Razvoj web aplikacija u programskom jeziku Ruby on Rails

Vrđuka, Robert

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Robert Vrđuka

Razvoj web aplikacija u programskom okviru Ruby on Rails

ZAVRŠNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Robert Vrđuka

Matični broj: 0016148751

Studij: Informacijski sustavi

Razvoj web aplikacija u programskom okviru Ruby on Rails

ZAVRŠNI RAD

Mentor:

doc. dr. sc. Matija Novak

Varaždin, rujan 2023.

Robert Vrđuka

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Ovaj radi bavi se programskim jezikom Ruby, programskim okvirom baziranom na Rubyju Rails te primjenom tog okvira na stvaranje konkretne web aplikacije. Prvo se polazi od samog Rubyja. Tu se razmatraju osnove sintakse te neke osnovne i napredne mogućnosti jezika. Zatim se opisuje Rails te dijelovi njegove arhitekture koja je primjena MVC arhitekture. Na kraju se programski okvir primjenjuje u konkretnoj web aplikaciji. Web aplikacija bi služila obrtnicima kao centralni dio njihovog poslovanja. Neke funkcionalnosti aplikacije su registracija i login, stvaranje popisa usluga, stvaranje računa te slanje istoga na mail klijenta i dr.

Ključne riječi: Ruby, Rails, web, programiranje, aplikacija, Internet,

Sadržaj

1. Uvod

Web stranice i web aplikacije jedne su od najkorištenijih vrsta softvera uopće. Prema stranici *Statista*, otprilike dvije trećine svjetske populacije koristi internet [1]. To znači da ta populacija koristi i određene web aplikacije. Web aplikacije nose mnoge prednosti, od kojih su neke nepotrebnost instalacije od strane korisnika, jednostavnija isporuka te istovremeni pristup aplikaciji sa osobnih računala, tableta i mobilnih uređaja.

Kako bi razvoj web aplikacija bio što jednostavniji i efikasniji, osmišljeno je mnogo programskih okvira koji se temelje na raznim programskim jezicima. Web programski okviri dijele se na okvire za klijentski dio web aplikacije (eng. *frontend*), pozadinski dio (eng. *backend*) te okvire koji obuhvaćaju oboje (eng. *full stack*). Jedan od *full stack* programskih okvira je i Ruby on Rails, koji je tema ovog rada.

U prvom dijelu rada opisuju se općenite značajke web aplikacija. Zatim se opisuju značajke programskoga jezika Ruby, odnosno daje se uvod u jezik koji je omogućio postojanje okvira Rails. Nakon toga ulazi se u sam Rails, dijelove njegove arhitekture, te način kako implementirati pojedine dijelove. Na kraju se teorijske značajke primjenjuju u izradi web aplikacije koja služi obrtniku kao centralni dio njegovog poslovanja, odnosno omogućuje popis klijenata, usluga, izradu računa i drugo.

2. Web aplikacije

Kako bi web aplikacije kao takve postojale, jedan od preduvjeta bio je razvoj arhitekture koja bi mogla povezati računala diljem svijeta. Ta arhitektura je na kraju dobila ime Internet. Osim samog interneta, bio je potreban način dijeljenja informacija preko interneta. To je na kraju postao WWW, odnosno World Wide Web.

World Wide Web koncipirao je Tim Berners-Lee 1989. godine u CERN-u. Stvorili su HyperText Transfer Protocol (HTTP) koji je standardizirao komunikaciju između klijenta i poslužitelja. Njihov internetski preglednik baziran na tekstu postao je dostupan u siječnju 1992. godine [2]

Popularnost Weba počela je rasti kako su se pojavljivali bolji preglednici. Preglednik Mosaic jedan je od prvih popularnih preglednika. Neke njegove funkcionalnosti uključuju strelice za vraćanje na prethodne stranice, knjižne oznake (eng. bookmarks), raznolike fontove, te, možda i najvažniju inovaciju u to vrijeme, mogućnost istodobnog prikazivanja fotografija i teksta. [3]

Web se može podijeliti na tri glavne verzije: Web 1.0, Web 2.0 i Web 3.0

Web 1.0 uključuje stranicu koja se može pregledavati te sadrži određene podatke WWW-a [4]. Glavne tehnologije Weba 1.0 bile su HTML, HTTP i URI. HyperText Markup Language, odnosno HTML glavni je jezik Weba. On služi kako bi preglednik mogao prikazati određeni sadržaj korisniku. HTTP protokol je protokol koji omogućuje komunikaciju između klijenta, najčešće preglednika, te poslužitelja koji sadrži datoteke. Uniform Resource Identifier (eng. URI) je način označavanja resursa na webu.

Web 2.0 nadograđuje koncepte prošle verzije na način povećane interakcije samih korisnika. Dok su u Web 1.0 samo određeni ljudi, najčešće vlasnici web mjesta ili administratori mogli dodavati novi sadržaj ili ažurirati postojeći, u ovoj verziji ga mogu dodavati i sami korisnici. Web 2.0 ponajviše su obilježile društvene mreže, stranice koje predstavljaju zajednicu ljudi koju mogu dijeliti poruke, informacije te sudjelovati u zajedničkim aktivnostima. [5]

Web 3.0 je najnovija evolucija Weba te se temelji na tehnologiji ulančanih blokova (eng. blockchain). Najveća prednost Weba 3.0 je decentralizacija, što omogućuje korisnicima da posjeduju dijelove interneta, umjesto da koriste internet uz pomoć servisa velikih tvrtki, poput Google-a, Apple-a ili Facebook-a [6]. Danas u koncept Weba 3.0 najviše vjeruje zajednica koja vjeruje u kriptovalute, nezamjenjive tokene (eng. non-fungible token, odnosno NFT), ali je danas ova verzija više konceptualna nego što je primijenjena.

2.1. Osnovno o web aplikacijama

Web aplikacija može se definirati kao računalni program koji koristi internetske preglednike i web tehnologije kako bi izvršavao zadatke preko interneta [7]

Osnovni dijelovi web aplikacije sastoje se od frontenda i backenda. Frontend obuhvaća sloj koji je u izravnom doticaju s korisnikom. Osnovne tehnologije koje se koriste u frontendu su HTML, CSS i JavaScript. HTML služi za prikaz željenog sadržaja, CSS taj sadržaj oblikuje, odnosno daje nam mogućnost dizajna, a JavaScript je zaslužan za interakciju korisnika s aplikacijom. Backend je sloj gdje se nalaze poslovna logika i baza podataka aplikacije. Kada korisnik npr. klikne neki gumb, taj gumb najčešće nešto šalje ili dohvaća s backenda. Situacija s jezicima je drukčija nego s klijentskom stranom. Backend poslužitelj može biti pisan u raznim jezicima, npr. JavaScript, Java, C#, PHP, Ruby i drugima.

Osnovni tok rada web aplikacije je sljedeći:

1. Korisnik otvara određenu aplikaciju. Važno je napomenuti kako korisnika ne zanima kako pozadinska strana radi, već samo klijentska.
2. Korisnik ostvaruje interakciju s aplikacijom te tako šalje određeni zahtjev poslužitelju klijentske strane.
3. Poslužitelj klijentske strane šalje određeni zahtjev poslužitelju pozadinske strane aplikacije.
4. Nakon što klijentska strana dobije odgovor, ona taj odgovor formatira i prikazuje korisniku.

Web aplikacije kao takve imaju i svoje prednosti, od kojih su neke:

1. Kompatibilnost sa svim platformama. Zbog toga što aplikacija koristi internetski preglednik za prikaz, ona može raditi istovremeno na bilo kojem uređaju koji može pokrenuti internetski preglednik, npr. mobilni uređaj, tablet, osobno računalo i drugi. Ipak, valja imati na umu kako programer mora prilagoditi dizajn aplikacije za razne veličine ekrana.
2. Jednostavnija isporuka. Budući da korisnik ne treba instalirati aplikaciju, dovoljno je da se aplikacija nalazi na jednom ili više poslužitelja, što olakšava verzioniranje, odnosno korisnik ne mora pratiti je li došlo određeno ažuriranje.
3. Teža krađa, odnosno piratiziranje aplikacije, zato što klijent nema datoteke koje mora instalirati ili pokrenuti.

2.2. Jezici za web

S obzirom na ogromnu količinu web aplikacija danas, nije teško pretpostaviti kako većina programskih jezika ima određene mogućnosti weba, bilo da je sam jezik tako koncipiran (npr. PHP), ili da postoji okvir namijenjen za izradu web aplikacija u praktički bilo kojem programskom jeziku.

Neki od najpopularnijih jezika koji se koriste na poslužiteljskoj strani weba su PHP, C#, Java, JavaScript (iako je originalno namijenjen za klijentsku stranu, danas se može koristiti i na poslužitelju) te Ruby. S druge strane, klijentska strana weba je zapečaćena u HTML-u, CSS-u i JavaScriptu.

Iako su popularni jezici za klijentsku stranu stabilni, ne znači da ne dolaze i novi. Jezici poput Elixira, Golanga, Kotlina i ostalih možda još nisu popularni, ali ne znači da nisu pogodni za razvoj web aplikacija

2.2.1. PHP

PHP, ili punog imena „PHP: Hypertext Preprocessor“ je jezik otvorenog koda koji je namijenjen za razvoj web aplikacija. [8]

Jezik je stvorio Rasmus Lerdorf 1994. godine. Isprva je bio namijenjen za Rasmusovu osobnu upotrebu, ali je s vremenom nadišao tu namjenu te postao jednim od najkorištenijih programskih jezika za web. S godinama su izlazile nove verzije jezika, a svaka je donosila poboljšanja u performansama, robusnosti, novim funkcionalnostima i sličnome. Danas je jezik na verziji 8.

Jednim dijelom PHP postiže svoju popularnost zbog WordPressa, sustava za upravljanje sadržajem (eng. content management system, ili kraće CMS) koji je baziran na PHPu. Sustav za upravljanje sadržajem je softver koji omogućuje korisnicima stvaranje, upravljanje i mijenjanje sadržaja web stranice bez potrebe za tehničkim znanjem. [9] Glavna prednost CMS-a je jednostavnost upotrebe te brzina razvoja, zato što su glavni infrastrukturni dijelovi integrirani u sustavu. Ipak, takvi sustavi imaju i svoje nedostatke. Neki nedostaci su velika potreba za vanjskim paketima, od kojih se neki plaćaju ili nisu održavani, te otežana prilagodba za kompleksnije stranice.

2.2.2. Java

Java je objektno orijentirani programski jezik opće namjene te softverska platforma koja može biti pokrenuta na milijardama uređaja, uključujući osobna računala, mobilne uređaje, konzole za videoigre, medicinski uređaji i drugo. [10]

Najveća prednost jave je njezina portabilnost. Svaki uređaj koji podržava JVM (Java Virtual Machine) ima mogućnost pokretanja identičnog koda. Kako bi se program mogao pokretati u JVM-u, on se mora moći kompilirati u „bytecode“, odnosno jedini jezik koji JVM razumije. To omogućava uređajima da ne moraju znati kako radi Java, već samo bytecode, što olakšava portabilnost. Još jedna prednost ovakvog sustava je i moguća promjena jezika. Danas se u bytecode kompiliraju i Groovy, Kotlin, Scala i drugi. U teoriji to znači kako bi u istom projektu mogli izmjenjivati ove jezike zato što se svi kompiliraju u isti bytecode.

Danas se Java najviše koristi u velikim tvrtkama koje, iznad svega, zahtijevaju stabilnost jezika. Java je izašla 1996, a razvija se i danas, što smanjuje mogućnost zamiranja jezika. Još jedna pozitivna strana Jave je u količini paketa i biblioteka koje ubrzavaju i olakšavaju razvoj.

2.2.3. C#

Slično kao i Java, C# je objektno orijentirani programski jezik opće namjene. Zbog sličnosti s Javom mnogi ga smatraju i njezinom alternativnom. Slično kao Javin JVM, C# posjeduje CLR, gdje također postoji međukorak između kompiliranja i izvršavanja koda.

Kada govorimo o C#, s njim u paketu najčešće dolazi .NET platforma. .NET je platforma otvorenog koda namijenjena razvoju desktop, web i mobilnih aplikacija koje mogu biti pokrenute na bilo kojem operacijskom sustavu. [11] .NET ima puno ogranaka, a onaj namijenjen za web je ASP.NET Core.

ASP.NET Core okvir za razvoj web aplikacija jedan je od najpopularnijih okvira za web uopće. Koristi MVC arhitekturu za razvoj web aplikacija, slično kao i Rails koji će biti obrađen u kasnijim poglavljima.

2.2.4. JavaScript

JavaScript je skriptni programski jezik čija je osnovna namjena implementacija kompleksnih funkcionalnosti na web stranicama. Ako naša stranica ažurira podatke u realnom vremenu, prikazuje grafiku i slično, možemo biti sigurni kako je JavaScript uključen [12]

Povijest JavaScripta počinje 1995. kada ga je u 10 dana napravio Brendan Eich za internetski preglednik Netscape Navigator. Namjena jezika bila je od početka klijentska strana web stranica. Budući da je bio jedini takav jezik, brzo je stekao popularnost među web programerima. [13]

Iako je prvotno namijenjen za klijenta, se danas može pokretati i na poslužitelju uz pomoć Node.js-a. To je omogućilo stvaranje web aplikacija gdje poslužiteljska i klijentska strana koriste JavaScript.

2.2.5. Usporedbe

Iako se svaki od prije navedenih jezika može koristiti na webu, svaki od njih ima neka obilježja kojeg ga izdvajaju od ostalih. To može biti u obliku sintakse, broja poslova u tom jeziku, jednostavnosti učenja i ostalog.

Tablica 1 Usporedba programskih jezika za web

	Programska paradigma	Kompilacija	Tipizacija	Performanse	Podrška za više dretvi	Popularnost
Ruby	Objektno orijentirani	Interpretirani	Dinamična	5.	Da	5.
PHP	Miješano	Interpretirani	Dinamična	4.	U obliku ekstenzije	4.
C#	Objektno orijentirani	Kompilirani	Statička	1.	Da	2.
Java	Objektno orijentirani	Kompilirani	Statička	1.	Da	1.
JavaScript	Miješano	Interpretirani	Dinamična	3.	Da	3.

Programska paradigma je temeljan način rješavanja problema u nekom jeziku. Svi od navedenih jezika su ili objektno orijentirani ili imaju mogućnost korištenja klasa i objekata. Danas su čisto funkcijski jezici rijetka pojava.

Osobne preferencije postoje i u načinu tipizacije. Ruby, PHP i čisti JavaScript su dinamično tipizirani, dok su C# i Java strogo tipizirani. Iako je JavaScript dinamične tipizacije, TypeScript, jezik koji se kompajlira u JavaScript zahtijeva tipove te dobiva sve veću popularnost. Iako je s dinamično tipiziranim jezicima jednostavnije raditi na prvu, mnogi smatraju kako uvođenjem tipova dobivamo na strukturi te jednostavnijem snalaženju u većim projektima.

Performanse označavaju korištenje procesorske snage i memorije pri izvođenju nekog zadatka. Prema GitHub repozitoriju koji mjeri brzinu jezika pri izračunu broja pi, Java je najbrži jezik, dok je Ruby na zadnjem mjestu. [14] Promatrajući dodatne izvore, rezultati su isti, jedino što se u određenim zadacima izmjenjuju Java i C#. Iako je Ruby najsporiji, tim zadužen za dodavanje novih značajki rade na performansama u obliku „just-in-time“ kompajlera.

Važna značajka pri odabiru jezika je i njegova popularnost. Prednosti korištenja popularnijeg programskog jezika je veći broj biblioteka i okvira, lakše pronalaženje informacija te veća zajednica. Prema TIOBE indeksu, Java je najpopularniji jezik, dok je Ruby na zadnjem mjestu. Neki od uzroka Javine popularnosti su zastupljenost u velikim tvrtkama te mogućnost razvoja za bilo koju platformu. Ruby, uspoređujući s Javom koja se koristi za desktop, web i mobilne aplikacije, poznat je po okviru Rails te se danas koristi najviše u kontekstu web aplikacija. Slična je situacija i s PHP-om. Na trećem mjestu stoji JavaScript zato što je on

standardan jezik weba. Svaka web aplikacija u klijentskom dijelu mora sadržavati JavaScript, ali je jezik za poslužiteljski dio proizvoljan. Slična situacija je i s PYPL indeksom. Zadnja dva mjesta zauzimaju Ruby i PHP, ali se prva tri jezika izmjenjuju u mjestima.

Java i C# imaju daleko najveće mogućnosti bez korištenja biblioteka treće strane. .NET platforma te Java Standard Edition imaju veliki broj ugrađenih funkcionalnosti koje bismo u drugim jezicima morali tražiti u obliku vanjskih paketa. To je moguće zbog količine resursa koje velike tvrtke, odnosno Microsoft i Oracle ulažu u svoje jezike. S druge strane, PHP, Ruby i JavaScript više ovise o samoj zajednici, gdje određena osoba napravi implementaciju za neki problem te ju odluči podijeliti s drugima.

Dokumentacija je također bitna značajka jezika. Dokumentacija služi kao osnovna literatura pri pronalaženju rješenja za neki problem, ili za informacije o sintaksi, metodama, klasama i sličnome. Osobno, najjednostavnije mi je pronaći informacije u dokumentaciji Rubyja. Dokumentacija se jednostavno pretražuje, a radi na način da svaka klasa ima popis metoda. PHP-ova dokumentacija sadržava komentare, za što smatram da nije dobro zato što bi dokumentacija trebala proizlaziti od ljudi koji su radili na jeziku, a ne od bilo kojeg korisnika. Za ostale jezike se ne može reći kako imaju lošu dokumentaciju. Svaka ima sadržaj koji uključuje opise i primjere korištenja.

Velike razlike između jezika su platforme za koje je određeni jezik namijenjen. Ruby, PHP i JavaScript su ponajviše korišteni za web aplikacije. Postoje projekti kojima je cilj proširiti korištenje jezika i na druge platforme, ali su oni još uvijek najpoznatiji u webu. S druge strane, Java i C# su korišteni za više platforma. C# uz pomoć .NET platforme je korišten u desktop aplikacijama, dok se Java koristi i za desktop i za mobilne aplikacije. Naime, Java te Kotlin, koji se također kompilira u JVM, nativni su jezici za razvoj Android aplikacija.

Iako ovi jezici dijele neka temeljna obilježja, velike razlike su u sintaksi. Načini alociranja varijabli, stvaranja funkcija ili metoda, petlji i ostalog su stvari gdje svaki programer može imati osobnu preferenciju. Osobna preferencija može proizaći i iz okvira za razvoj koji svaki jezik nudi. Budući da su svi navedeni jezici dosta popularni, svaki nudi više okvira za razvoj web aplikacija. Za ovaj rad sam odabrao Ruby zbog intuitivne sintakse te okvira Rails koji omogućuje visoku razinu produktivnosti zbog svoje jednostavnosti. Osim toga, Ruby i Rails se koriste dugi niz godina, što se vidi u količini informacija o njima te broju paketa za specijalizirane probleme.

3. Ruby

Ruby je interpretirani programski jezik opće namjene kojeg je dizajnirao Yukihiro “Matz” Matsumoto, a izašao je 1995. [15]

Cilj ili fokus Rubyja je optimizirati odnosno povećati razinu sreće programera dok programira (eng. programmer happiness). Ako uzmemo poznati citat Martina Fowlera iz knjige „Refactoring: Improving the Design of Existing Code“: „Svaka budala može pisati kod koji računalo razumije. Dobri programeri pišu kod koji razumiju ljudi.“ cilj Rubyja je taj proces maksimalno olakšati.

Ruby je objektno orijentirani jezik, ali u tolikoj razini da je sve objekt, stoga su sve funkcije zapravo metode. Čak i primitivni tipovi poput integera, stringova i slično su objekti, što nije slučaj u većini programskih jezika.

Ruby je također i dinamično tipizirani jezik, što znači da nema striktnih tipova podataka pri npr. inicijalizaciji varijable. Razlika između Rubyja i ostalih programskih jezika je u tome što za inicijalizaciju varijable ne postoji nikakva ključna riječ (npr. let ili var u JavaScriptu).

3.1. Osnovne mogućnosti

3.1.1. Tipovi podataka

Iako je Ruby programski jezik dinamične tipizacije, to ne znači u pozadini nema tipove podataka. Kako je u Rubyju sve objekt, svaki tip podatka ima i svoju klasu, zajedno s raznim metodama koje mogu biti korisne. Najčešće korišteni tipovi podataka su Array, Number (Float ili Integer), Symbol, String, Booleans i Hash. Većina ovih tipova nam je poznata, ali klase Symbol i Hash valja detaljnije objasniti.

3.1.2. Simboli

Simboli su instance klase Symbol te ih raspoznajemo uz pomoć prefiksne dvotočke [15].

```
String1 = „string“

Simbol1 = :simbol

Simbol2 = :“simbol s vise rijeci“
```

Iako na prvu simboli nalikuju na stringove zato što sadržavaju slova, postoje i neke važne razlike, za koje bismo rekli da su svojstveniji klasama koje predstavljaju brojeve:

- Nepromjenjivost – simboli kao takvi se ne mogu mijenjati. Ako imamo simbol „:abc“ te tom simbolu pokušamo dodati drugi simbol, dobit ćemo potpuno drugi objekt.
- Jedinstvenost – ako određeni simbol ima određeni sadržaj (npr. :abc iz gornjeg primjera), on će uvijek i bilo gdje imati isti „object_id“. Object_id je metoda koju možemo pozvati na bilo koji objekt te predstavlja jedinstveni broj koji služi za identifikaciju. S druge strane, ako više puta napravimo string s istim sadržajem, object_id će svaki puta biti drukčiji.

3.1.3. Rječnici

Rječnik (u Rubyju Hash, u c# Dictionary) je oblik podataka sličan polju na način da predstavlja kolekciju podataka, ali je razlika što je ključ praktički bilo koji objekt [15]. U Rubyju se rječnici označavaju vitičastim zagradama, a samo stvaranje rječnika izgleda vrlo slično JSON sintaksi.

```
h = { foo: 0, bar: 1 }
```

U navedenom primjeru možemo vidjeti primjer stvaranja rječnika, gdje su ključevi „foo“ i „bar“ simboli, a vrijednosti mogu biti bilo koje. Zbog navedenih svojstava simbola preporuča se koristiti simbole kao ključeve u usporedbi sa stringovima.

Pristup podacima u rječniku se postiže korištenjem uglatih zagrada gdje upišemo željeni ključ, slično kao u jeziku C#. Klasa Hash također sadržava i metode za apsolutno svaku radnju koja bi nam mogla zatrebati, što uključuje pretraživanje, uspoređivanje, dohvaćanje, dodavanje, iteriranje i drugo.

3.1.4. Varijable

U Rubyju se varijable inicijaliziraju samo s imenom, odnosno nema ključnih riječi kao u drugim jezicima. Ruby razlikuje 4 vrsta varijabli:

- Lokalne varijable (eng. local variables). Te varijable su najčešće korištene te je njihov doseg u pojedinom bloku naredbi (npr. određenoj metodi).
- Varijable instance (eng. instance variables). Te varijable služe za spremanje informacija o pojedinom objektu, odnosno imaju ulogu atributa pri stvaranju objekta određene klase. Za stvaranje ove varijable ispred imena stavljamo prefiks „@“.

- Klasne varijable (eng. class variables). Ove varijable imaju ulogu statičkih atributa u klasi. To znači da ne ovise o pojedinom objektu, već da svi objekti iste klase mogu pristupiti vrijednosti ove varijable. Za stvaranje ove varijable ispred imena stavljamo prefiks „@@“.
- Globalne varijable (eng. global varijables). Kako možemo zaključiti iz imena, ove varijable su vidljive svugdje. Ove varijable se preporuča izbjegavati zbog smanjene čitljivosti koda. Za stvaranje ove varijable ispred imena stavljamo prefiks „$“.

Ruby ima i određene konvencije imenovanja varijabli. Sve varijable osim globalnih se pišu malim slovima, a ako se ime varijable sastoji od više riječi, riječi odvajamo donjom crtom (snake_case stil pisanja). U slučaju globalnih varijabli sve riječi su pisane velikim slovima. Neki primjeri imenovanja varijabli su:

```
first_name

@first_name

@@first_name

$FIRST_NAME
```

3.1.5. Ulaz i izlaz

Osnovni ulaz i izlaz u Rubyju su dosta jednostavni. Osnovna naredba za ulaz je naredba „gets“, dok je osnovna naredba za izlaz „puts“. Korisna stvar kod naredbe „puts“ je što na ekran vrati i povratnu vrijednost koju želimo prikazati, što može olakšati debagiranje.

```
puts 'hello'

hello

=> nil
```

3.1.6. Grananja

Osnovno grananje, kao i u većini programskih jezika, sastoji se od „if-else“ bloka koji izgleda ovako:

```
if uvjet1
# kod koji se izvrsava ako je uvjet1 tocan
elsif uvjet2
# kod koji se izvrsava ako je uvjet2 tocan
else
# kod koji se izvrsava u ostalim slucajevima
end
```

Posebnosti u Rubyju su mogućnost skraćivanja grananja ako se radi o jednoj liniji, što olakšava pisanje takozvanih „guard clauses“. Sintaksa izgleda ovako:

```
neka_akcija if neki_uvjet
```

Još jedna korisna ključna riječ je „unless“. Ona predstavlja način rada obrnut „if“ grananju, odnosno kod se izvršava samo kada je uvjet netočan. Osim „if“ naredbi, Ruby još podržava i „switch-case“ grananje.

3.1.7. Petlje

Ruby sadržava pregršt metoda koje se koriste za petlje, od kojih je najjednostavnija metoda „loop“. „loop“ ne zahtijeva nikakve argumente, već samo izvršava skup instrukcija u bloku. S obzirom da bi se te instrukcije mogle izvršavati beskonačno mnogo puta, u „loopu“ je bitno uključiti uvjet za izlazak iz petlje. [15]

Nadogradnja na osnovni „loop“ mogu biti „while“ i „until“ petlje, koje rade istu stvar, ali na obrnuti način. „While“ petlja, slično kao u mnogim programskim jezicima, izvršava određeni blok naredbi dok je uvjet valjan. S druge strane, „until“ izvršava blok naredbi dok uvjet ne postane istinit, odnosno dok je lažan.

Ruby sadržava i klasičnu „for“ petlju, ali se njezino korištenje izbjegava zato što najčešće postoji prikladnija metoda.

Ako znamo koliko puta se određeni blok mora izvršiti, možemo koristiti „times“ petlju. Ona funkcionira na način da određeni broj puta nešto izvrši.

3.1.8. Metode

Budući da je Ruby objektno orijentiran jezik, svaka funkcija je zapravo metoda. Metode pozivamo s operatorom točke, gdje je metoda s desne strane točke, a objekt s lijeve strane. [15]

```
'11'.to_i
```

Naravno, možemo definirati i svoje metode. Početak metode mora početi s ključnom riječi „def“, a završavati s ključnom riječi „end“. Između se nalazi blok naredbi koji se izvršava. Ruby posjeduje i određene konvencije imenovanja metodi. Slično kao i varijable, imena se pišu malim slovima te odvajaju donjom crtom. U odnosu na varijable možemo još koristiti znakove „?“ i „!“. Ako metoda vraća bool vrijednost, imenujemo ju kao pitanje i završavamo upitnikom (npr. is_valid?). Ako metoda mijenja sam objekt nad kojim se poziva, ili je na neki drugi način „opasna“, završavamo ju uskličnikom.

Svaka metoda u Rubyju vraća određenu vrijednost, ali je najveća razlika to što Ruby ima mogućnost implicitne povratne vrijednosti. Ne moramo pisati „return“, već metoda automatski vraća povratnu vrijednost zadnje linije koja se u metodi izvršava. Ipak, ključna riječ „return“ postoji ako želimo iz metode izaći prije izvršavanja cijelog bloka.

3.2. Napredne mogućnosti

3.2.1. Enumerable

Modul Enumerable jedan je od najkorištenijih u Rubyju zato što sadrži mnogo metoda koje su korisne u radu s kolekcijama, odnosno u ovom slučaju poljima i rječnicima. Kako bi određena klasa mogla koristiti metode sadržane u ovom modulu, ona mora definirati metodu „each“. [15]

Metoda „each“ apstrahira najčešće korišteni „for“, odnosno prolazi kroz svaki član kolekcije na način da se izvršava dok je iterator manji od duljine kolekcije, a iterator povećava za jedan. Jednostavni „each“ s ispisom svakog člana polja može izgledati ovako:

```
array = [1,2,3,4,5]

array.each { |element| puts element }
```

U ovom primjeru možemo vidjeti sintaksu pisanja bloka koda s vitičastim zagradama. Ovaj je način pisanja koristan kada se blok sastoji od samo jedne linije. Ako želimo napisati blok od više naredbi, koristimo „do-end“ sintaksu. Iako se sastoji samo od jedne linije, primjera radi možemo gornji primjer napisati i u alternativnom obliku.

```
array = [1,2,3,4,5]

array.each do |element|

  puts element

end
```

Kada je definirana metoda „each“, dobivamo pristup raznim metodama, koje se mogu kategorizirati na sljedeće: [16]

- Metode za upite – ove metode najčešće vraćaju istinu ili laž, a vezane su za informacije o samim elementima u kolekciji. Neke metode su „any?“, „all?“, „none?“, „include?“
- Metode za dohvaćanje – ove metode dohvaćaju određeni element, npr: „first“, „last“, „min“, „max“ i druge.

- Metode za pronalaženje – ove metode također dohvaćaju element, ali po nekom proizvoljnom uvjetu, npr: „find“, „reject“, „uniq“
- Metode za sortiranje
- Metode za iteriranje – ove metode su nadogradnja na osnovnu metodu „each“

Iako su ove metode u modulu Enumerable, postoje i metode u klasama Array i Hash koje obnašaju sličnu funkciju. Dosta tih metoda vuče paralele s jezikom JavaScript, a neke su „map“, „select“, „filter“, „reject“ i druge.

3.2.2. Objektno orijentirano programiranje

Ruby je objektno orijentiran programski jezik, što znači da je sve objekt. Naravno, Ruby posjeduje načine stvaranja proizvoljnih klasa, atributa, odnosa između klasa i dr.

Klasa se stvara s ključnom riječi „class“, nakon čega slijedi proizvoljno ime klase. Konvencije imenovanja kažu kako ime klase treba biti velikim početnim slovom, a ako se ime sastoji od više riječi, koristi se takozvani „CamelCase“ način pisanja.

```
Class TestClass
  # kod
end
```

Jedna od razlika između Rubyja i ostalih programskih jezika je u imenovanju konstruktora. Ako programer želi konstruktor u svojoj klasi, umjesto imena klase koristi se ključna riječ „initialize“, nakon čega najčešće slijedi popis argumenata koji su potrebni po inicijalizaciji. S obzirom da Ruby posjeduje varijable instance, one se koriste kao atributi u klasi.

```
Class TestClass
  def initialize(argument)
        @argument = argument
  end
end
```

Ruby također zahtijeva i eksplicitno napisane „settere“ i „gettere“. Razlikuju se tri ključne riječi: [17]

- attr_reader – ekvivalent „getter“ metodi
- attr_writer – ekvivalent „setter“ metodi
- attr_accessor – „setter“ i „getter“ zajedno, s obzirom da programer najčešće treba oboje

Metode određene klase definiraju se na način koji je opisan u poglavlju o metodama.

Ruby ima mogućnost definiranja i statičkih atributa ili metoda. Kao što je navedeno u poglavlju o varijablama, varijable s prefiksom „@@“ označavaju statičke varijable čijoj vrijednosti mogu pristupati sve instance određene klase.

Statičke metode definiraju se slično kao i obične, ali s prefiksom „self“.

```
Class TestClass

  def self.static_method

          puts 'I am a static method'

  end
```

4. Rails

Rails je okvir za stvaranje web aplikacija pisan u Rubyju. Osmišljen je sa svrhom olakšavanja programiranja web aplikacija zbog pretpostavki koje uključuju sve što programer treba kako bi počeo. Rails je okvir koji pretpostavlja da postoji najbolji način programiranja određenih stvari te ohrabruje programera da se toga drži (eng. opinionated software). [18]

Sama filozofija Railsa sadržava dva važna principa

DRY – (eng. Don't Repeat Yourself) – princip softverskog inženjerstva koji govori kako ne smijemo pisati iste stvari na više mjesta, zato što to dovodi do grešaka te slabije održivosti koda.

Convention Over Configuration – Rails ima konvencije po kojima radi, što olakšava odlučivanje te omogućuje inženjeru da se bavi poslovnom logikom, umjesto konfiguracijom strukture datoteka, baze podataka i slično.

Rails se drži takozvane MVC strukture (sloj modela, upravljača i pogleda, eng. model, view, controller), od kojih svaki sloj ima određenu ulogu: [19]

- Modeli – sloj koji radi s bazom podataka. Sastoji se od konkretnih modela podataka (npr. korisnik, proizvod i sl.) te sadržava poslovnu logiku aplikacije.
- Pogledi – sloj koji se sastoji od predložaka koji su zaslužni za sam prikaz podataka aplikacije. Standardan je način korištenje ERB datoteka, koje su zapravo HTML datoteke proširene Ruby kodom.
- Upravljači – ovaj se sloj bavi odgovaranjem na HTTP zahtjeve. Najčešće se odgovor sastoji od HTML-a, ali može biti i XML, JSON, PDF datoteka i drugo.

Rails je kreirao David Heinemeier Hansson 2004. godine. [20]

Danas je Rails na verziji 7.0 te je i danas jedan od popularnijih okvira za stvaranje web aplikacija. Neke od većih tvrtki koje koriste Rails su GitHub, Airbnb, Kickstarter, Shopify i mnoge druge. [21]

4.1. Modeli

Sloj modela u Railsu implementira objektno relacijski maper Active Record. On se sastoji od objekata koji predstavljaju tablice u bazi podataka te dodatne poslovne logike koja je potrebna za pojedini objekt. [22]

S obzirom da Rails koristi određene konvencije, postoje i konvencije imenovanja modela. Ime modela je jednina imena u tablici. Na primjer, ako je ime tablice „users“, ime modela trebalo bi biti „User“. U slučaju imena s više riječi, koristi se „CamelCase“ način pisanja.

Modeli se najčešće stvaraju zajedno s tablicom u bazi podataka, a tome nam može pomoći naredba za generiranje modela „rails generate model“, gdje nakon naredbe dolaze argumenti poput imena modela, stupaca u tablici itd.

Kada stvorimo model, kod izgleda ovako:

```
class User < ApplicationRecord

end
```

Analogno, naredbom smo dobili i tablicu „users“ u bazi podataka. Stvorena klasa User namijenjena je definiranju veza s ostalim modelima te poslovne logike koja je potrebna.

4.1.1. CRUD naredbe

Rad s Active Recordom trebao bi biti što intuitivniji i jednostavniji, stoga su osnovne naredbe koje omogućuju rad s bazom podataka jednostavne.

Ako želimo stvoriti novi objekt, koristi se ključna riječ „new“, na primjer:

```
user = User.new(email: 'test@gmail.com', username: 'test')
```

Naravno, taj objekt nije spremljen u bazu, ali je to moguće učiniti naredbom „save“.

Ako zbog nekog razloga ne trebamo stvoriti objekt, pa ga kasnije spremiti, postoji naredba „create“ koja to radi automatski.

```
user = User.create(email: 'test@gmail.com', username: 'test')
```

Što se tiče dohvaćanja objekta iz baze, Active Record nudi pregršt metoda kako bi taj proces olakšao, od kojih većina ima i intuitivan naziv. Na primjer, ako želimo dohvatiti sve korisnike, postoji metoda „all“. Ako želimo dohvatiti korisnika po primarnom ključu, Active Record nudi metodu „find“ koja kao argument prihvaća broj.

Jedne od korisnijih Active Record metoda su metode „find_by“ i „where“. Ove metode rade sličnu stvar, odnosno dohvaćaju objekt po nekom kriteriju, ali je razlika što „find_by“ dohvaća samo jedan objekt iz baze, a „where“ skup objekata koji ispunjavaju kriterij. Ako na primjer želimo dohvatiti korisnika s određenim mailom, kod bi izgledao ovako:

```
user = User.find_by(email: 'test@gmail.com')
```

Ažuriranje je slično kao kreiranje, ali prvo moramo dohvatiti objekt kojeg želimo ažurirati.

```
user = User.find(1)

user.update(email: 'change@gmail.com')
```

U kodu iznad prvo smo dohvatili korisnika s primarnim ključem vrijednosti 1, te smo mu zatim promijenili email u navedenu vrijednost. Naredba „update“ odmah sprema vrijednosti u bazu. Ako želimo ažurirati neki stupac u svim objektima određene klase, odnosno neki atribut svih redova u tablici, koristimo naredbu „update_all“.

Brisanje je također sadržano u jednoj metodi, „destroy“. Nakon što pronađemo objekt i pozovemo tu metodu, redak je izbrisan iz baze.

4.1.2. Validacije

Validacije postoje kako bismo bili sigurni da se samo odgovarajući podaci spremaju u bazu. Na primjer, validacije se koriste kada svaki korisnik treba imati jedinstveni email ili korisničko ime. One su na razini modela, što znači da ih krajnji korisnik ne može zaobići.[23] Validacije se događaju pri spremanju u bazu, što znači da metode „save“, „create“ i „update“ pokreću validacije. Ukratko, one rade na način da odbijaju spremanje objekta u bazu ako neka validacija padne, odnosno baci grešku. Rails nudi razne pomoćnike (eng. helpers) kako bi programeri jednostavnije pisali validacije.

Pomoćnika za pisanje validacija ima puno, a neki od češće korištenih su:

- presence - ovaj pomoćnik zahtijeva da određeni atribut nije null ili prazan string
- length – ovaj pomoćnik zahtijeva određenu duljinu stringa određenog atributa (npr. ne želimo da korisnik može imati predugačak username)
- uniqueness – zahtjev za jedinstvenom vrijednosti atributa (npr. emaila ili korisničkog imena)

Svakoj od ovih validacija možemo dati određenu poruku koja se mora pojaviti ako validacija padne. To se radi s opcijom „message“.

```
class User < ApplicationRecord

  validates :username, :email, presence: true, presence: { message: "must be given please" }

  validates :username, :email, uniqueness: true

end
```

U ovom primjeru vidimo korištenje nekoliko jednostavnih validacija. Pri stvaranju ili ažuriranju korisnika atributi korisničkog imena moraju biti prisutni, odnosno ne smiju biti prazni te unikatni. Ako jedan od atributa nije prisutan, javlja se proizvoljna poruka koju možemo prikazati u sloju pogleda.

4.1.3. Asocijacije

Asocijacije modela u Railsu analogne su odnosima primarnih i vanjskih ključeva u bazi podataka. Rails, odnosno Active Record podržava šest vrsta asocijacija: [24]

- belongs_to - ova asocijacija koristi se u modelima koji pripadaju nekom drugom modelu, odnosno tablici u bazi koja posjeduje vanjski ključ na neku drugu tablicu. Na primjer, ako imamo modele objava i korisnika, svaka objava pripada jednom korisniku.
- has_one – ova asocijacija koristi se kada neki model može imati samo jednu instancu modela. Na primjer, auto može imati samo jedan motor.
- has_many – najčešća vrsta asocijacije. Ako koristimo primjer iz belongs_to, svaka objava pripada jednom korisniku, ali korisnik može imati više objava.
- has_many :through – ova se asocijacija koristi kada imamo vezu više-više. Postoje dva modela koji su vezani trećim. Na primjer, račun može imati više proizvoda, proizvod može biti na više računa. Zbog toga napravimo treći model kojeg možemo nazvati stavka_računa, gdje se nalaze primarni ključ proizvoda i računa, uz količinu ili neki drugi važan atribut
- has_one :through – slično kao prošla asocijacija, ali kada su modeli povezani vezom jedan-jedan
- has_and_belongs_to_many – slično kao „through“ asocijacija, ali bez trećeg modela. Ovaj se pristup preporuča kada nam ne trebaju neki dodatni atributi na trećem modelu.

Kada stvaramo asocijacije modela, važno je pripaziti na par stvari. Prvo, moramo biti sigurni da se vanjski ključevi nalaze u tablicama, zato što asocijacije ne mogu raditi bez tablica. Također je bitno provjeriti da su veze obostrane. Ako u jednom modelu imamo napisanu vezu „has_many“, u drugom modelu moramo napisati „belongs_to“, zato što inače jedan model neće znati za drugi**.**

4.2. Ruter

Kako bismo razumjeli koja je zadaća upravljača, potrebno je proučiti rute, odnosno kako Rails zna koji upravljač odgovara na koji zahtjev.

Railsov ruter prepoznaje URL-ove te ih povezuje s akcijom određenog upravljača. Također generira putanje i URL-ove koje možemo koristiti u pogledima, kako ne bismo morali hardkodirati putanje. [25]

Railsove konvencije kažu da postoji sedam osnovnih ruta koje upravljač mora pokrivati, a najčešće se vežu uz određeni model:

- index – ova akcija dohvaća sve objekte određenog modela. Koristi se GET metoda, zajedno s imenom modela (npr. /users)
- show – ova akcija dohvaća informacije o pojedinom objektu. Također se koristi GET metoda (npr. /users/:id, gdje je id primarni ključ nekog korisnika)
- new – ova akcija također koristi GET metodu, a najčešće vraća HTML obrazac koji može stvoriti novi objekt.
- create – akcija gdje se stvara novi objekt. Očekivano, koristi se POST metoda
- edit – akcija gdje se dohvaća HTML za ažuriranje podataka postojećeg modela.
- update – akcija gdje se ažurira pojedini model s podacima iz obrasca iz „edit“ akcije. Koristi se PATCH ili PUT metoda
- destroy – akcija koja briše određeni model te koristi DELETE metodu

Svaka se od ovih akcija veže na jednu metodu u upravljaču koji je zadužen za određen model. Naravno, Rails ima jednostavni način definiranja ovih sedam akcija, a to je korištenjem ključne riječi „resources“. Ako želimo napraviti obradu ovih sedam zahtjeva za korisnika, u kodu to izgleda ovako:

```
Rails.application.routes.draw do
  resources :users
end
```

4.3. Upravljači

Ako je Active Record implementacija „m“ dijela mvc arhitekture, Action Controller je implementacija „c“ dijela.

Nakon što ruter odredi koji se upravljač koristi za pojedini zahtjev, zadatak upravljača je vratiti odgovarajući odgovor. [26]

Kao i za sve ostalo, Rails posjeduje određene konvencije imenovanja upravljača. Pravilo glasi da su upravljači najčešće u množini, naprotiv modelima čije je ime u jednini. Na primjer, ako želimo napraviti upravljač za korisnika (eng. user), ime klase upravljača bit će „UsersController“.

Klasa upravljača sastoji se od metoda koje su nazvane isto kao akcije iz poglavlja o ruteru.

```
class UsersController < ApplicationController
  def index
    @users = User.all
  dnd
  def new
    @user = User.new
  end
end
```

U gornjem primjeru možemo vidjeti primjer upravljača zadužen za korisnike, zajedno s dvije metode odnosno akcije. U metodi „index“ dohvaćamo sve korisnike te ih spremamo u varijablu instance. Ta varijabla se zatim automatski šalje pogledu, gdje se podaci prikazuju u prikladnom obliku. Druga metoda je metoda „new“ gdje se stvara novi korisnik, ali bez ikakvih parametara. Parametara nema zato što je „new“ zaslužan za prikaz obrasca za stvaranje korisnika, a ne spremanje korisnika u bazu.

U upravljaču se također nalaze i dopušteni parametri za stvaranje ili ažuriranje nekog objekta. To je Railsov preduvjet, kako bi se smanjila vjerojatnost sigurnosnog propusta.

```
def user_params
    params.require(:user).permit(:username, :email)
end
```

Ovaj isječak koda osigurava da maliciozni korisnik ne može poslati neautorizirane podatke na poslužitelj.

4.3.1. Params

Params je naziv za rječnik u Railsu iz kojeg dohvaćamo podatke koji su nam potrebni, a ti podaci dolaze iz URL-a zahtjeva ili njegovog tijela. Na primjer, ako imamo sljedeći dio URL-a: users/2, vrijednosti primarnog ključa pojedinog korisnika možemo pristupiti na sljedeći način:

```
params[:id]
```

Rječnik „params“ koristi se i pri stvaranju i ažuriranju objekata, zato što se u tijelu zahtjeva nalaze atributi pojedinog objekta**.**

4.4. Pogledi

Sloj pogleda u Railsu je onaj koji se bazi prikazom podataka iz upravljača korisniku. Za to se koriste html.erb datoteke. ERB (punoga imena Embedded Ruby) je tehnologija koja omogućuje „ugradnju“ Ruby koda u tekstualne datoteke u svrhu prikaza podataka ili kontrole toka. [27]

Osnove ERB-a su oznake „<% %>“ te <%= %>. Prva oznaka se koristi kada želimo ugraditi kod, ali bez prikaza njegove povratne vrijednosti. To je korisno kod korištenja petlji ili uvjeta. Druga oznaka se koristi kada želimo prikazati povratnu vrijednost koda. U kontekstu Railsa to su najčešće informacije o varijabli instance koju stvorimo u upravljaču. Ako iskoristimo primjer korisnika koji posjeduje atribute korisničkog imena i mail adrese, prikaz tih podataka može izgledati ovako:

```
<h1> Informacije o korisniku </h1>

<% @users.each do |user| %>

  <p><%= user.username %></p>

  <p><%= user.email %></p>

<% end %>
```

U gornjem primjeru prvo vidimo „each“ petlju koja iterira po listi korisnika. Listu korisnika smo prethodno dohvatili u upravljaču uz pomoć „all“ metode nad klasom User. Klasu User stvorili smo kao model te analogno tome stvorili tablicu korisnika u bazi podataka. Nakon iteriranja po listi prikazujemo atribute svakog korisnika.

Jedna od korisnijih konvencija Railsa je i imenovanje html.erb datoteka. Metoda ili akcija upravljača koja je zadužena za prikaz svih korisnika zove se „index“. Pod uvjetom da Rails aplikacija nije API, ona pretpostavlja da na kraju akcije upravljača želimo prikazati datoteku istog imena. Zato se pogled zadužen za prikaz korisnika zove „index.html.erb“. Ista pravila vrijede za „edit“, „new“ i „show“ akcije.

4.4.1. Dijelovi (eng. partials)

„Partials“ je mogućnost prikaza jedne html.erb datoteke u drugoj html.erb datoteci. To pomaže u slučaju kada jedna datoteka postane glomazna, ili ako datoteku možemo podijeliti u jasne dijelove. U „partialsima“ se najčešće nalaze informacije o pojedinom objektu ili obrazac za stvaranje ili ažuriranje objekta. Imenovanje je uglavnom proizvoljno, ali početak imena mora početi s donjom crtom (npr. „_form.html.erb“).

Još jedna prednost korištenja dijelova kod obrazaca je primjena DRY (Do not repeat yourself) principa. Obrasci za stvaranje i ažuriranje su najčešće identični. U tom slučaju isti dio, odnosno datoteku, možemo prikazati i u pogledu za stvaranje i u pogledu za ažuriranje. Ako obrazac napravimo po konvencijama, Rails se brine o ispunjavanju podataka u obrascu. U slučaju da pošaljemo prazan objekt, podaci će biti prazni. U slučaju da pošaljemo već postojeći objekt iz baze, podaci u obrascu se automatski ispunjuju, bez potrebe pisanja uvjeta ili nečeg sličnog. U slučaju da smo napravili datoteku imena „_form.html.erb“, prikaz te datoteke je jednostavan, gdje u željeni pogled ubacimo ovu liniju koda:

```
<%= render "form" %>
```

4.4.2. Obrasci

Obrasci su osnova interakcije korisnika s web aplikacijom. Očekivano, Rails posjeduje pomoćnike (eng. helpers) koji programeru olakšavaju programiranje obrazaca.

Osnovni pomoćnik za stvaranje obrasca je „form_with“. Naravno, postoje razni argumenti za http metode, putanje i slično.

Najkorišteniji argument je argument „model“. On omogućuje stvaranje obrasca koji je vezan uz određen objekt. Time se postiže popunjavanje vrijednosti u slučaju ažuriranja. Ako iskoristimo primjer korisnika, jednostavna forma za stvaranje ili ažuriranje korisnika bila bi:

```
<%= form_with model: @user do |form| %>
  <%= form.label :username %>
  <%= form.text_field :username %>
  <%= form.label :email %>
  <%= form.text_field :email %>
  <%= form.submit %>
<% end %>
```

„label“ i „text_field“ jedni su od pomoćnika koji tvore obrazac te su analogni vrsti podataka koju očekujemo od korisnika. Naravno, postoje pomoćnici za praktički svaku vrstu podataka koju korisnik može upisati, od kojih su neki pomoćnici za lozinke, brojeve, datume, popise elemenata, kućice koje korisnik označuje i drugo.

5. Praktični dio

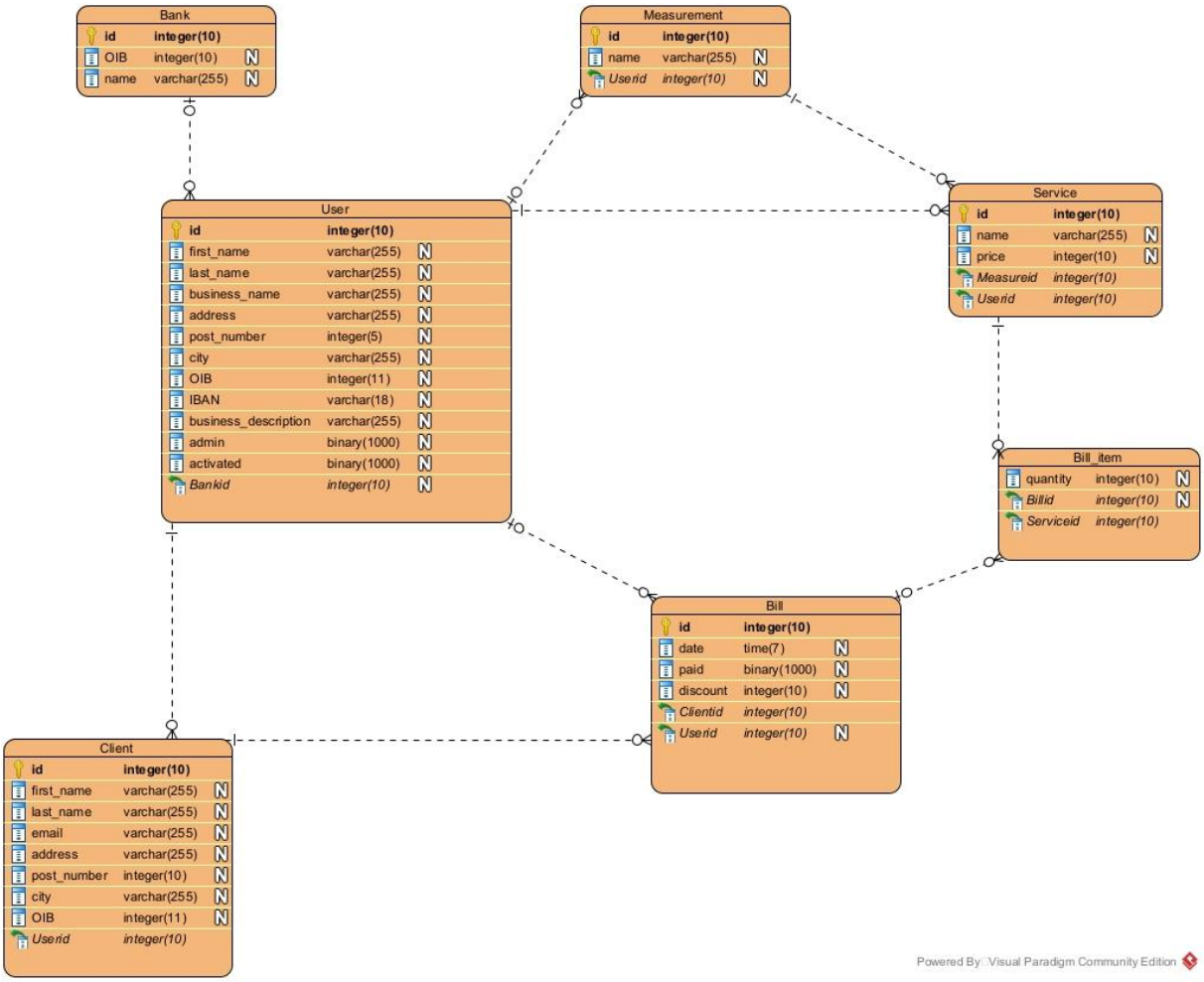
Praktični dio rada sastojao se od primjene programskog jezika Ruby te okvira Rails u izradi konkretne web aplikacije.

Aplikacija koju sam izradio zove se Paušalni Asistent (eng. Tradesman Assistant). Aplikacija bi služila obrtnicima kao centralni dio poslovanja. Odnosno, upravljanje klijentima, računima, uslugama i sličnome bi premjestili iz npr. Excel tablica u ovu aplikaciju. Također, neprijavljeni korisnici bi imali mogućnost pregledavanja popisa obrtnika i njihovih usluga.

Osnovne funkcionalnosti aplikacije uključuju:

- Upravljanje klijentima. Obrtnik ima mogućnost upravljanja klijentima (čitanje, stvaranje, ažuriranje i brisanje).
- Upravljanje uslugama. Slično kao i klijentima, obrtnik ima mogućnost upravljanja uslugama. Svaka usluga također može imati zasebnu mjernu jedinicu (1 sat ili komad). Obrtnici također mogu popis usluga stvoriti u PDF formatu kao cjenik.
- Upravljanje računima. Obrtnik može stvarati račune koji su vezani za jednog klijenta te proizvoljan broj usluga. Taj račun također može stvoriti u PDF formatu kako bi ga poslao klijentu. Svaki račun ima svoj status, odnosno obrtnik može označiti koji račun je plaćen, a koji nije.
- Statistika. Obrtnik može vidjeti svoju statistiku koja uključuje broj plaćenih i neplaćenih računa te njihov odnos, broj računa po svakom klijentu te ukupan profit u godini.
- Prihvaćanje korisnika. Administrator sustava (ne obrtnik) mora prihvatiti obrtnika kako bi se on mogao ulogirati u sustav.
- Upravljanje bankama. Budući da HNB nema API za automatizaciju tog procesa, administrator brine o popisu banaka koji obrtnik odabire pri registraciji.

5.1. ERA dijagram



Slika 1 ERA dijagram

ERA dijagram služi kao prikaz strukture baze podataka aplikacije.

Najvažnija tablica u aplikaciji je tablica korisnika (eng. User). U nju se spremaju obrtnici pri registraciji te administratori. Treba obratiti pažnju na stupce za aktivaciju te administratora. Budući da su u aplikaciji samo dvije uloge, nije bila potrebna tablica s popisom uloga, već je bilo dovoljno napraviti stupac „admin“ koji može bit lažan ili istinit. Stupac se pri stvaranju korisnika automatski postavlja na lažno. Stupac aktivacije označava je li administrator odobrio prijavu korisnika, te se taj stupac također pri stvaranju postavlja na lažno.

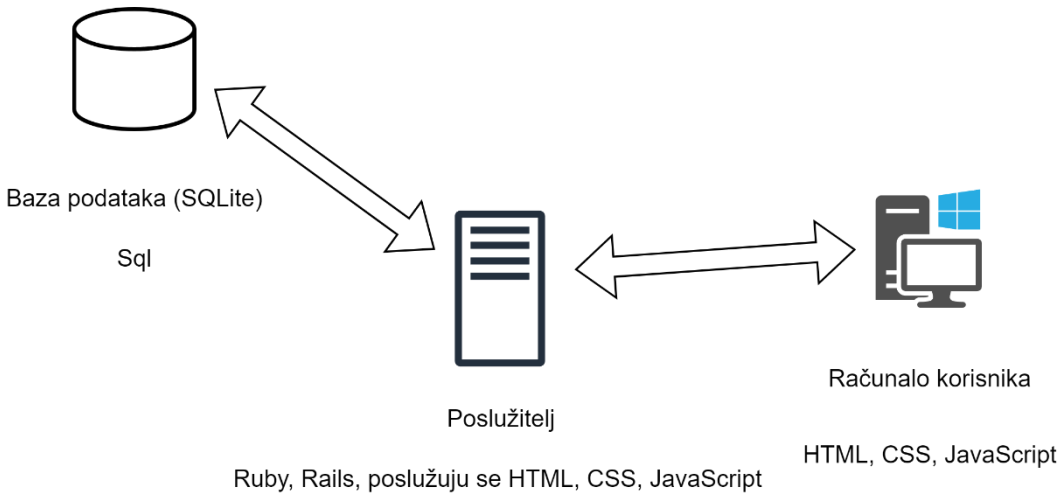
S obzirom da obrtnik pri registraciji odabire banku, svaki obrtnik pripada određenoj banci. Bankama upravlja administrator te ta tablica nema vanjski ključ od neke druge.

Zatim imamo klijente, usluge te mjerne jedinice koje posjeduju potrebne atribute te su vezane za pojedinog korisnika. Svaka usluga posjeduje jednu mjernu jedinicu.

Tablica za račune sastoji se od datuma stvaranja koji se automatski postavlja na današnji datum, stupca za popust koji je u obliku cjelobrojne vrijednosti zato što je intuitivnije napisati popust od npr. 20% nego 0.2, te stupca za status plaćanja. Svaki račun se veže uz jednog obrtnika te jednog klijenta.

Tablica za stavke računa označavaju pojedine stavke na računu te se sastoji od količine te jedne usluge. Račun može imati proizvoljan broj stavki, ali svaka stavka pripada jednom računu.

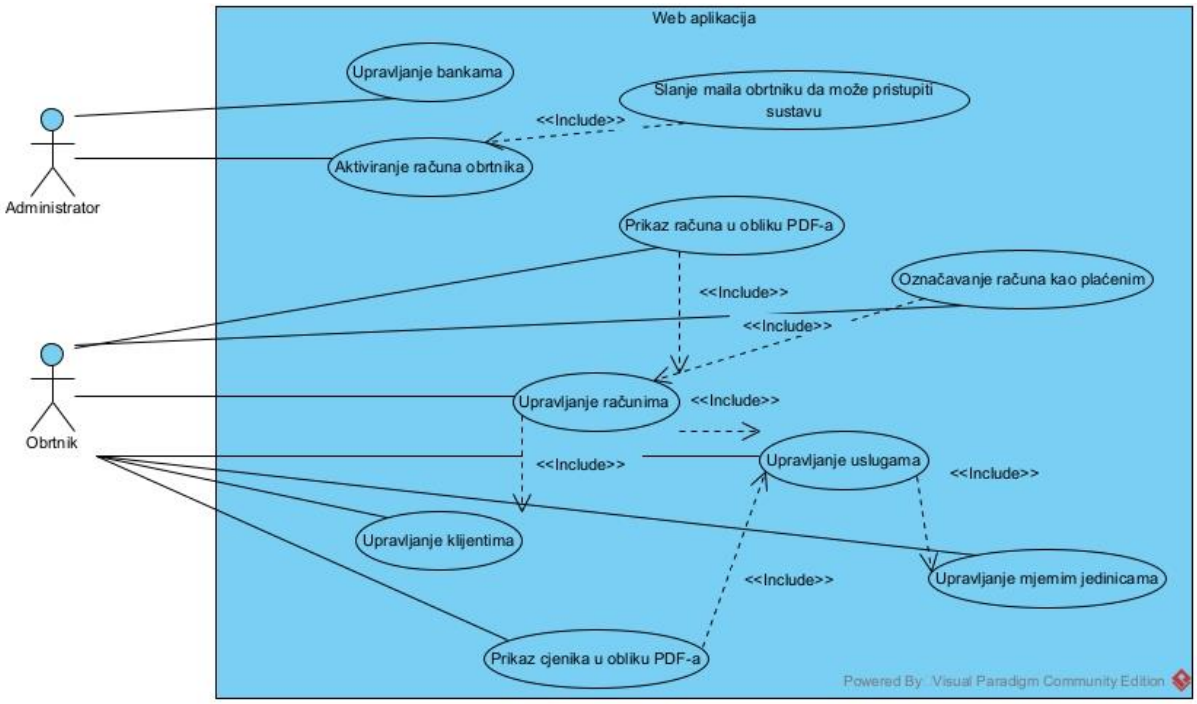
5.2. Arhitektura



Slika 2 Arhitektura aplikacije

Iako se radi o web aplikaciji, zahvaljujući okviru Rails arhitektura je dosta jednostavna. Budući da Rails može obuhvaćati i klijentsku i pozadinsku stranu aplikacije, dovoljan je jedan poslužitelj (ili jedan port) za cijelu web aplikaciju. Naravno, poslužitelj komunicira s bazom podataka koja je u ovom slučaju SQLite, ali se jednostavno promijeni u Postgres ili neku drugu bazu. S druge strane, klijent šalje zahtjeve te dobiva odgovore od poslužitelja. U bazi podataka koristi se jezik SQL, dok se na poslužitelju poslužuju HTML, CSS, JavaScript u obliku vanjskog paketa te Ruby s okvirom Rails.

5.3. Dijagram slučajeva korištenja



Slika 3 Dijagram slučajeva korištenja

Dijagram slučajeva korištenja služi za jasniji prikaz funkcionalnosti aplikacije. On se ne zamara sa samom implementacijom. Funkcionalnosti su objašnjene u prijašnjem poglavlju, ali valja objasniti veze između funkcionalnosti. Najveći broj funkcionalnosti je povezan „include“ vezama, koje označavaju da je neka funkcionalnost preduvjet za ostvarenje neke druge. Na primjer, kako bi obrtnik napravio račun, on prvo mora dodati klijenta, barem jednu uslugu, koja pak zahtijeva dodavanje mjerne jedinice. Slično vrijedi i za prikaz usluga i računa u obliku PDF-a. Prikaz nije moguć ako nema usluga ili računa koji bi se mogli prikazivati.

5.4. Implementacija

U ovome poglavlju bit će opisani načini implementacije konkretnih funkcionalnosti u aplikaciji. Detaljnije objašnjeni bit će zahtjevniji dijelovi implementacije, dok stvari koje su u svakoj web aplikaciji poput registracije, logina i sličnog neće biti objašnjene.

5.4.1. Paketi i okviri

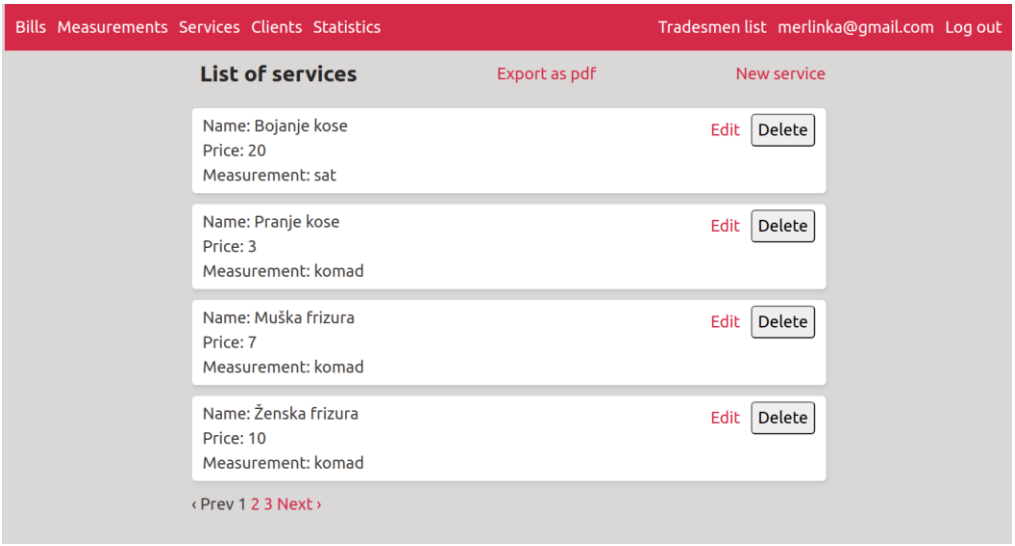
Jedan od razloga zašto se za Rails smatra kako je dobar za razvoj minimalno vitalnih proizvoda (eng. minimum viable product ili kraće MVP) je velika količina paketa i okvira koji su

napravljeni kao proširenje Rails-u. Kako bih ubrzao i pojednostavio razvoj svoje web aplikacije, odlučio sam dodati određene pakete:

- Devise – jedan od najpopularnijih Ruby paketa uopće s preko 168 milijuna preuzimanja. Namjena paketa je pojednostaviti autentikaciju korisnika. Dolazi s predodređenim upravljačima i pogledima. Ipak, neke upravljače i poglede sam morao izmijeniti kako bih ostvario funkcionalnosti aktiviranja korisnika te registracije s više polja od mail adrese i lozinke.
- Letter opener – paket koji služi za jednostavnije testiranje slanja mailova. Nakon određenih postavki, umjesto da se mail šalje na adresu, on se prikazuje u novoj kartici preglednika.
- Wicked PDF – paket koji omogućuje pretvaranje HTML pogleda u PDF format. Nažalost, paket je dosta zastario sa svojim funkcionalnostima (npr. ne podržava sve znakove, CSS flexbox i dr), ali noviji paket s istom namjenom trenutno ne postoji.
- Pagy – paket koji pojednostavljuje i automatizira straničenje u aplikaciji.

Rails od verzije 7 dolazi s okvirom Hotwire koji je oglašavan kao alternativa JavaScript okvirima za klijentsku stranu poput Angulara, Reacta i drugih. S obzirom da je tema rada okvir Rails, koristio sam Hotwire za određene dijelove aplikacije. Rails se tradicionalno koristio za stvaranje višestraničnih aplikacija (eng. multiple-page application) ili kao JSON API, ali s okvirom Hotwire moguće je stvoriti dojam jednostranične aplikacije (eng. single-page application).

5.4.2. Upravljanje uslugama



Slika 4 Upravljanje uslugama

Upravljanje uslugama jedna je od funkcionalnosti gdje sam odlučio upotrijebiti mogućnosti Hotwire okvira. Specifičnost ovog ekrana je što se svi dijelovi funkcionalnosti (dodavanje, ažuriranje, brisanje) odvijaju na istom ekranu.

Kao većina funkcionalnosti u Rails aplikaciji, implementacija se sastoji od modela koji je analogan bazi podataka, upravljaču te pogledima. Model usluge u Rails kodu izgleda ovako:

```
class Service < ApplicationRecord
  belongs_to :measurement
  belongs_to :user
  validates :name, :price, presence: true
  validates :price, numericality: true
  has_many :bill_items, dependent: :restrict_with_error
end
```

S obzirom da je cilj Rubyja izgledati što jednostavnije, kod modela je analogan engleskom jeziku. Iz koda se može iščitati kako klasa ili tablica usluge pripada klasama mjerne jedinice te korisnika. Tu asocijaciju koristimo kada u određenoj klasi imamo vanjske ključeve neke druge klase. Također, vidimo jednostavne validacije koje zahtijevaju da su svi atributi postojeći te da je cijena brojčana vrijednost. Još je navedeno kako određena usluga ima mnogo stavki računa, te je ograničeno brisanje usluge ako postoji stavka s tom uslugom.

Upravljač ove funkcionalnosti izgleda kao prosječni Rails upravljač, odnosno sadržava sve upravljačke akcije navedene u poglavlju vezanom uz upravljače.

Pogled zaslužan za prikaz svih usluga je najvažniji, stoga njegov kod slijedi:

```
<main class="container">

  <% flash.each do |type, msg| %>
    <div>
      <%= msg %>
    </div>
  <% end %>

  <div class="header">
    <h1>List of services</h1>
    <%= link_to "Export as pdf", services_path(format: :pdf), target:
:blank %>
    <%= link_to "New service", new_service_path, data: { turbo_frame:
dom_id(Service.new) } %>
  </div>
```

```
  <%= turbo_frame_tag Service.new %>

  <%= turbo_frame_tag "services" do %>
    <%= render @services %>
  <% end %>

  <%== pagy_nav(@pagy) if @pagy.pages > 1 %>
</main>
```

Najvažnija oznaka za rad Hotwire okvira je oznaka „turbo_frame_tag“. Hotwire radi na principu zamjene određenih dijelova stranica. Nakon deklariranja „turbo“ oznake, moramo dodati ime oznake. Nakon što korisnik pritisne gumb ili link na stranici, vrši se zamjena okvira na trenutnoj stranici, s okvirom istog imena na nekoj drugoj stranici. Stoga, ako želimo zamijeniti jednu uslugu s obrascem za ažuriranje, samu uslugu i obrazac moramo „zaogrnuti“ „turbo“ oznakom istoga imena. Tako se dobiva učinak SPA aplikacije.

Malo kompleksnija situacija je pri stvaranju nove usluge. Prvi dio Hotwire okvira, Turbo Okviri (eng. Turbo Frames) omogućuje samo zamjenu dva okvira, odnosno ne radi ako želimo zamijeniti više dijelova dokumenta istovremeno. Rješenje tome dolazi u obliku drugog dijela okvira, Turbo Streams. On omogućuje dodavanje, ažuriranje ili micanje više dijelova dokumenta. U našem slučaju, kada stvorimo novu uslugu, istovremeno moramo maknuti obrazac za stvaranje usluge te dodati novu uslugu na stranicu. To se radi s jednostavnom datotekom.

```
<%= turbo_stream.append "services", @service %>

<%= turbo_stream.update Service.new, "" %>
```

Kao što se može pretpostaviti iz glagola, nakon stvaranja nove usluge, okviru s imenom „services“ dodaje se nova usluga te se istovremeno okvir zaslužan za obrazac zamjenjuje s praznim stringom.

5.4.3. Upravljanje računima

Upravljanje računima bila je najkompleksnija funkcionalnost za implementirati. Model računa je najopširniji zato što se veže na gotovo sve ostale modele u aplikaciji.

```
class Bill < ApplicationRecord
  after_initialize :set_defaults, if: :new_record?
  after_find :calculate_price
  before_save :check_items

  belongs_to :client
```

```
  belongs_to :user
  has_many :bill_items, dependent: :delete_all
  accepts_nested_attributes_for :bill_items, reject_if: :all_blank,
allow_destroy: true
  validates :discount, presence: true, numericality: { in: 0..100 }

  attr_accessor :price

  private

  def set_defaults
    self.paid = false
    self.date = Date.current
  end

  def check_items
    if bill_items.empty?
      errors.add(:base, "You cannot create an empty bill")
      throw :abort
    end
  end

  def calculate_price
    price = (bill_items.sum { |bill_item| bill_item.service.price *
bill_item.quantity }) * (1 - (discount / 100.0))
    self.price = price.round(2)
  end
end
```

Za početak, na vrhu su navedene funkcije koje se moraju odvijati u nekom dijelu životnog ciklusa računa. Nakon stvaranja novog računa, poziva se funkcija „set_defaults“. Ona na razini modela postavlja stupce statusa računa na lažni, te datum stvaranja računa na današnji datum. To nije moglo biti napravljeno na razini baze, zato što je datum dinamička vrijednost. Nakon pronalaska računa poziva se funkcija „calculate_price“. Ta funkcija u objekt računa dodaje novi stupac koji se ne sprema u bazu. Prednost toga je to što ako bismo promijenili cijenu neke usluge, cijena računa se automatski ažurira. Još se prije spremanja provjerava je li račun prazan, odnosno ima li stavke. Ako nema, brani se spremanje u bazu.

Nakon funkcija nalaze se asocijacije računa s drugim modelima. Najbitnija stavka ovdje je „accepts_nested_attributes_for“. Ta metoda omogućava istovremeno stvaranje

„roditeljskog“ objekta i objekta „djeteta“. U našem slučaju, pri stvaranju računa istovremeno u bazu spremamo i stavke.

Za ostvarenje ove funkcionalnosti bio je potreban i odgovarajući upravljač „BillsController“, čije isječke možemo vidjeti ovdje:

```
def show
  respond_to do |format|
    format.html
    format.pdf do
      render pdf: "Bill", template: "bills/pdf", formats: [:html]
    end
  end
end

def create
  @bill = current_user.bills.build(bill_params)
  if @bill.save
    redirect_to @bill
  else
    render :new, status: :unprocessable_entity
  end
end

def set_as_paid
  if @bill.update(paid: true)
    respond_to do |format|
      format.html { render @bill, notice: "Bill set as paid" }
      format.turbo_stream {render turbo_stream: turbo_stream.update("paid",
                           "<p>Payment status: paid </p>") }
    end
  end
end
```

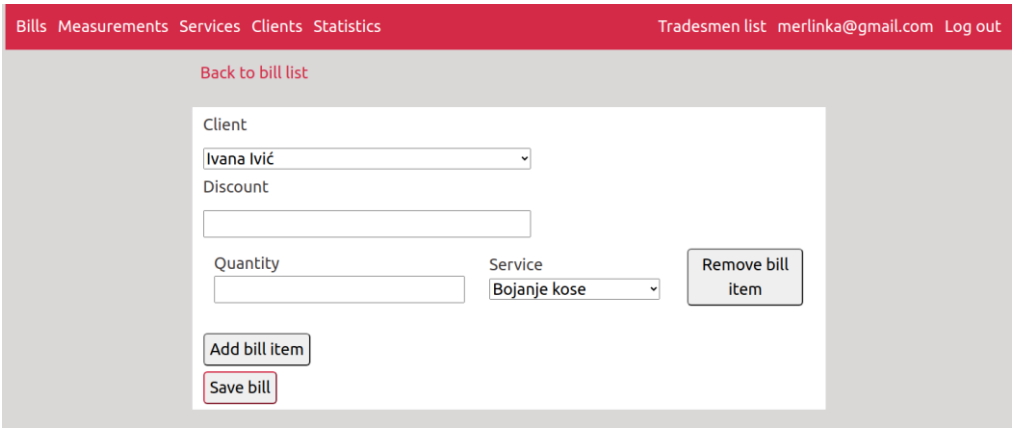
Prva važna metoda u upravljaču je metoda „show“. Ona služi za prikaz pojedinog računa, a zanimljivi dio je blok koda „respond_to do“. Taj blok zapravo govori da ako korisničko računalo zahtijeva HTML, odgovor će biti HTML stranica računa. Ako s druge strane računalo zahtijeva PDF, odgovor će biti u obliku PDF datoteke.

Metoda „create“ zaslužna je za odgovaranje na POST zahtjev gdje se stvaraju račun te stavke računa. U „bill_params“ sadržani su i atributi stavki računa.

Zadnja važna metoda, te ona koja je specifična za ovaj slučaj korištenja, je metoda „set_as_paid“. Ta metoda služi za označavanje računa kao plaćenog. Tu smo metodu morali

posebno dodati u ruteru, a njezin odgovor je zapravo primjena Hotwirea gdje se gumb i status računa mijenjaju s novim statusom.

Najkompleksniji dio funkcionalnosti bio je napraviti obrazac gdje možemo micati i dodavati proizvoljan broj stavki. Za to sam se poslužio vanjskim JavaScript paketom čija je namjena upravo to. Cijeli se paket zove Stimulus Components, a jedan dio je „nested forms“ koji radi upravo ovo što treba.



Slika 5 Obrazac za račun

Obrazac radi na način ručnog dodavanja ili micanja pojedinih stavaka računa. Ako se pokuša spremiti neodgovarajući račun, javlja se greška.

5.5. Kritički osvrt

Do sada sam za izradu web aplikacija u sklopu gradiva fakulteta koristio okvir za pozadinsku stranu Express te okvir Angular za klijentsku stranu. Express i Rails predstavljaju suprotne filozofije. Rails je takozvani „batteries included“ okvir, gdje je predodređena struktura datoteka, koji ORM se koristi te kako se rješavaju najčešći problemi pri izradi web aplikacija. Cilj Railsa je odluke programera ograničiti na rješavanje poslovne logike, odnosno pomoći mu da se ne zamara sa strukturom. Ss druge strane, Express pokušava biti što manji i jednostavniji. Sam po sebi ne nudi puno, ali zato ima mogućnost potpune prilagodbe programeru ili timu programera. Struktura datoteka, ORM i sve ostale stvari su pod odlukom programera.

Teško je izdvojiti najteži dio realiziranja aplikacije. Svaki je dio nosio različite probleme koje je trebalo riješiti. Na nekim sam obrascima odlučio primijeniti Hotwire okvir. Taj okvir zahtijeva drukčiji način razmišljanja kojeg je trebalo shvatiti.

Trebalo je promijeniti i upravljače i poglede paketa za autentikaciju Devise. Običan Devise nudi stupce maila i lozinke, stoga je trebalo dodati stupce za potrebe aplikacije,

promijeniti poglede kako bi se u obrascu zahtijevaju svi stupci, promijeniti upravljače kako bi se mogao vršiti odabir banaka iz predodređenog izbora. Na kraju, trebalo je promijeniti samu logiku autentikacije na način da se ona odbija ako korisnik nije aktiviran.

Još jedan kompleksan dio bio je snalaženje sa CSS-om. Jedan od nedostataka Railsa je globalni CSS, odnosno Rails radi na način da sve CSS datoteke kompilira u jednu te ih tako šalje pregledniku. Nedostatak toga je to što ako bismo neki izgled primijenili na opću oznaku (npr. h1), taj izgled će se primijeniti na apsolutno svaki naslov na svakoj stranici aplikacije. Taj sam problem riješio korištenjem klasa. Ovaj rad bila je i odlična prilika za iskoristiti ugnježđivanje. Ove je godine u „vanilla“ CSS dodana mogućnost ugnježđivanja, slično kao i u predprocesorima za CSS.

Najveći dio rješavanja problema svodio se na rješavanje problema na način koji Rails zahtijeva, što je na prvu zamorno, ali kasnije omogućuje veću produktivnost. Smatram da je Rails dobar izbor za razvoj web aplikacija, ali bi „front-end“ rješenje moglo biti bolje. Danas se radi na rješenjima poput InertiaJs koji nude najbolje od oba svijeta: JavaScript okvir za klijentsku stranu te okvir za pozadinsku stranu po želji, ali se komponente klijentske strane stvaraju kao što npr. Rails stvara obične poglede. Prednost ovog pristupa je to što nema potrebe za dva repozitorija te se smanjuje potreba za duplom logikom na oba repozitorija.

6. Zaključak

Programiranje web aplikacija je danas najraširenији oblik programiranja. Cijelo čovječanstvo je na internetu, a osobno ne vidim način na koji bi se to promijenilo. Naprotiv, smatram da će s nekim konceptima poput progresivnih web aplikacija web uživati još veću popularnost.

Ovaj se rad bavio programskim jezikom Ruby te okvirom za izradu web aplikacija Rails. Prvo se govorilo o povijesti weba te nekim obilježjima web aplikacija. Nakon toga se ušlo u svijet Rubyja, programskog jezika kojemu je sreća programera najviši prioritet, što se vidi iz sintakse. Prvo su opisane osnove mogućnosti jezika, poput varijabli, petlji, grananja i slično, tj. stvarima koje ima svaki programski jezik. Nakon toga govorilo se o naprednijim mogućnostima.

Glavni dio rada bio je posvećen okviru Rails. Rails je okvir za izradu web aplikacija kojemu je cilj pojednostaviti proces koliko može. On je temeljen na MVC arhitekturi koju smo razložili u naknadnim poglavljima. Dio modela služi za obuhvaćanje tablica u bazi podataka te za poslovnu logiku koja je potrebna za određeni model. Dio pogleda služi kao klijentska strana web aplikacije, odnosno u izravnom je doticaju s korisnikom. Upravljači su na neki način mozak aplikacije. Oni dohvaćaju potrebne podatke te prikazuju odgovarajuće poglede.

Na kraju su ovaj jezik i okvir primijenjeni za izradu web aplikacije.

Korištenje jezika Ruby te okvira Rails bilo je ugodno iskustvo. Najviše mi se svidjela sintaksa koja je vrlo slična engleskom jeziku. Također je moguće biti vrlo produktivan kada se upoznamo s načinima rješavanja najčešćih problema u okviru. Danas je Rails manje popularan nego što je bio prije npr. 10 godina, ali s pojavom okvira Hotwire to bi se moglo promijeniti.

7. Popis literature

[1] „Topic: Internet usage worldwide“, *Statista*. https://www.statista.com/topics/1145/internet-usage-worldwide/ (pristupljeno 29. lipanj 2023.).

[2] „World Wide Web | History, Uses & Benefits | Britannica“, 23. svibanj 2023. https://www.britannica.com/topic/World-Wide-Web (pristupljeno 30. lipanj 2023.).

[3] „NCSA Mosaic™“, *NCSA*. https://www.ncsa.illinois.edu/research/project-highlights/ncsa-mosaic/ (pristupljeno 30. lipanj 2023.).

[4] K. J. Shakir M. Abass, „Development History Of The World Wide Web“, sv. 8, izd. 09, ruj. 2019.

[5] „Social network | Definition, Examples, & Facts | Britannica“, 27. lipanj 2023. https://www.britannica.com/technology/social-network (pristupljeno 30. lipanj 2023.).

[6] A. Mak, „What Is Web3 and Why Are All the Crypto People Suddenly Talking About It?“, *Slate Magazine*, 09. studeni 2021. https://slate.com/technology/2021/11/web3-explained-crypto-nfts-bored-apes.html (pristupljeno 30. lipanj 2023.).

[7] D. Roberts, „What is a Web Application?“, *StackPath*, 24. svibanj 2021. https://www.stackpath.com/edge-academy/what-is-a-web-application (pristupljeno 30. lipanj 2023.).

[8] „PHP: Preface - Manual“. https://www.php.net/manual/en/preface.php (pristupljeno 06. rujan 2023.).

[9] „What Is a Content Management System (CMS)?“, *Kinsta*®, 20. ožujak 2023. https://kinsta.com/knowledgebase/content-management-system/ (pristupljeno 06. rujan 2023.).

[10] „What is Java? | IBM“. https://www.ibm.com/topics/java (pristupljeno 06. rujan 2023.).

[11] „What is .Net? - Dotnet Explained - AWS“, *Amazon Web Services, Inc.* https://aws.amazon.com/what-is/net/ (pristupljeno 06. rujan 2023.).

[12] „What is JavaScript? - Learn web development | MDN“, 15. kolovoz 2023. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript (pristupljeno 07. rujan 2023.).

[13] „History of JavaScript“, *GeeksforGeeks*, 15. svibanj 2023. https://www.geeksforgeeks.org/history-of-javascript/ (pristupljeno 07. rujan 2023.).

[14] N. Heer, „Speed comparison of programming languages“. 10. rujan 2023. Pristupljeno: 10. rujan 2023. [Na internetu]. Dostupno na: https://github.com/niklas-heer/speed-comparison

[15] D. A. Black, *The well-grounded Rubyist*, Third edition. Shelter Island, NY: Manning, 2019.

[16] „module Enumerable - Documentation for Ruby 3.2“. https://docs.ruby-lang.org/en/3.2/Enumerable.html (pristupljeno 03. kolovoz 2023.).

[17] „What Are Accessor Methods in Ruby?“, *Designcise*, 22. listopad 2022. https://www.designcise.com/web/tutorial/what-are-accessor-methods-in-ruby (pristupljeno 04. kolovoz 2023.).

[18] „Getting Started with Rails“, *Ruby on Rails Guides*. https://guides.rubyonrails.org/getting_started.html (pristupljeno 05. kolovoz 2023.).

[19] „Welcome to Rails“. Ruby on Rails, 05. kolovoz 2023. Pristupljeno: 05. kolovoz 2023. [Na internetu]. Dostupno na: https://github.com/rails/rails

[20] „Everything You Need To Know About Ruby on Rails Web Application Framework“, *Flexiple*. https://flexiple.com/ruby-on-rails/deep-dive/ (pristupljeno 05. kolovoz 2023.).

[21] „Ruby on Rails“, *Ruby on Rails*. https://rubyonrails.org/ (pristupljeno 05. kolovoz 2023.).

[22] „Active Record Basics“, *Ruby on Rails Guides*. https://guides.rubyonrails.org/active_record_basics.html (pristupljeno 06. kolovoz 2023.).

[23] „Active Record Validations“, *Ruby on Rails Guides*. https://guides.rubyonrails.org/active_record_validations.html (pristupljeno 06. kolovoz 2023.).

[24] „Active Record Associations“, *Ruby on Rails Guides*. https://guides.rubyonrails.org/association_basics.html (pristupljeno 07. kolovoz 2023.).

[25] „Rails Routing from the Outside In“, *Ruby on Rails Guides*. https://guides.rubyonrails.org/routing.html (pristupljeno 07. kolovoz 2023.).

[26] „Action Controller Overview“, *Ruby on Rails Guides*. https://guides.rubyonrails.org/action_controller_overview.html (pristupljeno 07. kolovoz 2023.).

[27] „class ERB - Documentation for Ruby 3.2“. https://docs.ruby-lang.org/en/3.2/ERB.html (pristupljeno 08. kolovoz 2023.).

8. Popis slika

9. Popis tablica

10. Prilozi

Ovom radu priložena je zip datoteka izvornog koda praktičnog dijela rada, odnosno zip datoteka izvornog koda aplikacije Paušalni Asistent.