusporedivo velik broj tih alata namijenjen je isključivo za internetske aplikacije. Zajednica programera koji razvijaju internetske aplikacije jedna je od najvećih unutar svijeta razvoja aplikacija, te samim time nije ni čudno što iz te zajednice dolaze impresivna rješenja za gotovo sve probleme na koje razvojni programeri nailaze.

Rastom broja alata koji su potrebni za implementaciju optimalnog rješenja u skladu sa zahtjevima korisnika, povećava se i potreba za arhitektima programskog rješenja. Jedan od njihovih glavnih zadataka je donositi odluke koji će se alati koristiti za koju primjenu. Takve odluke su vrlo teške te često mogu imati loše posljedice. Stotine sati razvoja mogu biti izgubljene donese li arhitekt krivu odluku koja će se kasnije morati revidirati. Samim time, u modernom se svijetu internetskih aplikacija sve više govori o arhitekturi istih i sve se više vremena ulaže u dobru procjenu svih potreba koje klijenti mogu imati.

Internetske aplikacije još uvijek imaju razne prepreke na svom putu do šire rasprostranjenosti. No, ovom brzinom razvoja, sve je manje ljudi koji misle da internetske aplikacije nisu sadašnjost i budućnost razvoja aplikacija namijenjenih za velik broj korisnika.

11. LITERATURA

[1] Službena NestJS dokumentacija, https://docs.nestjs.com/, 8.10.2022.

[2] Službena Prisma dokumentacija, https://www.prisma.io/docs/, 8.10.2022.

[3] Službena Docker dokumentacija, https://docs.docker.com/, 8.10.2022.

[4] Službena dokumentacija platforme Hetzner, https://docs.hetzner.com/, 8.10.2022.

[5] Službena dokumentacija alata Nginx, https://nginx.org/en/docs/, 8.10.2022.

[6] Službena dokumentacija alata Cerbot, https://eff-certbot.readthedocs.io/en/stable/, 8.10.2022.

12. OZNAKE I KRATICE

B2B - Business to business

CSS - Cascading Style Sheets

DNS - Domain Name System

GB - Gigabyte

GNU - GNU's not Unix

GPG - GNU Privacy Guard

HTML - HyperText Markup Language

HTTP - Hypertext Transfer Protocol

HTTPS - Hypertext Transfer Protocol Secure

IP - Internet Protocol

JS - JavaScript

JSON - JavaScript Object Notation

JWT - JSON Web Token

MVC - Model View Controller

TB - Terabyte

TS - TypeScript

TTL - Time to Live

UUID - Universally Unique Identifier

vCPU - Virtual Central Processing Unit

YAML - Yet Another Markup Language

13. SAŽETAK

Arhitektura modernih web aplikacija:

Završni rad opisuje proces kreiranja arhitekture za internetsku aplikaciju koja mora biti skalabilna i modularna. Aplikacija posjeduje razne programske tehnike spremne za produkcijsku okolinu i koristi razne koncepte koji se mogu pronaći u aplikacijama koje se koriste na razini poduzeća. Neki od koncepata su autorizacija, kontrola pristupa temeljena na ulogama, kontinuirana integracija i implementacija, rad s bazama podataka, objavljivanje aplikacija te mnogi drugi česti koncepti unutar internetskih aplikacija. Tehnološki skup uglavnom se sastoji od Angulara, NestJSa i PostgreSQLa te pokazuje kako moderne tehnologije za razvoj web aplikacija rade zajedno kako bi postigle različite funkcionalnosti. Razne druge tehnologije kao što su Docker, Nginx, Keycloak, Prisma, HTML, CSS itd. pomažu programerima u implementaciji željenih funkcionalnosti.

Keywords: internetska aplikacija, autorizacija, mikroservisi

14. ABSTRACT

Architecture of modern web applications:

The final thesis describes the process of creating an architecture for a web application that's required to scale and be modular. The application contains various production grade code practices and covers multiple topics which are found in enterprise level applications. These topics consist of authorisation, role-based access control, continuous integration and deployment, working with databases, publishing applications and many more common web development topics. The technology stack mainly consists of Angular, NestJS and PostgreSQL and shows how modern web development technologies work together to achieve various functionalities. Various other technologies such as Docker, Nginx, Keycloak, Prisma, HTML, CSS, etc. help the developers implement the wanted functionalities.

Keywords: web application, authorisation, microservices

IZJAVA O AUTORSTVU ZAVRŠNOG RADA

Pod punom odgovornošću izjavljujem da sam ovaj rad izradio/la samostalno, poštujući načela akademske čestitosti, pravila struke te pravila i norme standardnog hrvatskog jezika. Rad je moje autorsko djelo i svi su preuzeti citati i parafraze u njemu primjereno označeni.

Mjesto i datum	Ime i prezime studenta/ice	Potpis studenta/ice
U Bjelovaru, 14.10.2022.	Petar Cvetko Voćanec	Petar Cvetko Voćanec

Prema Odluci Veleučilišta u Bjelovaru, a u skladu sa Zakonom o znanstvenoj djelatnosti i visokom obrazovanju, elektroničke inačice završnih radova studenata Veleučilišta u Bjelovaru bit će pohranjene i javno dostupne u internetskoj bazi Nacionalne i sveučilišne knjižnice u Zagrebu. Ukoliko ste suglasni da tekst Vašeg završnog rada u cijelosti bude javno objavljen, molimo Vas da to potvrdite potpisom.

Suglasnost za objavljivanje elektroničke inačice završnog rada u javno dostupnom nacionalnom repozitoriju

Petar Cvetko Vočanec

ime i prezime studenta/ice

Dajem suglasnost da se radi promicanja otvorenog i slobodnog pristupa znanju i informacijama cjeloviti tekst mojeg završnog rada pohrani u repozitorij Nacionalne i sveučilišne knjižnice u Zagrebu i time učini javno dostupnim.

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada.

U Bjelovaru, 14.10.2022.

Petar Cvetko Vočanec

potpis studenta/ice