



Završni rad prvog ciklusa studija na Elektrotehničkom fakultetu u
Sarajevu

**Razvoj aplikacije za prikupljanje i vizualizaciju podataka sa
bankomata koristeći Angular JS i Google Mape**

Kandidat: Benjamin Ramić
Broj indexa: 17119

Mentor: Dr. Emir Buza, dipl.ing.el.

Sarajevo, 2017. godina

Univerzitet u Sarajevu
Naziv fakulteta/akademije: **Elektrotehnički fakultet**
Naziv odsjeka i/ili katedre: **Računarstvo i informatika**
Predmet: **Završni rad prvog ciklusa studija**

Izjava o autentičnosti radova

Seminarski rad, završni (diplomski odnosno magistarski) rad za I i II ciklus studija i integrirani studijski program I i II ciklusa studija, magistarski znanstveni rad i doktorska disertacija¹

Ime i prezime: **Benjamin Ramić**
Naslov rada: **Razvoj aplikacije za prikupljanje i vizualizaciju podataka sa bankomata koristeći Angular JS i Google Mape**
Vrsta rada: **Završni rad prvog ciklusa studija**
Broj stranica: **63**

Potvrđujem:

- da sam pročitao/la dokumente koji se odnose na plagijarizam, kako je to definirano Statutom Univerziteta u Sarajevu, Etičkim kodeksom Univerziteta u Sarajevu i pravilima studiranja koja se odnose na I i II ciklus studija, integrirani studijski program I i II ciklusa i III ciklus studija na Univerzitetu u Sarajevu, kao i uputama o plagijarizmu navedenim na web stranici Univerziteta u Sarajevu;
- da sam svjestan/na univerzitetskih disciplinskih pravila koja se tiču plagijarizma;
- da je rad koji predajem potpuno moj, samostalni rad, osim u dijelovima gdje je to naznačeno;
- da rad nije predat, u cjelini ili djelimično, za stjecanje zvanja na Univerzitetu u Sarajevu ili nekoj drugoj visokoškolskoj ustanovi;
- da sam jasno naznačio/la prisustvo citiranog ili parafraziranog materijala i da sam se referirao/la na sve izvore;
- da sam dosljedno naveo/la korištene i citirane izvore ili bibliografiju po nekom od preporučenih stilova citiranja, sa navođenjem potpune reference koja obuhvata potpuni bibliografski opis korištenog i citiranog izvora;
- da sam odgovarajuće naznačio/la svaku pomoć koju sam dobio/la pored pomoći mentora/ice i akademskih tutora/ica.

Mjesto, datum _____

Potpis _____

¹ U radu su korišteni slijedeći dokumenti: Izjava autora koju koristi Elektrotehnički fakultet u Sarajevu; Izjava o autentičnosti završnog rada Centra za interdisciplinarne studije – master studij „Evropske studije“, Izjava o plagijarizmu koju koristi Fakultet političkih nauka u Sarajevu.

Postavka zadatka

Teme za završne radove 1. ciklusa za 2016/2017 studijsku godinu

Nastavnik: doc. dr. Emir Buza

Student: Benjamin Ramić

Tema: Razvoj aplikacije za prikupljanje i vizualizaciju podataka sa bankomata koristeći Angular JS i Google Mape

Cilj:

- Upoznavanje sa osnovnim konceptima korištenja udaljene baze podataka za mobilne aplikacije
- Upoznavanje sa osnovnim konceptima vizualizacije podataka
- Razvoj konkretne aplikacije

Opis:

Povećanjem broja podataka sa kojima moderne kompanije posluju, javlja se potreba za kvalitetnom klasifikacijom i prikazom istih. Sve više kompanija se odlučuje da unutar svojih poslovnih informacionih sistema implementira i module za vizualizaciju podataka, kako bi sa istim lakše mogli operirati. Prednosti koje nudi kvalitetna vizualizacija podataka se prvenstveno ogledaju u njihovom lakšem iščitavanju, prezentaciji, kao i njihovoj koncentraciji na jednom mjestu. Pored vizualizacije podataka, velika većina modernih informacionih sistema unutar sebe integriše više različitih platformi koje koriste jedinstvenu bazu podataka. Te platforme su obično web aplikacije, mobilne aplikacije, desktop aplikacije i sl. Ovaj rad treba da pokaže sintezu svih gore navedenih koncepata kroz implementaciju jednog konkretnog rješenja, koji će koristiti opisane modele.

Okvirni sadržaj rada:

1. Uvod - U okviru uvodnog poglavlja prikazati će se uvod u temu, ciljevi rada, metodologija korištena za izradu rada kao i struktura rada.
2. Koncepti udaljenih baza podataka – U ovom poglavlju će se objasniti osnovni koncepti korištenja udaljenih baza podataka, načini upisivanja i čitanja podataka sa istih (POST, GET, WEB-API)
3. Koncepti vizualizacija podataka – U ovom poglavlju će se objasniti princip vizualizacija podataka, alati koji se koriste za vizualizaciju podataka i opravdanost korištenja ovakvog pristupa prezentacije podataka
4. Razvoj aplikacija za prikupljanje i vizualizaciju podataka sa bankomata - Opis funkcionalnosti aplikacije putem odgovarajućih UML dijagrama, kao i upotrijebljenih tehnologija prilikom implementacije aplikacije.
5. Zaključak - U okviru ovog poglavlja rezimirat će se urađeno, dati osvrt na rad i smjernice za buduće istraživanje vezano uz ovu temu, kao i prijedlozi za unaprjeđivanje kreirane aplikacije.

Očekivani rezultati: Implementirane aplikacije za prikupljanje i vizualizaciju podataka sa bankomata.

Polazna literatura:

1. Ben Fry: Visualizing Data, 2008 (O'Reilly)

2. Chun-houh Chen, Wolfgang Härdle, Antony Unwin: Handbook of Data Visualization, 2008 (Springer)
3. Dan Wahlin: AngularJS in 60 Minutes, 2014
4. Frederik Diet: Recipes with Angular.js, 2003
5. J.F. DiMarzio: Android™ A programmer's Guide, 2008
6. Leonard Richardson, Sam Ruby: RESTful Web Services, 2007 (O'Reilly)

Mentor: Dr. Emir Buza, dipl.ing.el.

Abstract

Da bi se najbolje objasnio sažetak rada, bitno je istaknuti da ovaj rad ne spada u domen naučnih radova, nego predstavlja praktični rad koji je vođen „proof of concept“ logikom, kojoj je za cilj dokazati da je postavljeni problem moguće realizirati korištenjem određenih alata i korištenjem već do sada kreiranih logika i mogućnosti.

Rad je realiziran kao završni rad prvog ciklusa studija na Elektrotehničkom fakultetu u Sarajevu na odsjeku za računarstvo i informatiku. Motivacija autora za izradu rada je bila da nauči koncepte i nove tehnologije, kao i da sumira kompletno stečeno znanje tokom tri godine studija. Kako je ovo fakultetski projekat i rad, on je namjenjen za prezentaciju i čitanje svim studentima i profesorima iz struke, kao i ostalim čitateljima koje zanima tematika vizualizacije podataka.

Rad se sastoji iz dva dijela, od kojih je prvi dio teoretski, a drugi dio praktični. Pri izradi rada korištena je literatura dostupna na internetu, a koja je referencirana na kraju rada. Također, pri izradi praktičnog dijela rada, autor je koristio isključivo „open source“ alate. Metode koje su korištene pri izradi rada su isključivo vezane za istraživanje i učenje kroz dostupne alate i literature na internetu.

Kao rezultat rada, dobijena je i razvijena funkcionalna „proof of concept“ platforma (dvije aplikacije) koja sumira osnovne i reprezentativne metode prilikom vizualizacije podataka, korištenjem mobilnih i web aplikacija.

Zaključak i mišljenje autora se nalaze na kraju rada.

Abstract – English

To best explain the summary of the paper, it is important to point out that this paper does not fall into the domain of scientific papers, rather it shows a practical paper that has been led with “proof of concept” logic, whose goal is to prove that a set problem is solvable by using certain tools and using previously created logical approaches and possibilities.

The paper is realized as a final paper of the first cycle of study on Faculty of Electrical Engineering Sarajevo on the department of computing and informatics. Author's motivation for the making of the paper was to learn the concepts and new technologies, as it was to summarize the knowledge gathered during the three year-long study. This being a faculty and a project paper, it is intended to be presented and read in front of all students and professors from this field, as it is intended for other readers who are interested in the theme of visualizing data.

The paper is made out of two parts, of which the first part is the theoretical, while the second one is practical. During the making of the paper, literature found on the internet has been used, as it is stated in the end of the paper. Also, while making the practical part of the paper, the author used entirely „open source“ tools. Methods that have been used while making the paper are exclusively linked to research and learning through the available tools and literature available on the internet.

As a result of the paper, a functional „proof of concept“ platform (two applications) has been developed that summarizes the basic and the representative methods during the visualisation of data, by using mobile and web applications.

The conclusion and author's opinion are at the end of the paper.

Sadržaj

Uvod	2
Uvod u temu	2
Ciljevi i metodologija rada:	2
Kratki osvrt na historiju vizualizacije podataka	3
Kratki uvod u REST servise sa naglaskom na dijeljene baze podataka.....	4
Koncepti udaljenih baza podataka - WEB API	5
Dijeljene baze podataka	5
Kako više aplikacija koristi jednu bazu podataka.....	6
Realizacija sloja za komunikaciju aplikacija i baze podataka.....	6
Koncepti vizualizacije podataka.....	10
Zašto vizualiziramo podatke?	10
Principi i vrste vizualizacije podataka	11
Metodologije vizualizacije podataka	15
Biblioteke za vizualizaciju podataka	16
Razvoj aplikacija za prikupljanje i vizualizaciju podataka sa bankomata.....	19
Postavka problema (<i>Case study</i>).....	19
Objašnjenje načina realizacije kroz troslojnu arhitekturu	20
Objašnjenje pojedinačnih dijelova i načina na koji rade.....	21
Backend layer - Database	21
Middleware - Servisi sa NodeJS.....	21
Presentation layer - Web platform sa AngularJS.....	21
Presentation layer - Android aplikacija sa Java.....	22
UML Dijagrami	23
Component dijagram	23
Sequence dijagram	23
Use Case dijagram.....	24
Activity dijagrami.....	25
ER dijagram baze podataka.....	29
Demonstracija aplikacije kroz screenshotove i <i>dijelove koda</i>	30
Tehnički aspekt realizacije aplikacija	30
Web aplikacija.....	30
Mobilna aplikacija.....	40
Korisnički aspekt aplikacija.....	43
WEB aplikacija.....	43

Mobilna aplikacija.....	49
Zaključak	51
Reference.....	53
Literatura.....	53
Slike.....	54
Tabele	55

Uvod

Uvod u temu

Povećanjem broja podataka sa kojima moderne kompanije posluju, javlja se potreba za kvalitetnom klasifikacijom i prikazom istih. Sve više kompanija se odlučuje da unutar svojih poslovnih informacionih sistema implementira i module za vizualizaciju podataka, kako bi sa istim lakše mogli operirati. Prednosti koje nudi kvalitetna vizualizacija podataka se prvenstveno ogledaju u njihovom lakšem iščitavanju, prezentaciji, kao i njihovoj koncentraciji na jednom mjestu. Pored vizualizacije podataka, velika većina modernih infomacionih sistema unutar sebe integriše više različitih platformi koje koriste jedinstvenu bazu podataka. Te platforme su obično web aplikacije, mobilne aplikacije, desktop aplikacije i sl. U ovom radu će biti prikazana sinteza svih gore navedenih koncepata kroz implementaciju jednog konkretnog programskog rješenja, koji će koristiti opisane modele.

Ciljevi i metodologija rada:

Ovaj rad ima za cilj pokazati neke od osnovnih koncepata vizualizacije podataka i korištenja dijeljenih baza podataka. Ovi koncepti će biti prikazani kroz implementaciju jednog jednostavnog programskog rješenja u vidu web aplikacije i android aplikacije. Tehnologije koje su korištene, prilikom realizacije ovih aplikacija su:

- Java (Android)
- AngularJS
- NodeJS sa Express frameworkom
- MySql
- Razne JavaScript biblioteke za vizualizaciju podataka
 - NVD 3
 - Google Maps

Rad je podijeljen na dva dijela, teoretski i praktični dio.

U prvom dijelu rada prikazan je kratki osvrt na historiju vizualizacije podataka, kao i dat uvod koncepta REST servisa sa naglaskom na dijeljene baze podataka. Nakon uvoda, sljedeća dva odjeljka su posvećena teoretskom dijelu, i to prezentaciji načina na koji rade dijeljene baze podataka, kao i metodologijama, alatima i vrstama vizualizacije podataka.

Za praktični dio rada, prikazan je opis i implementacija web i android aplikacije kroz UML dijagrame, specifične isječke koda, kao i screenshotove. Kroz prezentaciju načina implementacije programskog rješenja prikazani su i teoretski osnovi tehnologija koje su korištene pri realizaciji.

U okviru zaključka rezimirano je urađeno, a dat je i osvrt na rad i smjernice za buduće istraživanje vezano uz ovu temu, kao i prijedlozi za unaprjeđivanje kreirane aplikacije.

Kratki osvrt na historiju vizualizacije podataka

Uvriježeno je mišljenje da su statistički grafovi i vizualizacija podataka moderni pristupi obrade podataka, no ustvari grafička reprezentacija informacija i podataka ima duboke korijene u historiji i u najranijim godinama kreiranja mapa i vizualnih prikaza, a kasnije i u tematskim kartama, statistici i statističkim grafovima, medicini i drugim naučno-istraživačkim poljima. Ovo se ogleda u čovjekovoj konstantnoj potrebi za klasifikacijom i kvalitetnoj prezentaciji informacija i podataka, kako bi bilo što razumljiviji i što upotrebljiviji.

Razvoj pristupa vizualizaciji podataka se može posmatrati kroz epohe i razdoblja, od najranijih, prije 17. stoljeća, do najnovijih modernih pristupa. U periodu prije 17. stoljeća, fokus vizualizacije je bio na jednostavnim geometrijskim dijagramima i tabelama u kojima su prikazivane pozicije zvijezda i ostalih nebeskih tijela. Ideja o koordinatnom sistemu je korištena od drevnih Egipćana.

17. stoljeće donosi velike pomake u vizualizaciji podataka i uvodi neke nove koncepte u vizualizaciju. Najveći problemi, u ovoj oblasti, vezani za 17. stoljeće, su oni koji su se bavili fizičkim mjerenjima - vremena, razdaljine, prostora - za astronome, kao i kreiranje mapa, navigacijska i teritorijalna ekspanzija. U ovom stoljeću dolazi i do razvoja analitičke geometrije, koordinatnih sistema, teorije izračunavanja i procjene, kao i rođenje teorije vjerovatnoće (Pascal i Fermat).

Sa začecima teorije vjerovatnoće i statistike, podataka od interesa i važnosti i ideje o grafičkoj reprezentaciji istih, osnovi za ozbiljniji razvoj oblasti vizualizacija podataka su osigurani. 18. stoljeće svjedoči ekspanziji ovih aspekata prema novoj domeni i novim grafičkim formama. U kartografiji, kreatori mapa počinju pokazivati mnogo više informacija, od pukog predstavljanja geografske pozicije na mapi. Kao rezultat toga izumljeni su novi načini prezentacije podataka (izolinije i konture), kao i tematsko mapiranje fizičkih kvantitativnih podataka. Krajem ovog stoljeća se vide prvi pokušaji tematskog mapiranja u geografskih, ekonomskih i medicinskih podataka. S tehničkog aspekta, razvoju svega navedenog, pomaže i izum trobojnog printera, kojeg je osmislio Jacob le Blon. Wiliam Playfair se smatra inovatorom većine današnjih oblika prezentacije podataka, kao što su linijski grafovi i barčartova (grafikona), a kasnije i piečartova i kružnih grafova.

Prva polovina 19. stoljeća se smatra kao početak moderne vizualizacije. U statističkoj grafici, sve moderne forme prikaza podataka su unaprijeđene ili osmišljene. Neke od njih su, bar i pie čartovi, histogrami, linijski grafovi, time-series plotovi, kontur plotovi... U tematskoj kartografiji, dolazi do unaprjeđenja mapa, pa se počinju koristiti opsežni atlasi sa tematskim kartama koje prikazuju razne podatke od interesa, kao što su, ekonomski, socijalni, fizički, medicinski...

Druga polovina 19. stoljeća se još može nazvati i zlatno doba statističke grafike. U ovom periodu počinju pokušaji reprezentacije podataka sa više od dvije varijable i pojavljuju se prvi trodimenzionalni grafovi. Posebno u kartografiji javljaju se izoleveli, podijeljeni kružni dijagrami koji prikazuju razne podatke po totalima, sektorima, područjima i sl.

Ako se druga polovina 19. stoljeća može nazvati zlatno doba statističke grafike, onda za prvu polovinu 20. stoljeća se može reći da je to mračno doba moderne vizualizacije podataka. U ovom periodu nije izmišljeno ništa novo na polju vizualizacije podataka, osim što su napravljena neka poboljšanja u dosadašnjim. Počinju se ozbiljnije koristiti textboxovi kao legende pored grafova, i za objašnjenje grafova. Kompletan ovaj period je prošao u čekanju na tehnološke i naučne inovacije. Inovacije na polju moderne statističke metodologije,

prednosti kompjuterske snage i prednosti jačine uređaja za prikaz podržale su ponovno rođenje ozbiljnih i danas poznatih metodologija vizualizacije podataka i grafičke prezentacije. Od 1975. godine do danas vizualizacija podataka je ušla u novu eru High-D, interaktivne i dinamičke vizualizacije podataka. Pojavom brzih procesora, novih statističkih alata, kao i razvojem medija za prikaz podataka, vizualizacija podataka uzima sve veći mah u industriji, nauci, ekonomiji... Velika količina podataka koja je došla sa ekspanzijom interneta i veliki broj korisnika tog medija je bila plodno tlo za razvoj interaktivnih mapa, grafika i sličnih oblika prezentacije i vizualizacije podataka. Sve više inženjera kreira vlastite biblioteke za vizualizaciju podataka, pa čak i velike kompanije kao što su Google i facebook imaju svoje module za vizualizaciju podataka u raznim poljima, mape, grafici... U dijelu ovog rada o vizualizaciji podataka će detaljnije biti prikazane neke moderne metode vizualizacije podataka kao i alati koji se koriste za istu. [1]

Kratki uvod u REST servise sa naglaskom na dijeljene baze podataka

Ekspanzijom interneta, i inovacijama u tehnologijama, sve više uređaja ima pristup otvorenoj mreži. Nove tehnologije omogućavaju integraciju namjenskih uređaja u informacione sisteme, poznato još kao i IoT (Internet of Things). Također, razvoj mobilnih uređaja i prijenosnih računara omogućio je da se veliki broj podataka može prikupljati, obrađivati i isporučivati geografski decentralizovano. Kako bi se iskoristili benefiti navedenih tehnologija, bilo je potrebno osmisliti načine kojim će se omogućiti kreiranje, brisanje, mijenjanje, povlačenje podataka iz baza podataka za različite uređaje. Većina današnjih informacionih sistema nastoji da unutar sebe integrira dvije ili više platformi, počevši od klasičnih web, desktop aplikacija, preko mobilnih aplikacija, do ugradbenih sistema. Tu susrećemo pojam dijeljenih baza podataka. Pod pojmom dijeljenih baza podataka se podrazumijeva jedna centralizovana baza podataka koju koristi više aplikacija i platformi. Na ovaj način, sve aplikacije, ili familije aplikacija koje koriste tu jednu bazu podataka su uvijek konzistentne u smislu podataka koje dobijaju. Postoje razni modeli koji omogućavaju korištenje ove vrste arhitekture, ali onaj koji je zanimljiv za ovaj rad je model realizacije servisa koji su sloj između baze podataka i aplikacija koje koriste tu bazu podataka. Ovaj model se često susreće i pod nazivom SOA, ili servis orijentirane arhitekture. Naime, SOA je stil dizajniranja informacionih sistema gdje različiti dijelovi tih sistema komuniciraju jedni sa drugima, preko servisa koji mogu biti servisi koji traže, ili isporučuju neke podatke ili akcije.

REST (**RE**presentational **S**tate **T**ransfer) je postao skoro standard za WEB i mobilne aplikacije u zadnje vrijeme. O čemu se tu radi? Svaki sistem koristi neku vrstu resursa, bilo da su to slike, audio zapisi, ili najjednostavniji tekstualni zapisi. Svrha servisa je da omogućiti klijentu (aplikaciji, platformi, dijelu softvera) da pristupi tim podacima i resursima. Servis arhitekti i razvojni inženjeri imaju za cilj da ti servisi budu laki za implementaciju, održavanje i nadogradnju. RESTful dizajn omogućava upravo to i mnogo više. Generalno RESTful servisi podrazumijevaju implementaciju svih potrebnih akcija za manipulaciju resursima unutar aplikacije ili familije aplikacija. U REST arhitekturi, server jednostavno omogućava pristup, izmjenu i brisanje resursa. Svaki od tih resursa je definiran svojim URI-em, odnosno njegovim jedinstvenim ID-om. REST koristi različite oblike isporuke podataka, kao što su Text, XML(**eX**tensible **M**arkup **L**anguage), JSON (**J**ava**S**cript **O**riented **N**otation). JSON je trenutno najpopularniji format korišten u RESTful web servisima. [2]

U narednim poglavljima je objašnjena realizacija WEB-API servisa kroz primjere koda.

Koncepti udaljenih baza podataka - WEB API

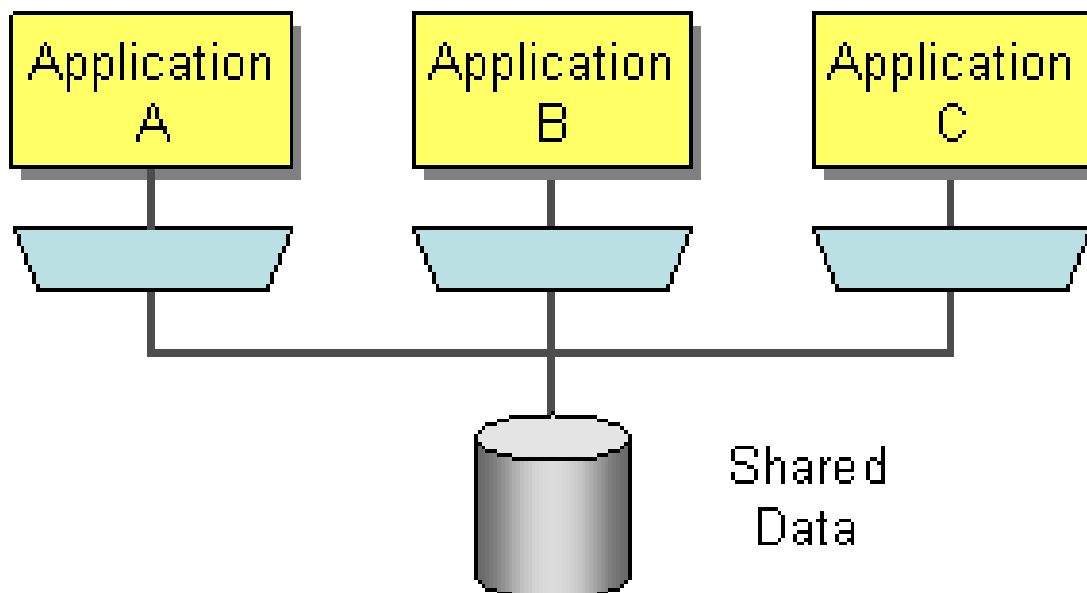
Dijeljene baze podataka

Kako bi se što bolje razumio koncept dijeljenih baza podataka, poželjno je da objašnjenje istog ide kroz praktičan primjer iz prakse. U realnim enterprise sistemima, nerijetko se susreće veliki broj aplikacija koje su građene nezavisno jedna od druge i pisane u različitim programskim jezicima i razvijane na različitim platformama. Kako su te aplikacije vezane za isto poslovanje i kreirane za slične namjene, logičan slijed događaja jeste da koriste iste, ili slične podatke. Aplikacije koje se kreiraju odvojeno, nezavisno i na različitim platformama, često koriste vlastite baze podataka u kojima čuvaju podatke od interesa. Pošto su aplikacije razvijane da rade za korist jednog poslovanja i procesa vezano za njega, postavlja se pitanje kako da se održi i postigne konzistencija, kao i brzina razmjene podataka između tih aplikacija. [3]

Problem koji se javlja kod situacije u kojoj svaka aplikacija koristi vlastitu bazu podataka je to što je otežana razmjena podataka između aplikacija u realnom vremenu. S druge strane, osiguravanje konzistencije podataka je jako upitno, zbog toga što odvojene baze podataka ne moraju, i najčešće nisu uniformisane što se tiče oblika u kojem čuvaju podatke.

Ova dva glavna razloga su motiv za pristupanje modelu dijeljenih baza podataka. Kao što samo ime kaže, dijeljena baza podataka je baza podataka koju dijeli više aplikacija, i u kojoj se nalaze svi podaci potrebni za funkcioniranje različitih aplikacija i različitog broja aplikacija (gdje aplikacije ne moraju da brinu o tome kako su podaci sačuvani u bazi).

Grafički prikazano, to izgleda kao na slici ispod.



Slika 1 - Šema dijeljene baze podataka

Ovaj model čuvanja i razmjene podataka između aplikacija omogućava da sve aplikacije u svakom trenutku mogu imati konzistentne podatke i da tim podacima pristupaju na uniforman i brz način.

Kako više aplikacija koristi jednu bazu podataka

Prirodno je postaviti pitanje, na koji način više aplikacija može da koristi jednu bazu podataka. Odgovor na to pitanje leži u pristupu da se kod ovog modela čuvanja podataka kreira jedan sloj koji se nalazi između baze podataka i aplikacija koje koriste te podatke, a koji omogućava upravljanje bazom podataka. Pod upravljanjem bazom podataka se podrazumijeva upis, izmjena, brisanje podataka iz baze, kao i čitanje istih. Također, ovaj sloj omogućava uniforman pristup podacima, svim aplikacijama od interesa, pa tako sve aplikacije koje konzumiraju podatke iz dijeljene baze ne treba da znaju šta se nalazi u pozadini “ispod haube”, nego samo oblik u kojem podatke dobijaju od sloja koji je zadužen za nad bazom podataka. Spomenuti princip je objašnjen na praktičnom primjeru u poglavlju 4, gdje dvije različite aplikacije na različitim platformama koriste istu bazu podataka.

Realizacija sloja za komunikaciju aplikacija i baze podataka

Kada je koncept dijeljenih baza podataka shvaćen, može se početi govoriti o tehničkoj realizaciji ovog modela. U ovom radu, naglasak će biti na WEB-servis konceptu realizacije sloja za komunikaciju aplikacija i baze podataka. WEB-servis je koncept u kojem je ovaj sloj realiziran kao WEB aplikacija koja ima bazu podataka nad kojom manipulira podacima (upisuje, briše, modifikuje, čita). Također, ta WEB aplikacija omogućava drugim aplikacijama da preko nje manipuliraju podacima koji se nalaze u toj bazi, a preko servisa i interfejsa koje omogućuje drugim aplikacijama.

Podaci koje WEB aplikacije pružaju su obično formatirani u nekom od standardizovanih oblika za web aplikacije, a to su najčešće XML ili JSON. Svaki servis kojeg aplikacija pruža “vanjskom svijetu” je definisan svojim jedinstvenim URI-em koji adresira koji dio servisa se poziva i na koji način.

Da bi se dobila kompletnija slika o onome što je gore napisano, možemo uzeti jedan primjer. Recimo da je aplikaciji koja se razvija potrebno da dobije podatke o svim korisnicima koji se nalaze u nekoj tabeli u bazi podataka. WEB servis koji omogućava pristup tim podacima, je definisan sljedećim URI-em:

<https://www.aplikacija.com/api/Korisnici>

Kada se ova adresa ukuca u browser, ili pozove iz aplikacije, WEB servis vraća odgovor u JSON formatu, u kojem su sadržane informacije o svim korisnicima.

Taj odgovor može da izgleda kao u tabeli ispod.

```
[
  {
    "idKorisnik":1,
    "ime":"Benjamin",
    "prezime":"Ramić",
    "korisnickoIme":"bramic",
    "lozinka":"*****"
  },
  {
    "idKorisnik":9,
    "ime":"Belmin",
    "prezime":"Mustajbašić",
    "korisnickoIme":"bmustajbas",
    "lozinka":"*****"
  },
  {
    "idKorisnik":11,
    "ime":"Vedad",
    "prezime":"Ramić",
    "korisnickoIme":"Vedo02",
    "lozinka":"*****"
  },
  {
    "idKorisnik":12,
    "ime":"Sanil",
    "prezime":"Musić",
    "korisnickoIme":"smusic",
    "lozinka":"*****"
  },
  {
    "idKorisnik":26,
    "ime":"Minka",
    "prezime":"Ramić",
    "korisnickoIme":"mramic",
    "lozinka":"*****"
  },
  {
    "idKorisnik":28,
    "ime":"Jasmin",
    "prezime":"Ramić",
    "korisnickoIme":"Jramic",
    "lozinka":"*****"
  },
  {
    "idKorisnik":30,
    "ime":"Ahmed",
    "prezime":"Popović",
    "korisnickoIme":"apop1",
    "lozinka":"*****"
  }
]
```

Tabela 1 – Odgovor u JSON formatu sa više objekata

Odgovor koji je gore prikazan se tumači na način da je pozvani servis vratio JSON niz u kojem su sadržani JSON objekti tipa Korisnik, koji ima atribute, "idKorisnik", "ime", "prezime", "korisnickoIme", "lozinka".

Na isti, ili sličan način se može kreirati i bilo kakav drugi servis, koji će pružati drugačije podatke. Bitno je spomenuti da se u URI-u mogu specificirati i neki parametri koji će služiti za preciznije definisanje onoga šta aplikacija očekuje od WEB Servisa, pa tako u prethodnom URI-u je moguće da dodamo informaciju o ID-u specifičnog korisnika kojeg želimo iz baze, pa će onda taj link izgledati ovako:

<https://www.aplikacija.com/api/Korisnici/1>

Odgovor koji bi nam WEB servis mogao ponuditi u ovom slučaju je prikazan u tabeli ispod:

```
[
  {
    "idKorisnik":1,
    "ime":"Benjamin",
    "prezime":"Ramić",
    "korisnickoIme":"bramic",
    "lozinka":"*****"
  }
]
```

Tabela 2 – Odgovor u JSON formatu sa jednim objektom

Razlika je odmah uočljiva, primjećuje se da je sada za odgovor dobijen samo jedan korisnik, čiji je ID specificiran kroz gore navedeni URI.

Parametri u URI-e se mogu slati na dva načina, GET metodom i POST metodom. GET metodom, parametre specificiramo odmah u URI-u, kao što je to pokazano na primjeru iznad, dok se sa POST metodom parametri šalju unutar tijela HTTP zahtjeva.

Bitno je da se naglasi da servisi nisu namjenjeni samo za dobavljanje podataka, nego servisi mogu da implementiraju i brisanje, izmjenu i dodavanje podataka. Ono što je bitno za realizaciju ovih funkcionalnosti je da se kreira i specificira URI koji se poziva servis koji će obaviti neku od ovih akcija. Uzmimo za primjer da želimo da izbrišemo korisnika iz baze sa ID-om 1. Servis kojeg koristimo bi mogao da implementira logiku potrebnu za to, a za pristup tome bi se mogao koristiti sljedeći URI:

<https://www.aplikacija.com/api/ObrisiKorisnika/1>

Nakon što se pozove ovaj link, servis obavi brisanja i validaciju oko brisanja i vraća odgovor, u kojem se sadrži informacija o uspješnosti, ili neuspješnosti brisanja. Na sličan način se mogu kreirati i servisi za dodavanje i izmjenu podataka, a podaci koji treba da se dodaju ili izmjene se mogu specificirati kroz GET, ili POST metode.

Da bi se gore navedeni koncept standardizovao, kreirana je ideja oko REST-a (**RE**presentational **S**tate **T**ransfer), koji ima za cilj da utvrdi neka pravila po kojima se WEB servisi treba da kreiraju. U ovom standardu koriste se HTTP metode (GET, POST, PUT, DELETE) i HTTP kôdovi (200, 404, 500...) kako bi se odredila operacija koju servis treba da obavi.

Ukratko:

- Za preuzimanje objekta koristi se metoda GET.
- Za kreiranje novog objekta koristi se POST.
- Za izmjenu postojećeg objekta koristi se PUT.
- Za brisanje objekta koristi se HTTP metoda DELETE.

Razvojem ovih koncepata, i u procesu standardizacije pojavio se veliki broj frameworka, za realizaciju tzv. RESTful WEB servisa. U tabeli ispod je dat pregled nekih od najpoznatijih frameworkova koji se koriste pri kreiranju WEB servisa. [4]

Naziv frameworka	Kratak opis
ASP.Net Core	Microsoftova tehnologija koja je Microsoftov odgovor na open source frameworkove koji se mogu izvršavati na više platformi. Microsoft je svoj .NET framework postepeno prebacio na .NET core framework.
Spring (Java)	Spring je open source framework koji omogućava Java developerima da kreiraju web servise na različite načine koristeći module koji im omogućavaju i olakšavaju različite funkcionalnosti.
Express (Node.js)	Express jako brzo postaje jedan od najpopularnijih frameworka koji se koriste za brzu izgradnju dinamičkih web aplikacija. Node.js omogućava web developerima da sa jezikom koji već koriste za client-side (JavaScript) kreiraju server-side funkcionalnosti.
Loopback (Node.js)	Loopback je još jedan od dinamičkih node.js baziranih web frameworka. Kreiran i održavan je od strane StrongLoopa i IBM-a. Kompanije imaju za cilj da kreiraju framework koji je dobro testiran i spreman da bude ubačen u produkcijske stekove kao pouzdan framework.
Restify (Node.js)	Restify ima za cilj da bude reducirana, bolje formirana verzija Expressa. Express ima mnoge funkcionalnosti koje nisu potrebne za kreiranje REST WEB API-a, pa s tim u vezi, Restify ima za cilj da kreira čist, lagan alat za kreiranje REST API-a i web socket baziranih aplikacija.
Django (Python)	Django je open-source web framework za python developere. Dizajniran je da kreira kompleksne database driven web stranice i ima princip ponovo iskoristivog koda u modulima. Ima veoma veliki set funkcionalnosti.
Flask (Python)	Flask je izbor za kreiranje lightweight REST API-a u pyhtonu. Ima veoma mali uticaj na resurse sistema i omogućava veoma lagane metode koje daju informacije kao što su HTTP endpoints u formatu koji je lagan za čitanje na strani klijenta.
Ruby on Rails	Ruby on Rails je framework za web aplikacije za Ruby developere. To je MVC (Model View Controller) bazirani framework koji renderuje web stranice i web servise. Koristeći RubyGems system, ima mogućnosti dodavanja hiljada modula koji omogućavaju mnoge funkcionalnosti, a koje ne zahtjevaju previše kodiranja.

Tabela 3 – Pregled najpoznatijih frameworka za izradu WEB servisa[5]

Za realizaciju praktičnog dijela ovog rada, framework koji je korišten je Express koji radi sa Node JS.

Koncepti vizualizacije podataka

Zašto vizualiziramo podatke?

Američki matematičar, John Tukey je rekao: *“Najveća vrijednost slike je kada nas prisili da primjetimo ono što nikada ne bi očekivali da vidimo”*, upravo to može biti odgovor na pitanje, zašto vizualiziramo podatke. Ozbiljnim razvojem statistike, a lata za prikupljanje podataka kao i velikom količinom podataka koji bivaju prikupljeni, sve je teže podatke analizirati i iz njih izvući kvalitetne informacije (informacije od interesa). U eri, takozvanih “velikih podataka”, gdje se u svakom trenutku, u svim sferama života, politici, informatici, ekonomiji, trgovini, školstvu i slično, sakupljaju enormne količine podataka, koje imaju tendencije ka stalnom mijenjanju, pa čak često i nepredvidivom mijenjanju, klasična analiza podataka susreće probleme pri pokušaju da efikasno izvuče korisne informacije iz svih tih podataka.

Pitanje koje se postavlja, u sistemima sa jako puno podataka je šta se sa tim podacima može uraditi, i kako iskoristiti njihove benefite. Cilj analize podataka je, u većini slučajeva da se iz njih izvuku njihove osobine, uoče paterni, predvide ponašanja i slično. Još jedno od pitanja koje se postavlja, je pitanje brzine analize podataka.

Sva ova pitanja i izazovi su bili pokretač da vizualizacija podataka, kao oblik analize i prikaza podataka zaživi i doživi nagli razvoj. Trend vizualizacije podataka se najviše očituje u biznisu i procesu donošenja odluka, gdje sve veći broj kompanija ulaže napore da informacije koje imaju u svom posjedu kvalitetno vizualiziraju. Razlozi tome su višestruki, od kojih će samo neki biti navedeni ovdje:

- Laka i brza sistematizacija podataka

Ovo ne iznenađuje, jer 90% od svih informacija koje dopru do ljudskog mozga su “u slikama”. S tim u vezi, dobro kreirani grafovi, mape i kombinacije istih, mogu da znatno skrate vrijeme analize bitnih skupova podataka.

- Vizualizacija podataka omogućuje jednostavnost

Zaista, koristeći vizualizaciju, kao metodu analize podataka, na jednostavniji način dobijamo širu sliku o analiziranim podacima, a u isto vrijeme dobijamo i bitne detalje, što omogućava baratanje sa samo bitnim podacima

- Lakše uočavanje paterni

Prolaženje kroz veliku količinu podataka u formi tabela, ili lista, otežava primjećivanje paterni i zakonitosti ponavljanja u podacima. Vizualizacija omogućava lakšu absorpciju podataka i uočavanja zakonitosti u istim.

- Olakšana kolaboracija tima

Timovi, koji rade na analizama podataka, ili prosto timovi koji su uključeni u procese odlučivanja, mnogo lakše mogu kolaborirati kada imaju podatke prikazane na sistematičan, klasificiran, lako-pristupačan način.

- Lakše poređenje podataka

Poređenje podataka u cilju poboljšanja poslovanja može ići na različite strane. Upoređivanje internih skupova podataka, upoređivanje internih podataka sa podacima konkurentnih poslovanja, ili upoređivanje podataka sa javnim istraživanjima i javnim podacima je mnogo lakše, kada su oni prikazani na isti, ili sličan način, a koji omogućava njihovo lahko poređenje.

Svi gore navedeni benefiti su razlog da se vizualizacija posmatra kao ozbiljno rješenje problema analize skupova podataka. [6]

Principi i vrste vizualizacije podataka

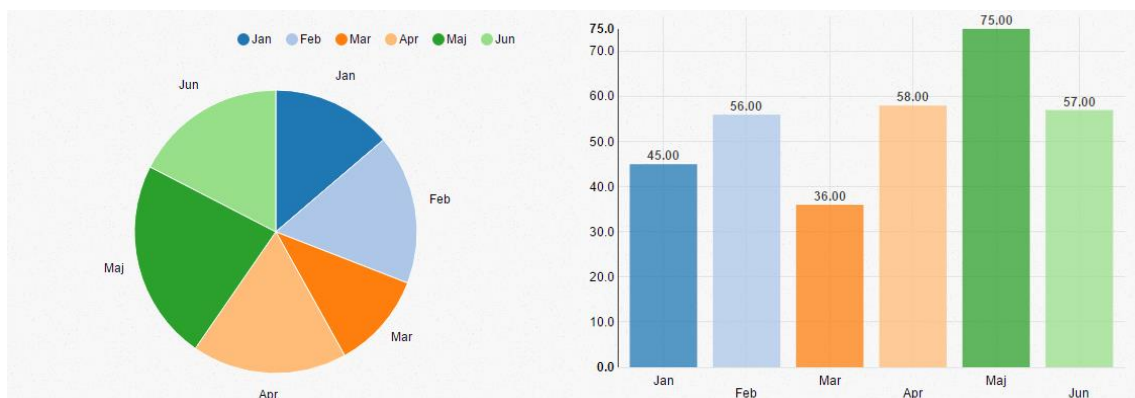
Uzmimo jedan primjer, kako bi lakše dobili sliku o tome, koliko vizualizacija može ubrzati, olakšati i unaprijediti proces analize podataka.

Neka imamo neku kompaniju koja ima polugodišnje rashode, i da je data tabela sa ukupnim rashodima u hiljadama dolara, po mjesecima u prvoj polovini godine. Potrebno je uočiti mjesec sa najvećim rashodima.

Mjesec	Januar	Februar	Mart	April	Maj	Juni
Rashodi	45	56	36	58	75	57

Tabela 4 – Pregled polugodišnjih rashoda po mjesecima

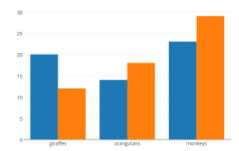
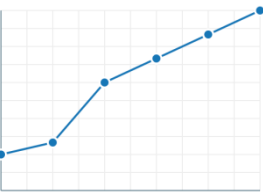
Isti ti podaci, vizualizirani, nam daju istu informaciju u znatno manjem vremenu, za sekundu ili dvije. Dajmo primjer istih tih podataka, vizualiziranih:

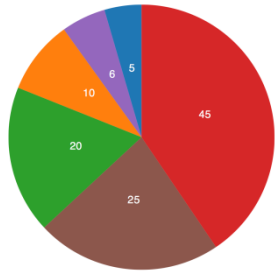
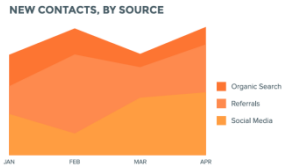
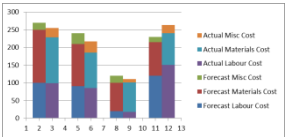
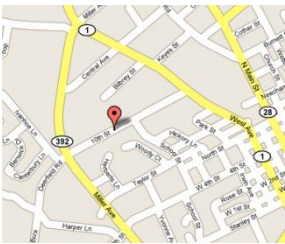


Slika 2-Pie i Bar chart gore, tabelarno, prikazanih podataka

Naravno, ovaj primjer je jako malog obima, tako da se ne može na pravi način prikazati značaj i olakšice koje dobijaju putem vizualizacije. Tako naprimjer ako je potrebno pronaći dan u godini sa najvećim rashodima, koji je također prikazan u tabeli od 365 kolona, i gdje su podaci u užem rasponu vrijednosti, onda gore navedeni primjer zaista može približiti zašto je vizualizacija bitna.

Analiza podataka vizualizacijom može imati dva cilja, objašnjavanje podataka, ili istraživanje istih. Oba ova cilja zahtjevaju dobre metode vizualizacije. Bilo da želimo predstaviti set jednostavnih podataka, ili set komplikovanih podataka velikog obima, koristi se širok set grafova, mapa, ili kombinacije istih za vizualizaciju. Od veoma mnogo načina i metoda na koji se podaci mogu vizualizirati, u narednoj tabeli će biti prikazani samo oni koji su interesantni za implementacioni dio ovog rada, s fokusom na razne vrste statističkih grafova i geo-mapama.

Naziv	Objašnjenje	Kada se koristi	Grafički primjer
Bar chart	Jedan od najčešćih načina vizualizacije podataka. Zašto? Brz je za komparaciju informacija, otkrivanje najviših i najnižih vrijednosti u nizu vrijednosti. Posebno je pogodan kada su podaci podjeljeni u kategorije, tako da se brzo mogu primjetiti trendovi u podacima.	Upoređivanje podataka po kategorijama. Primjer: Procenti potrošnje po odjelima kompanije.	
Line chart	Ovi Chartovi su odmah poslije Bar chartova na listi najčešće korištenih. Line chartovi spajaju pojedinačne numeričke podatke. Rezultat je jednostavno prikazana sekvenca vrijednosti. Primarno se koriste da prikažu trendove u određenom periodu vremena.	Pregledanje trendova u vremenskom periodu. Primjer: Promjena trendova na berzi u roku od pet godina	

Pie chart	Pie chart bi trebao da se koristi za prikaz relativne proporcije ili procenat informacija. To je zapravo njihova jedina svrha. Iako je njihova definicija prilično jednostavna, često se pogrešno koriste. Zato je bitno uvijek napomenuti za šta je namijenjen pie chart.	Prikazivanje proporcija. Primjer: Procenat trošenja različitih odjela u kompaniji.	
Area chart	Area chart je u suštini line chart, ali prostor između x ose i linije je popunjen nekom bojom. Koristan je za prikazivanje dio-dio-cjeline relacija. Pomaže pri analizi ukupnih i pojedinačnih trendova datih podataka.	Ovu vrsta chartova se koristi za pregled učešća pojedinačnih trendova, u ukupnom trendu podataka. Također se može koristiti za kumulativne serije vrijednosti.	
Stacked bar chart	Ovakav chart prikazuje komparaciju različitih stavki, kao i kompoziciju svake od kompariranih stavki.	Koristi se kada se želi prikazati šta je uticalo i od čega su sastavljene stavke koje se porede.	
Mape	Mape su izuzetno korisne za prikazivanje takozvanih lokacijskih podataka. Bilo da su ti podaci poštanski brojevi, rasporedi lokacija, nazivi lokacija, rasporedi i lokacije određenih vrijednosti u državi,	Koriste se za prikazivanje geokodiranih podataka. Primjer: Lociranje svih poslovnica neke korporacije.	

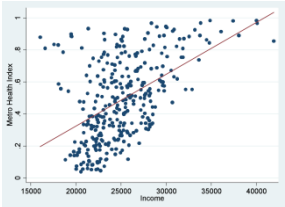
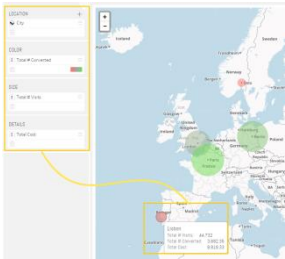
	gradu, mjestu, i slično.		
Scatter plot chart	Korisni za malo dublji pogled u podatke, ali koji nemaju neke jasne veze i poveznica. Scatter plot chartovi mogu dati efektan uvid u trendove, koncentraciju podataka i slično, a u cilju da pokažu gdje treba da se fokusira dalje istraživanje podataka.	Istraživanje relacija između više različitih promjenjivih podataka. Primjer: troškovi dostave različitih kategorija proizvoda u različite regije.	
Scatter Map / Area Map chartovi	Ovaj vid vizualizacije podataka pomaže da se lakše vizualiziraju geografski raspoređeni podaci, organizirani kao skupovi tačaka na mapi. Može se vizualizirati više različitih vrsta setova podataka preko mape, u cilju upoređivanja.	Istraživanje relacije numeričkih i geografski orijentiranih podataka.	

Tabela 5 –Pregled najreprezentativnijih oblika vizualizacije podataka [7][8]

Gore u tabeli su prikazani najreprezentativniji oblici vizualizacije podataka, i najčešće korišteni. Upravo oni će biti korišteni za realizaciju praktičnog dijela ovog rada. Osim ovih vrsta vizualizacije, postoje razne varijacije gore navedenih mapa i grafova, kao i neki drugi, novi oblici i vrste vizualizacije. Ovdje se mogu spomenuti samo neki od njih, kao što su, mrežni chartovi, stabla, heat mape, bullet chartovi, waterfall chartovi, bubble chartovi, histogrami, highlight map i slični.

Metodologije vizualizacije podataka

Kako bi podaci bili kvalitetno vizualizirani, potrebno je napraviti sistematičan i obuhvatan redoslijed koraka koji vode do kvalitetne vizualizacije. Postoje razne metodologije za vizualizaciju podataka, ali manje-više, kroz sve se prožima slična filozofija. Ovdje će biti prikazani preporučeni i skoro ustaljeni koraci za vizualizaciju podataka, a po kojima će se vršiti i realizacija praktičnog dijela ovog rada.

Prvo je potrebno shvatiti da je vizualizacija podataka kombinacija više različitih disciplina. Kako bi se ponudilo smisleno vizualizirano rješenje, potrebno je uključiti više disciplina, kao što su: statistika, data mining, grafički dizajn, vizualizacija informacija. Kakogod, svako od ovih polja koje je uključeno u realizaciju vizualizacije je u izolaciji od ostalih. Kako bi ove izolirane discipline mogle raditi skupa na vizualizaciji, potrebno je osmisliti strategiju po kojoj će se podaci prikupljati, obrađivati i vizualizirati.

U skladu s time, nudi se sljedeći niz koraka:

- **Prikupljanje podataka**

Prikupljanje podataka podrazumijeva objedinjavanje podataka na jedno mjesto, od bilo kuda da dolaze, sa diska, sa interneta, sa geo-dislociranih stanica, uređaja i slično.

- **Raščlanjivanje**

Osiguravanje neke strukture koja daje podacima značenje i raspoređivanje istih u kategorije.

- **Filtriranje**

Uklanjanje svih podataka, osim onih od interesa.

- **Mining - "Iskopavanje"**

Primjenjivanje metoda iz statistike ili data-mininga kao način da se raspoznaju paterni među podacima, ili da se jednostavno podaci prikažu u matematičkom kontekstu.

- **Reprezentacija**

Izabiranje osnovnog vizualnog modela, kao što su chartovi, mape i slično.

- **Prečišćavanje**

Unaprjeđivanje osnovnog vizuelnog modela, u svrhu jasnijeg prikaza podataka i vizuelne privlačnosti.

- **Interakcija**

Dodati metode za manipulaciju podacima ili za kontrolu načina prikaza. [9]

Kod reprezentacije podataka i biranja modela za prikaz podataka, bitno je odgovoriti na neka pitanja, kako bi se što lakše došlo do odgovora kakav graf ili mapu koristiti.

Pitanje:	Savjet:	Šta koristiti:
Da li se podaci porede?	Ovi chartovi su odlični za komparaciju jednog ili više setova podataka. Vrlo lahko mogu pokazati minimume i maksimume u skupovima podataka.	<ul style="list-style-type: none">• Column• Bar• Circular Area• Line• Scatter Plot• Bullet

Da li treba da bude prikazana kompozicija nečega?	Koristiti ovu vrstu chartova kada se želi pokazati kako individualni dijelovi kreiraju neki sistem podataka.	<ul style="list-style-type: none"> • Pie • Stacked Bar • Stacked Column • Area • Waterfall
Da li je potrebno razumijevanje distribucije podataka?	Ovi chartovi pomažu boljem razumijevanju granice podataka, njihovu tendenciju, kao i rang informacija u podacima.	<ul style="list-style-type: none"> • Scatter Plot • Line • Column • Bar
Da li je potrebno analizirati trendove u skupu podataka?	Ako je potrebno znati više informacija o tome kako se skup podataka ponaša u toku nekog vremena ovi chartovi pomažu u tome.	<ul style="list-style-type: none"> • Line • Dual-Axis Line • Column
Da li je potrebno bolje razumijevanje relacija između više različitih skupova podataka?	Ovi chartovi su odlični za prikazivanje kako se jedna varijabla odnosi sa više različitih, ili kako jedna varijabla utiče na više različitih.	<ul style="list-style-type: none"> • Scatter Plot • Bubble • Line

Tabela 6– Pitanja i savjeti vezani za odabir oblika vizualizacije[10]

Naravno, sva ova pitanja i koraci ne trebaju biti slijeđeni slijepo. Za očekivati je da će oni biti involvirani različito u različitim projektima, nekada svi, nekada samo dio, ali je bitno da se ovi koraci, u skladu sa potrebama projekta realiziraju. Također, treba obratiti pažnju, da ovaj slijed koraka i pitanja, često ne mora da se izvršava ovim redom, ponekada će postojati potreba za ponavljanjem nekih koraka i vraćanje unazad, kako bi konačna vizualizacija bila zadovoljavajuća.

Biblioteke za vizualizaciju podataka

Razvojem interneta, internet tehnologija i pristupu velikom broju podataka i web stranica, rodila se potreba za IT alatima za vizualizaciju podataka. Ti alati se mogu podjeliti u inženjerski orijentirane alate, koji su predodređeni za korištenje od strane razvojnih inženjera, koji te alate koriste unutar svojih softverskih rješenja i user-friendly alate, koji su predodređeni da budu korišteni od bilo koga ko ima dovoljnu količinu znanja za unos i obradu podataka, a nakon toga alati automatski generišu vizualizacijske chartove.

Kako će praktični dio ovog rada biti realiziran u setu tehnologija koje su JavaScript orijentirane, u ovom odjeljku će biti predstavljene najkorištenije biblioteke za vizualizaciju podataka, a koje su realizirane u JavaScript programskom jeziku.

Naziv biblioteke	Opis	Licenca	Link na dokumentaciju
Chartist.js	Jako intuitivna biblioteka koja ima velikupodršku comuntiya. Posjeduje osnovne statističke grafove i animacije istih.	Open - source	http://gionkunz.github.io/chartist-js/
Fusioncharts	Jedna od najopsežnijih JavaScript biblioteka za vizualizaciju. Ima podršku velike baze uređaja, na kojima se može prikazivati. Podržava veliki broj formata podataka za vizualizaciju.	Besplatan za nekomercijalne projekte.	http://www.fusioncharts.com/
Dygraphs	Odlična biblioteka za vizualizaciju velikih setova podataka. Podržava zumiranje u dijelove grafova i prilagođena je za touchscreen uređaje.	Open - source	http://dygraphs.com/
Chart.js	Ova biblioteka ima super podršku za manje projekte, i koristi se kada je potrebno kreirati, male, elegantne chartove koji su brzi.	Open - source	http://www.chartjs.org/
Google charts	Google charts daju širok spektar chartova za skoro sve vrste i oblike podataka.	Besplatno, ali ne Open - source	https://developers.google.com/chart/
Highcharts	Interaktivna charting bibiloteka koja je bazirana na HTML5/SVG/VML. Podržava veliku većinu poznatih grafova.	Besplatan za nekomercijalne projekte.	http://www.highcharts.com/
Flot	Jedna od najstarijih charting biblioteka. Daje potpunu kontrolu nad iscrtavanjem grafova. Fokusira se na jednostavno korištenje i interakciju grafova.	Open - source	http://www.flotcharts.org/
D3.js	Jako snažan open source projekat koji omogućava kreiranje odličnih vizualnih	Open - source	https://d3js.org/

	efekata kontrolirajući DOM i dodavanjem childova u DOM.		
n3-charts	Jednostavna, brza biblioteka koja dobro radi sa AngularJS-om. Ne posjeduje veliku bazu chartova.	Open - source	http://n3-charts.github.io/line-chart/
NVD3	Projekat koji za cilj ima da kreira ponovo iskoristive chartove i komponente za D3.js. Posjeduje veliki broj chartova, koji se mogu integrirati.	Open - source	http://nvd3.org/
Ember charts	Biblioteka koja odlično radi sa EmberJS. Ova biblioteka omogućava lako korištenje, proširivanje chartova koji su kreirani na bazi D3.js i Ember.js frameworka.	Open - source	http://opensource.addepar.com/ember-table/#/overview
Sigma JS	Veoma specijalizirana biblioteka za interaktivne mape i mreže. Posjeduje ogromnu bazu interaktivnih postavki.	Open - source	http://sigmajs.org/
Google maps	Najrazvijenija platforma za mapiranje i prikazivanje georijentiranih podataka. Posjeduje veliku bazu podataka, podršku od strane Google-a.	Besplatno, ali ne Open - source	https://developers.google.com/maps/

Tabela 7– Pregled biblioteka za vizualizaciju[11]

Razvoj aplikacija za prikupljanje i vizualizaciju podataka sa bankomata

Postavka problema (*Case study*)

Da bi bili prikazani svi iznad objašnjeni principi, razvijana je platforma (dvije aplikacije) pod nazivom "VisualiseATM".

Objašnjenje i postavka problema:

Potrebno je razviti sistem za nadzor, administraciju i praćenje statistika bankomata jedne banke. Ovaj sistem treba da integriše komunikaciju mreže svih bankomata, kao i platformu za nadzor i praćenje svih bankomata. Pod nadzorom i komunikacijom mreže bankomata se podrazumijeva praćenje lokacije bankomata, stanja novca u bankomatu i svih transakcija u istom. Praćenje statistike mreže bankomata je primarni cilj rješavanja ovog problema. Pod ovim se podrazumijeva da se sve statistike, transakcije, količine novca, broj transakcija, količina novca po transakciji i slično mogu pratiti sa jednog centralizovanog mjesta. Platforma koja bi bila zadužena za sve gore navedeno bi trebala da uz prosto ispisivanje svih ovih podataka, pravi i vizualne prikaze statistika, geolokacija kroz prilagođene geo i statističke grafove. Osim ovoga, na ovoj platformi bi se trebali moći dodavati novi bankomati, locirati, brisati, i slično.

Podaci koji treba da budu vizualizirani su:

- Lokacije bankomata
- Stanje broja novčanica u bankomatu
- Udjeli korištenih kartica na bankomatima
- Najkorištenije kartice iz perspektive korisnika
- Tabela prikaz svih transakcija
- Top transakcije
- Top bankomati
- Poređenje transakcija na dva izabrana bankomata
- Sve transakcije za odgovarajući bankomat

Svaka promjena na bankomatu treba da bude očitana odmah na aplikaciji i da u svakom trenutku svaki podatak bude vizualiziran i dostupan.

Detaljan opis svakog dijela sistema:

Bankomati su opisani jedinstvenim ID-om, sigurnosnim passwordom, lokacijom, količinom novca i vrstom kartica koje primaju.

Kroz web platformu se registrira postavljanje novog bankomata, postavlja se njegova lokacija i svi dodatni detalji.

Na web platformi će također biti omogućeno i dodavanje i izmjena i pregled stanja računa svih korisnika i bankomata.

Iščitavanje i prikaz vizualiziranih podataka ide kroz navigaciju u meniju na web platformi.

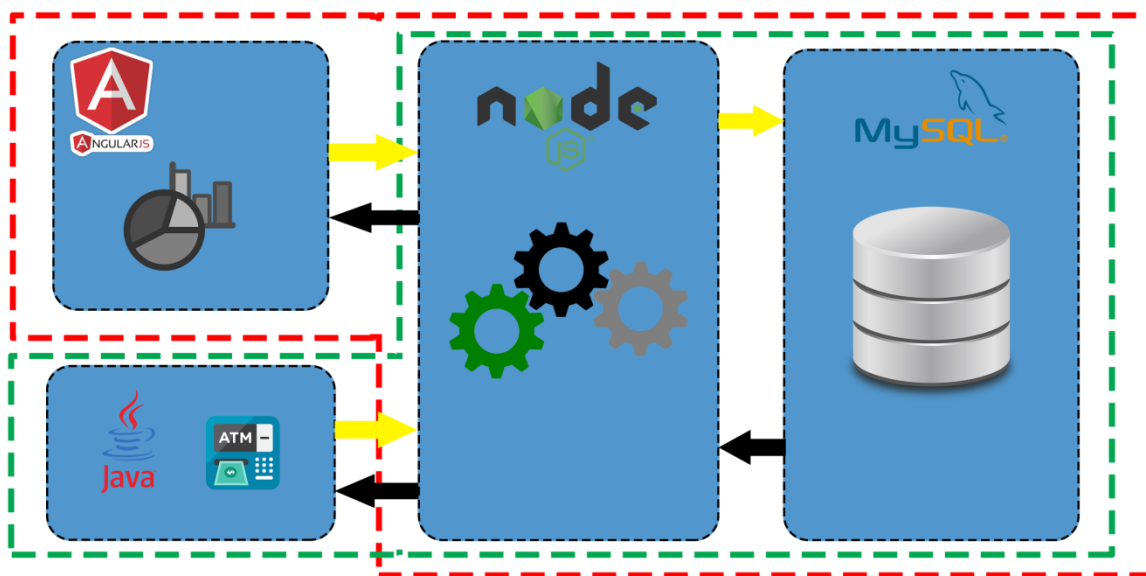
Web platforma će se pobrinuti za dodavanje, brisanje i izmjenu svih podataka potrebnih za realizaciju svega navedenog.

Objašnjenje načina realizacije kroz troslojnu arhitekturu

Objašnjena platforma je realizirana kroz troslojnu arhitekturu, gdje postoje dvije aplikacije, i to:

- Web aplikacija
- Mobilna aplikacija, koja će simulirati rad bankomata

Troslojna arhitektura spada u arhitekturne paterne, i to je način izgradnje aplikacija gdje su prezentacijska, aplikacijska i podatkovna logika fizički razdvojene. Na slici ispod je prikazan dijagram troslojne arhitekture ove platforme.



Slika 3-Troslojna arhitektura sistema

Vidimo da je web aplikacija biti podjeljena u tri fizički odvojena dijela, i to redom:

- MySQL baza podataka koja predstavlja podatkovni-backend sloj
- NodeJS realizacija REST servisa koja predstavlja aplikacijski sloj i middleware između baze podataka i prezentacijskog sloja
- AngularJS prezentacijski dio koji predstavlja frontend sloj

Mobilna aplikacija je također realizirana u troslojnoj arhitekturi, kao što se može vidjeti na slici, s tim da su prva dva sloja, ista i za mobilnu i za web aplikaciju. Dio prezentacijske logike, vezan za mobilnu aplikaciju će biti realiziran u Javi za Android mobilne aplikacije.

Crvena linija pokazuje granice web aplikacije, dok zelena predstavlja granice mobilne aplikacije. Primjetno je da i mobilna i web aplikacija koriste isti backend i middleware sloj. Na dijagramu, žute strelice pokazuju koji sloj od kojeg zahtjeva podatke, a crne strelice pokazuju koji sloj kojem podatke isporučuje.

Objašnjenje pojedinačnih dijelova i načina na koji rade

Ispod su data objašnjenja kako svaki od ovih slojeva rade i na koji način učestvuju u aplikaciji.

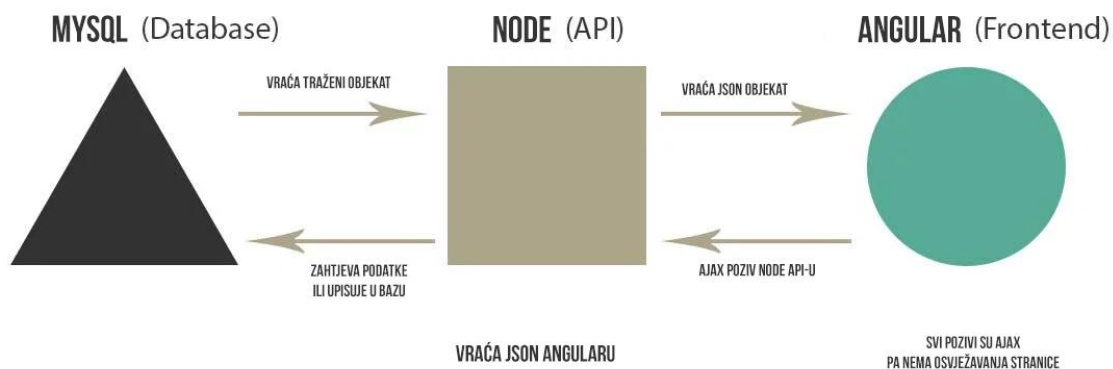
Backend layer - Database

Backend layer, bolje reći, sloj baza podataka je sloj u kojem se nalazi baza podataka koja čuva sve podatke vezane za platformu koja se realizira. Baza podataka koja se koristi u ovoj aplikaciji je MySQL baza podataka. Kako je MySQL baza podataka, relaciona baza podataka, podaci su organizovani u tabele, koje su povezane relacijama. Model baze podataka je prikazan u dijelu o UML dijagramima. Ovaj sloj komunicira sa middleware slojem i njemu isporučuje podatke na zahtjev.

Middleware - Servisi sa NodeJS

Middleware sloj ove aplikacije je realiziran kao isporučioc servisa. Ova logika je realizirana koristeći NodeJS, server-side JavaScript okruženje. U ovom sloju platforme, izgrađeni su REST servisi, o kojima je bilo govora ranije. Navedeni REST servisi omogućavaju da prezentacijski dio platforme dobija podatke od sloja u kom se nalazi baza podataka. Na poziv servisa iz prezentacijskog sloja, middleware komunicira sa bazom podataka, i šalje zahtjev za podacima, nakon toga, baza podataka vraća zahtjevane podatke, a middleware te podatke prosljeđuje u prezentacijski sloj koji vrši prikazivanje navedenih podataka.

Struktura navedene komunikacije je prikazana na slici ispod.



Slika 4 - Struktura komunikacije baze podataka i frontenda preko NodeJS-a

Presentation layer - Web platform sa AngularJS

Web aplikacija, odnosno njen prezentacijski je realiziran pomoću AngularJS-a, JavaScript frameworka i biblioteka za vizualizaciju podataka, kao što su NVD3 i GoogleCharts. Ovaj sloj dobija podatke iz baze podataka preko middlewarea. Pregled, izmjena, brisanje podataka u bazi se vrši na osnovu ovog sloja, koji preko POST i GET zahtjeva kontaktira middleware, a koji kontaktira bazu podataka. Nakon što

middleware dobije podatke iz baze, obrađuje ih i u JSON formatu vraća nazad u prezentacijski sloj gdje se AngularJS brine da isti budu prikazani na strani klijenta.

Presentation layer - Android aplikacija sa Java

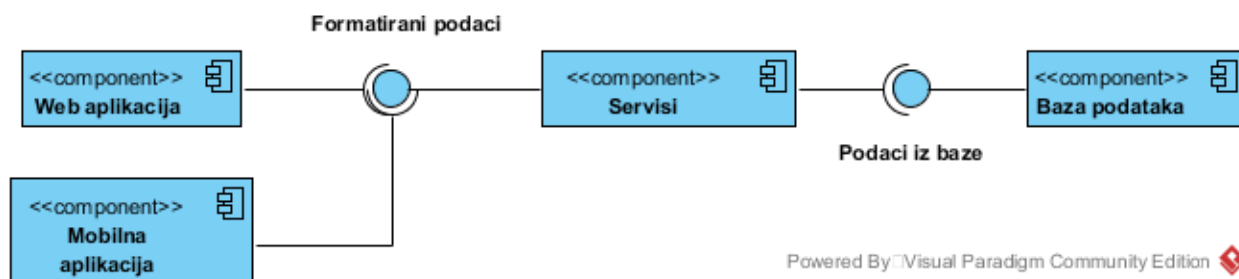
Što se tiče Android mobilne aplikacije, ona simulira rad bankomata. U tom smislu, ova aplikacija ima neke osnovne mogućnosti, a to je da se odabere količina novca koja se podiže računa, provjeri stanje računa i slično.

Prva dva sloja, odnosno sloj baze podataka i middleware su isti kao i za web aplikaciju. Dakle, svi potrebni podaci iz baze se dobivljaju preko middlewarea koji će komunicira sa bazom podataka, a koji nakon dobijenih podataka, iste prosljeđuje u android aplikaciji u JSON formatu. Android aplikacija poziva servise iz middlewarea i prosljeđuje potrebne parametre.

UML Dijagrami

U nastavku su prikazani UML (Unified Modeling Language) dijagrami koji bolje prikazuju način realizacije aplikacija, kao i potrebna objašnjenja uz iste.

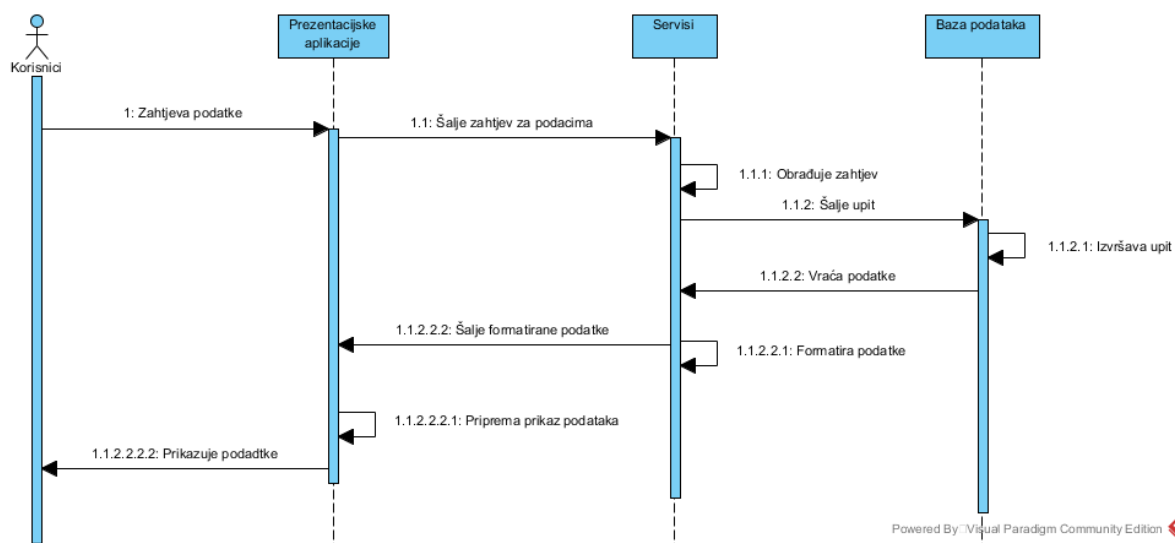
Component diagram



Slika 5 - Component dijagram

Na dijagramu iznad, može se vidjeti UML arhitektura kompletnog sistema, odnosno način razmjene podataka u sistemu. Middleware sloj aplikacije isporučuje servise, kojeg zahtjevaju web i mobilna aplikacija, a koji povlači podatke iz baze.

Sequence diagram

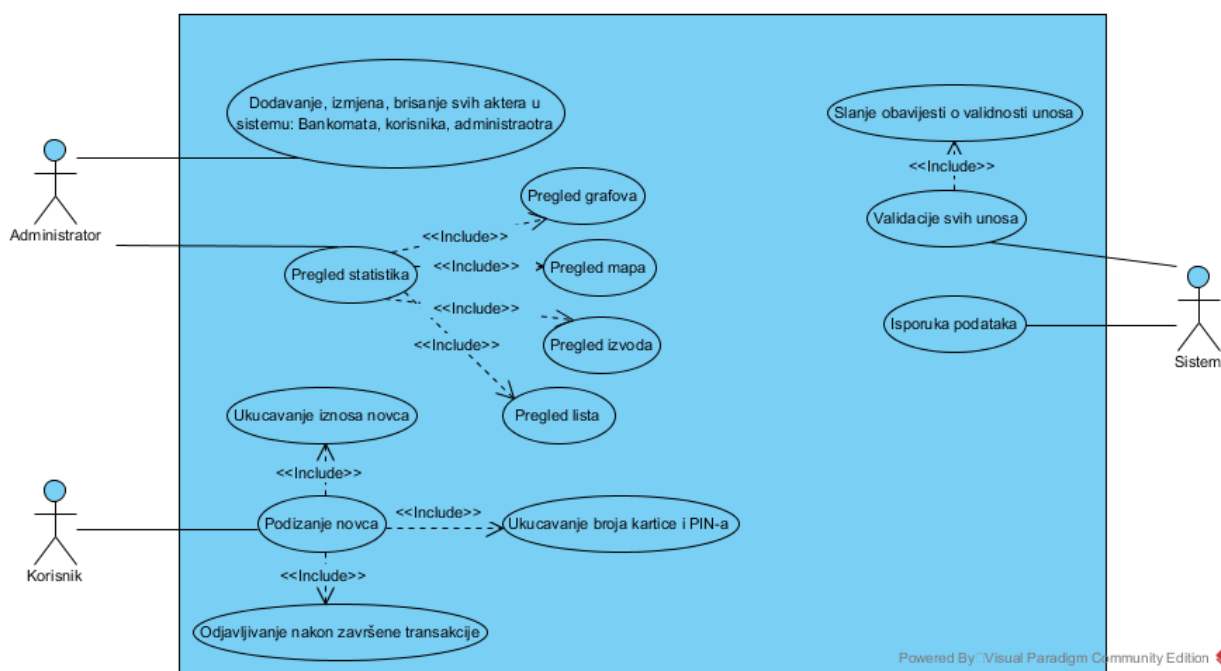


Slika 6 - Sequence dijagram

Na dijagramu iznad je pokazan tok zahtjeva i isporučivanja podataka od korisnika aplikacija, preko aplikacije i servisa, pa sve do baze podataka.

Use Case dijagram

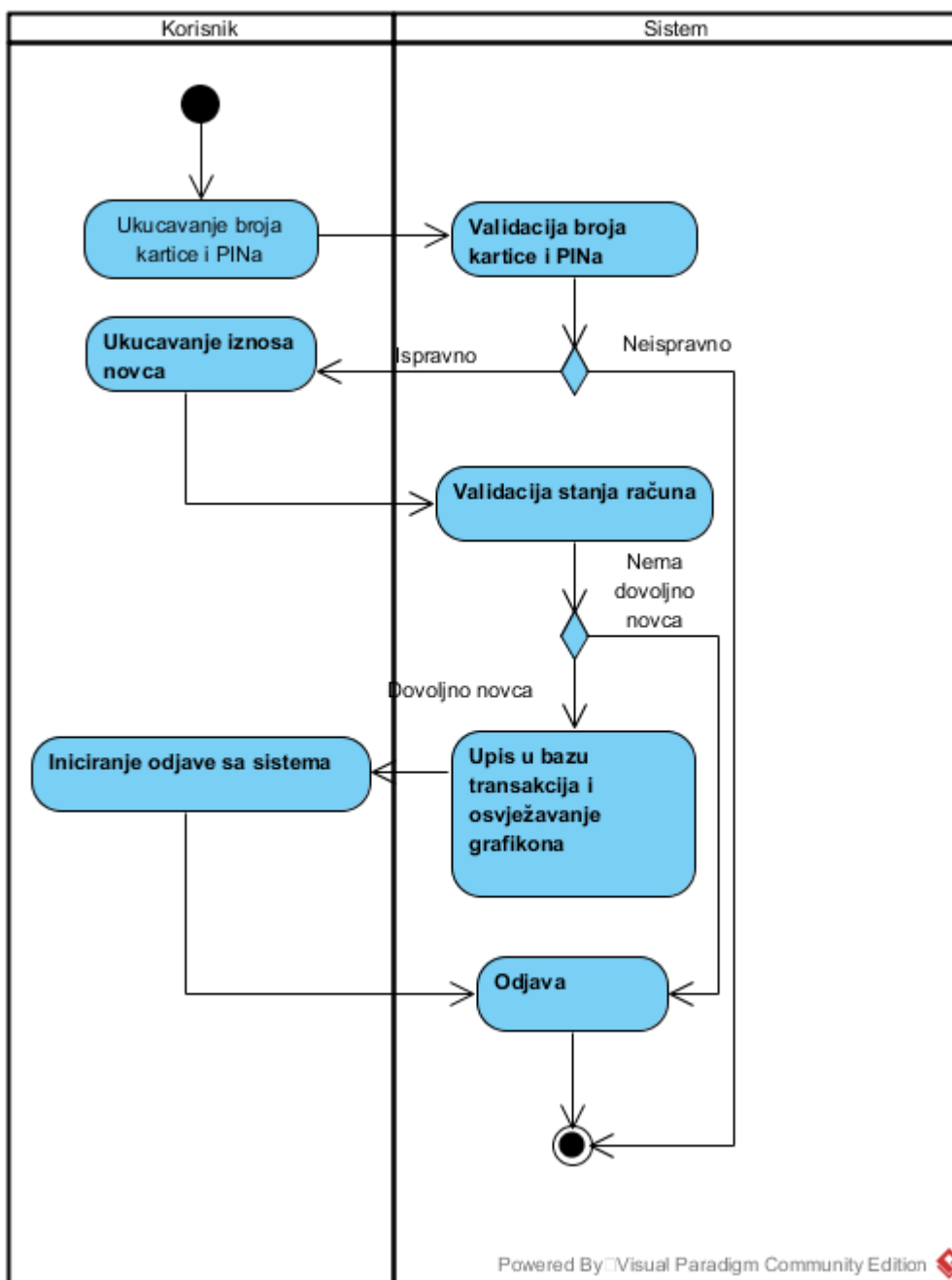
Use Case dijagramom su modelirane aktivnosti koje pojedini učesnici i akteri sistema obavljaju.



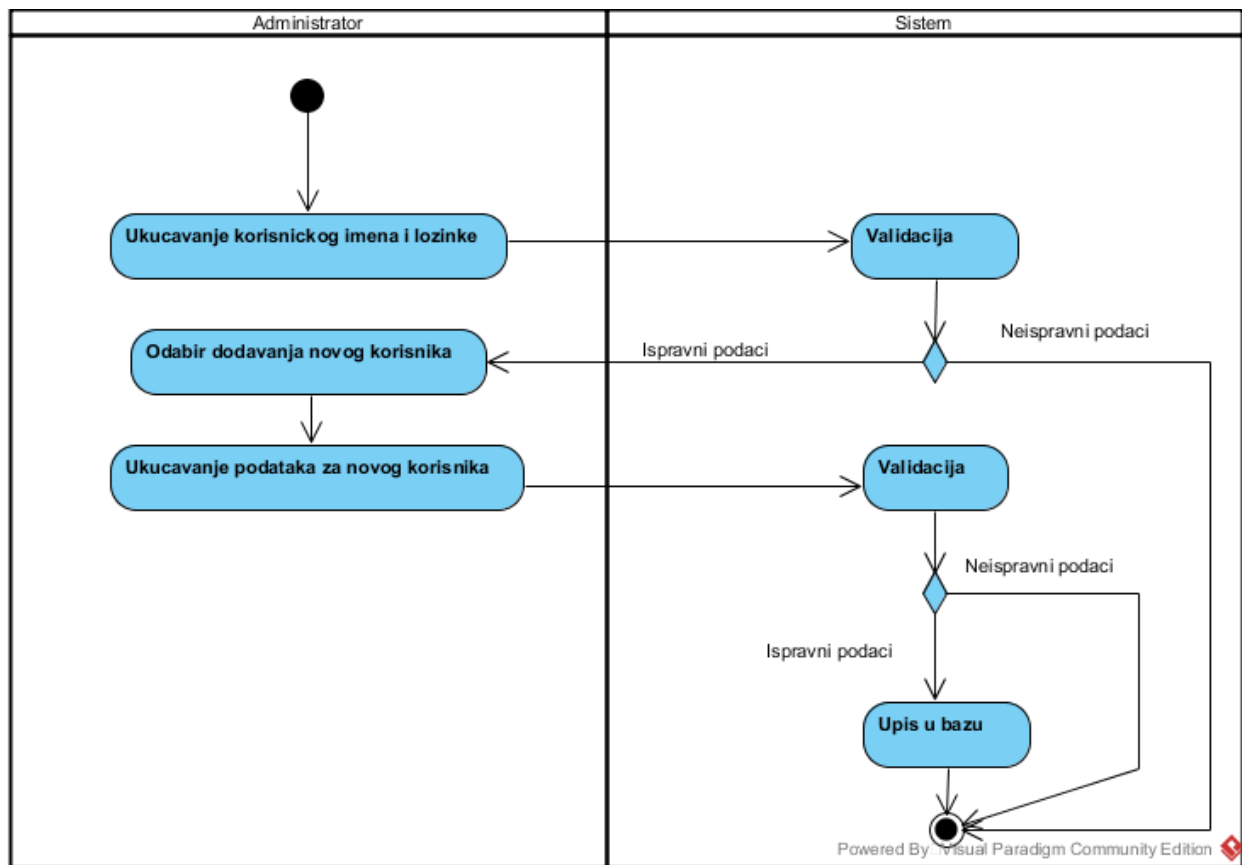
Slika 7 - UseCase dijagram

Activity dijagrami

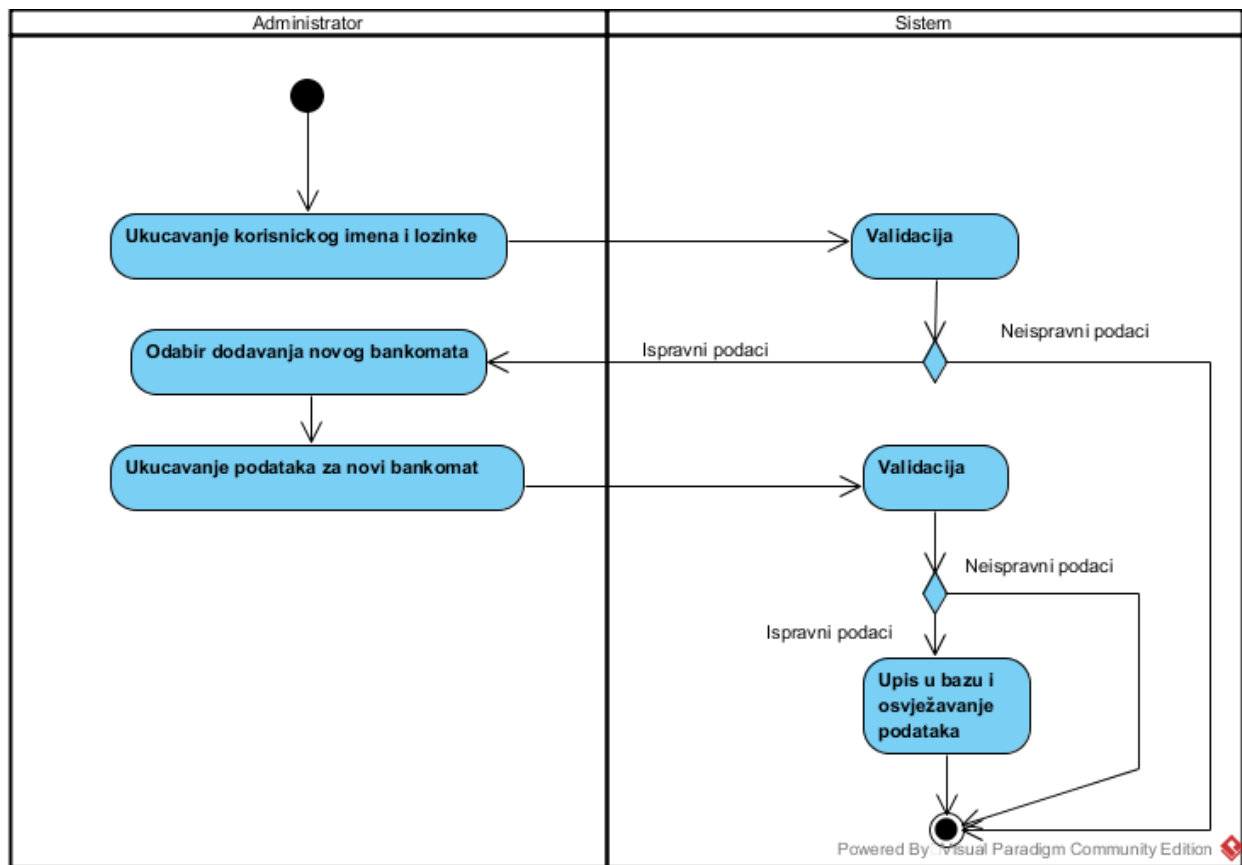
Activity dijagrami koji su prikazani ispod detaljno pokazuju na koji način i kojim redoslijedom se dešavaju aktivnosti za pojedine akcije unutar sistema.



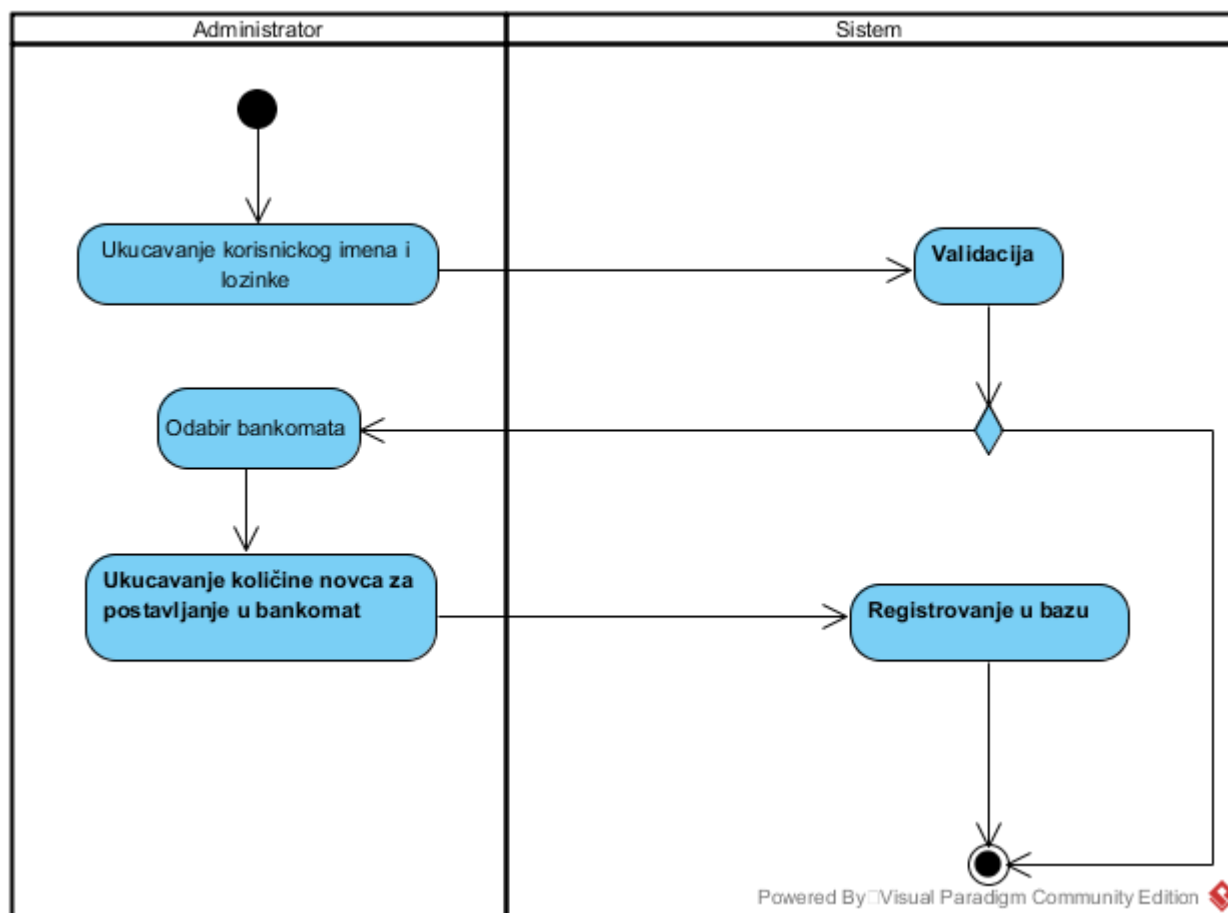
Slika 8- Podizanje novca sa bankomata



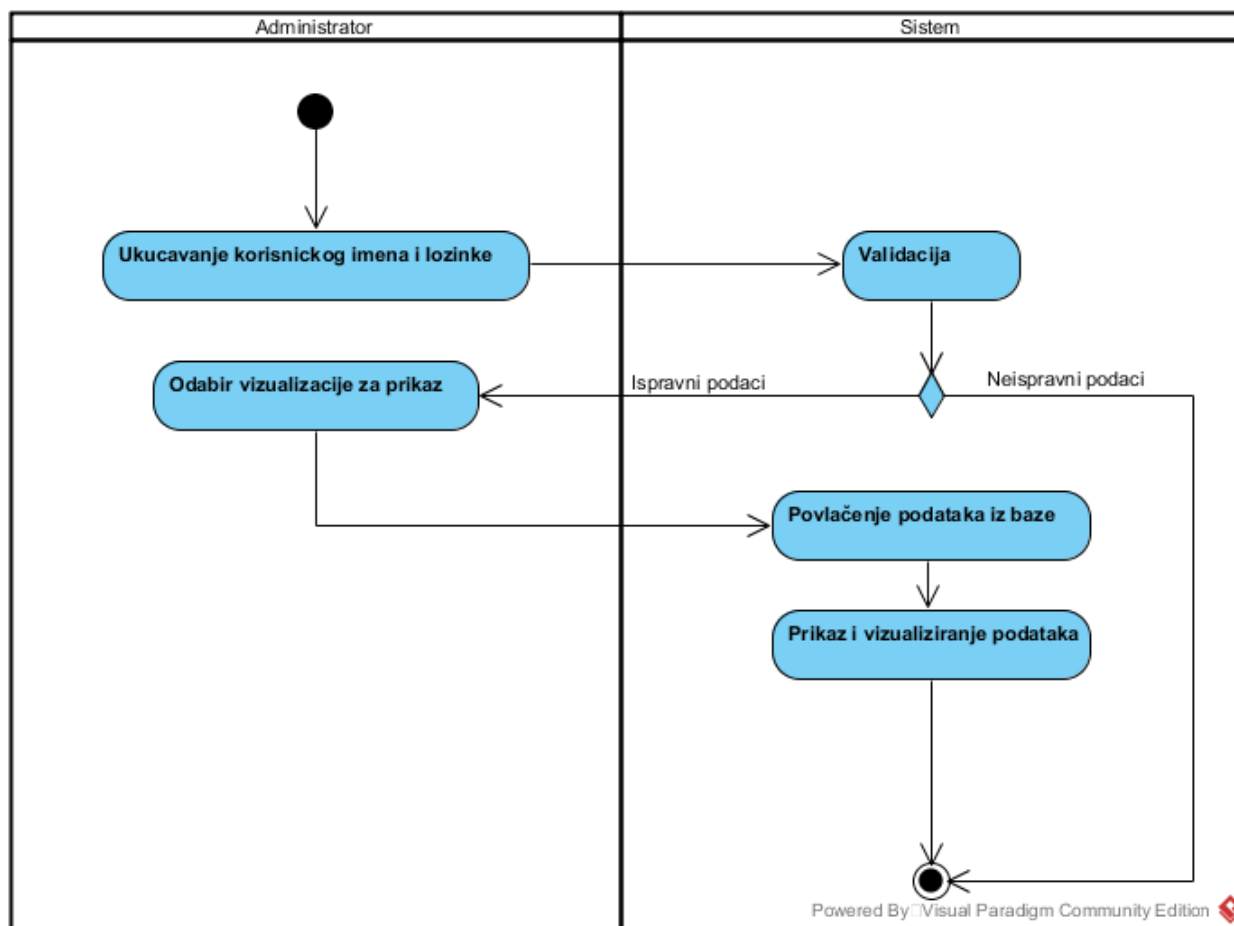
Slika 9 -Dodavanje korisnika



Slika 10- Dodavanje bankomata



Slika 11- Dopuna novca u bankomat



Slika 12 - Pregled statistika

ER dijagram baze podataka



Demonstracija aplikacije kroz screenshotove i *dijelove koda*

Demonstracija i prezentacija razvijanih aplikacija je podjeljena u dva dijela. Prvi dio je razvojni dio, gdje su prikazani tehnički detalji i struktura projekta, i drugi dio, gdje je prikazan korisnički aspekt aplikacija.

Tehnički aspekt realizacije aplikacija

Web aplikacija

Kao što je već rečeno u prethodnim poglavljima, web aplikacija koja se razvija je realizirana u sljedećim tehnologijama:

- MySQL - Baza podataka [12]
- NodeJS sa Express frameworkom - Backend sloj [13][14]
- AngularJS - Frontend sloj [15]
- Odgovarajuće biblioteke za vizualizaciju
 - NVD3 [16]
 - MapsGoogle [17][18]

Da bi se počela razvijati aplikacija sa gore navedenim stekom tehnologija, potrebno je da se napravi kvalitetna struktura foldera koje će aplikacija koristiti. Struktura foldera u projektu "VisualiseATM" je data u listi ispod.

- VisualiseATM
 - node_modules
 - public
 - css
 - fonts
 - js
 - app.js
 - controllers.js
 - lib
 - views
 - dashboard.html
 - index.html
 - login.html
 - app.js
 - package.json

Bitno je primjetiti par bitnih fajlova i foldera, koji su ključni za realizaciju aplikacije sa ovim stekom tehnologija.

Prvi fajl na koji treba da se obrati pažnja je “package.json” koji se nalazi u root folderu “VisualiseATM”. U ovom fajlu su sadržane sve informacije o NodeJS aplikaciji koja se razvija, od imena aplikacije, preko verzije i opisa, do dependenciesa (zavisnosti).

Prilikom otkucavanja komande “npm install” u komandnoj liniji, u folderu u kojem se nalazi “package.json”, npm package manager instalira sve potrebne zavisnosti i sve potrebne module za rad date NodeJS aplikacije i smješta ih u folder “node_modules”. Više o “package.json” fajlu se može pročitati na sljedećoj web stranici:

<https://docs.npmjs.com/files/package.json>

U tabeli ispod je prikazan sadržaj “package.json” fajla:

```
{
  "name": "visualiseatm",
  "version": "1.0.0",
  "description": "Aplikacija za završni rad bachelor studija elektrotehnike",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Ben",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.17.1",
    "ejs": "^2.5.6",
    "express": "^4.15.2",
    "mysql": "^2.13.0"
  }
}
```

Tabela 8 – package.json

Sljedeći fajl je fajl app.js koji se nalazi u root folderu “VisualiseATM”. App.js fajl u kojem se nalazi kod NodeJS aplikacije koja se razvija. Unutar ovog fajla se nalazi kod koji komunicira sa bazom podataka, kreira servise, obrađuje i validira podatke, kao i kod koji predaje rutiranje AngularJS aplikaciji.

Zbog obimnosti koda, u tabelama ispod će biti prikazani samo bitni isječki koda, koji su važni za ispravno funkcioniranje aplikacije.

Prvi isječak koda koji je prikazan ispod, prikazuje kod koji služi za kreiranje NodeJS aplikacije i povezivanje i korištenje ExpressJS frameworka za servisiranje.

```
var express = require('express');
var app = express();
```

```

app.engine('html', require('ejs').renderFile);
app.set('view engine', 'html');
//Dohvatanje post podataka
var bodyParser = require('body-parser')
app.use(bodyParser.json()); // podržavanje JSON-encoded bodies
app.use(bodyParser.urlencoded({ // podržavanje URL-encoded bodies
    extended: true
}));

```

Tabela 9 – Kreiranje NodeJs aplikacije i povezivanje i korištenje ExpressJS frameworka

Sljedeći isječak koda pokazuje na koji se način NodeJS aplikacija konektuje na bazu podataka.

```

//Konekcija na bazu
var mysql = require('mysql');
var connection = mysql.createConnection({
    host: 'localhost'
    , user: 'root'
    , password: ''
    , database: 'visualiseatm'
});

```

Tabela 10–Konekcija na bazu podataka

Kao što je u prethodnim poglavljima prezentirano, NodeJS aplikacija u ovom projektu je zadužena za ispostavljanje servisa i komunikaciju sa bazom podataka. Svi servisi aplikacije koja se razvija se nalaze pod linkom ".../api/", kao što je, recimo, servis, koji daje podatke o svim bankomatima, (link bi izgledao ovako: ".../api/dajSveBankomate"). U tabelama ispod će biti prikazana realizacija dva servisa, jedan koji radi sa GET metodom, i drugi koji radi sa POST metodom. U kodu oba servisa se može vidjeti i način na koji se komunicira sa bazom podataka, kao i način na koji se šalju odgovori na upit.

```

app.get('/api/dajStanjeRacuna/:idKartice', function(req, res){
    connection.query('SELECT racun.kolicinaNovca FROM racun WHERE
    racun.Kartica_idKartica = '+ req.params.idKartice, function(error,
    results, fields){
    if(error){
        connection.end();
    throw error;
    }
        res.send(results);
    });
});

```

Tabela 11 – Servis sa GET metodom


```

app.post('/api/promijeniStanjeRacuna',function(req, res){
    connection.query('UPDATE racun SET kolicinaNovca =
'+req.body.novaKolicinaNovca+' WHERE idRacun =
'+req.body.idRacuna,function(error, results, fields){
    if(error){
        connection.end();
        throw error;
    }
    res.send(results);
});
})

```

Tabela 12 – Servis sa POST metodom

Svi ostali servisi su realizirani na sličan način, sa manje ili više logike, ali u suštini po istom modelu, kao što je prikazan u prethodne dvije tabele.

Od bitnih stvari iz koda NodeJS aplikacije je važno da se prikaže sintaksa pokretanja servera i serviranja rutiranja² AngularJS aplikaciji, što se nalazi u tabeli ispod.

```

//Serviranje rutiranja ANGULARU
app.use(express.static(__dirname +'/public'));
//Pokretanje servra
app.listen(3000,function(){
    console.log('Magic happens on port 3000');
});

```

Tabela 13 – Servisiranje rutiranja Angular aplikaciji

NodeJS aplikacija se pokreće na portu 3000 i pokreće se sa naredbom “node app.js” u komandnoj liniji, u folderu gdje se nalazi “app.js” fajl. Što se tiče rutiranja, ono se prebacuje na AngularJS aplikaciju koja se nalazi u folderu “public”.

Nakon što je objašnjena sturktura NodeJS aplikacije, možemo preći na analizu “public” foldera.

Unutar tog foldera se nalazi 6 podfoldera koji sadrže bitne fajlove za funkcioniranje AngularJS aplikacije. U folderu “css” se nalaze potrebni .css fajlovi za izgled frontenda aplikacije. Folder fonts sadrži sve fontove koji se koriste u aplikaciji, dok folder lib sadrži biblioteke koje se koriste u realizaciji aplikacije. views folder sadrži sve .html fajlove koji se prikazuju na frontendu, a fajlovi “dashboard.html”, “index.html”, “login.html” su .html fajlovi koji su početni fajlovi i placeholderi za za sve ostale .html fajlove koji se injectaju u njih. Folderj “js” je folder u kom se nalaze fajlovi “app.js” i “controllers.js” i još neki javascript fajlovi potrebni za funkcionisanje aplikacije. “App.js” i “controllers.js” su zapravo dijelovi jedne aplikacije koji su samo radi sistematicacije i logičnosti razdvojeni u dvije odvojene cjeline.

Analiza koda AngularJS aplikacije počinje od analize postavljanja potrebnih zavisnosti za Angular aplikaciju, i to je prikazano na kodu ispod.

² Rutiranje je proces selektiranja putanje na kojoj se nalazi specifičan website.

```
//Inicijalizacija Angular aplikacije
var app = angular.module("VisualiseATM",[
  'ui.router'
  , 'controllers'
  , 'ngCookies'
  , 'ngMap' ]));
```

Tabela 14 – Inicijalizacija Angular aplikacije

Na kodu iznad se primjećuje da je registriran angular modul pod nazivom “VisualiseATM” i da su mu registrirane zavisnosti koje će koristiti. Te zavisnosti su, “ui.router”, koji služi za rutiranje AngularJS aplikacije; “controllers”, koji služi za kreiranje kontrolera AngularJS aplikacije; “ngCookies”, koji služi za manipulaciju sa cookieima u pretraživaču i “ngMap” koji je modul za Google mape koje se koriste u ovoj aplikaciji.

U tabeli ispod je prikazan način na koji aplikacija registruje rutu za određeni pogled, zatim kontroler i .html fajl koji odgovara određenom pogledu.

Radi količine ruta, prikazane su samo reprezentativni dijelovi koda. Sav ostali kod se realizira na isti način.

```
app.config(function($stateProvider, $urlRouterProvider){
  $urlRouterProvider.otherwise('/login');
  $stateProvider.state('dashboard',{
    url: '/dashboard'
  , templateUrl: 'dashboard.html'
  , controller: 'dashboardCtrl'
  }).state('dashboard.home',{
    url: '/home'
  , templateUrl: 'views/home.html'
  , controller: 'HomeController'
  }).state('login',{
    url: '/login'
  , templateUrl: 'login.html'
  , controller: 'loginCtrl'
  })

  .state('dashboard.dodavanjeBankomata',{
    url: '/dodavanjeBankomata'
  , templateUrl: 'views/dodavanjeBankomata.html'
  , controller: 'dodavanjeBankomataCtrl'
  })
});
```

Tabela 15 – Primjer registriranja ruta

Da bi se kreirali kontroleri za AngularJS aplikaciju, potrebno je registrovati novi angular modul. Linija koda koja to pokazuje je data u tabeli ispod.

```
var Controllers = angular.module("controllers",[]);
```

Tabela 16 – Registracija angular modula za kontrolere

Nakon što su kontroleri registrovani, može se pristupiti implementaciji istih. Kontroleri su dio AngularJS aplikacije u kojem se obavlja sva logika vezana za rad aplikacije. U ovom projektu, kontroleri se nalaze u fajlu „controllers.js“. U tabelama ispod su prikazani reprezentativni kontroleri, koji su realizirani unutar ove aplikacije.

```

Controllers.controller('dodavanjeBankomataCtrl',[
'$scope'
,'$http'
,'NgMap'
,'$location'
,function($scope, $http, NgMap, $location)
{
var lat =0;
var lng =0;
$scope.googleMapsUrl
="https://maps.googleapis.com/maps/api/js?key=AIzaSyAsoQGoDwX9eIFvUw8_Fkr
jjnBWCE8JUjI";
NgMap.getMap().then(function(map){
console.log(map.getCenter());
console.log('markers', map.markers);
console.log('shapes', map.shapes);
});

$scope.IspisiLokaciju =function(event){
lat = event.latLng.lat();
lng = event.latLng.lng();
console.log(lat);
console.log(lng);
}

$scope.tmpBankomat ={};
$scope.unesiBankomat =function(){
var MasterCard = document.getElementById("MasterCard").checked;
var Visa = document.getElementById("Visa").checked;
var Maestro = document.getElementById("Maestro").checked;
$http({
method:'POST'
, data:{
kolicinaNovca: $scope.tmpBankomat.kolicinaNovca
, lokacija: $scope.tmpBankomat.lokacija
, lozinka: $scope.tmpBankomat.lozinka
, lat: lat
, lng: lng
, MasterCard : MasterCard
, Visa : Visa
, Maestro : Maestro
}
, url:'/api/dodajBankomat'
}).then(function successCallback(response){
console.log(response);
$location.url('dashboard/lokacijeBankomata');
},function errorCallback(response){
console.log(response);
});
}
}]);

```

Tabela 17 – Primjer realizacije kontrolera

U tabeli iznad je prikazan kontroler koji omogućava dodavanje novih bankomata u sistem, i njihovo lociranje na mapi. Analizom koda kontrolera, može se uočiti način na koji se kontroler registrira. Kontroler se registruje svojim imenom, zavisnostima i pripadajućom funkcijom, u kojoj se obavlja kompletna logika kontrolera. Zavisnosti koje se koriste su „\$scope“, koja služi za upravljanje i bind varijabli na frontendu, „\$http“ koja služi za http pozive i komunikaciju sa servisima NodeJS aplikacije, „NgMap“, koja služi za manipulaciju sa GoogleMaps, „\$location“, koja služi za korištenje rutiranja unutar AngularJS aplikacije. Pomoću „\$scope“ u kontroleru je moguće registrirati varijable, nizove, objekte i funkcije unutar kontrolera. Način na koji se radi bind (povezivanje HTML atributa sa Angular objektima) funkcija i varijabli na frontendu je dat u isječcima kodova ispod.

```
<body ng-app="VisualiseATM">
<div ui-view></div>
</body>
```

Tabela 18 – Povezivanje HTML-a sa Angular aplikacijom

Da bi se html povezao sa AngularJS, potrebno je da se u „body“ tag doda „ng-app = 'VisualiseATM'“. „VisualiseATM“ je naziv AngularJS aplikacije koji je registrovan u „app.js“ folderu. Nakon što je aplikacija, na ovaj način, povezana sa html-om, moguće je unutar html tagova bindati varijable sa poljima unosa i pozivati funkcije na klik i slično. „ui-view“ označava da će se unutar tog diva injectati svi pogledi aplikacije.

```
<div class="input-field col s12">
<input id="lokacijaBankomata" type="text" class="validate" ng-
model="tmpBankomat.lokacija">

<label for="lokacijaBankomata">Lokacija</label>
</div>
```

Tabela 19 – Bind varijable iz Angular aplikacije sa HTML tagom

Iz gore navedenog koda primjetno je da se bind varijable radi sa „ng-model“, gdje se za „ng-model“ dodjeljuje naziv varijable koji se koristi u kontroleru sa „\$scope.datiNazivVarijable“.

```
<div class="row">
<div class="col s12"><a class="waves-effect btn-large right velikoDugme"
ng-click="unesiBankomat()">Dodaj</a></div>
</div>
```

Tabela 20 – Bind funkcije iz Angular aplikacije sa HTML tagom

Pozivanje funkcije unutar kontrolera se radi na sličan način, u kodu iznad je dat primjer poziva funkcije na klik dugmeta. Sintaksom „ng-click“, a zatim dodavanje naziva funkcije, specificira se koja funkcija će biti pozivana. Funkcija unutar kontrolera je definirana kao „\$scope.imeFunkcije = function(){}“.

Potrebno je obratiti pažnju na način na koji se šalju http pozivi. U tabeli kontrolera iznad se nalazi jedan http poziv. Unutar http poziva se specificira kojom metodom se poziva, zatim podaci koji se šalju u pozivu i url sa kojeg se poziva servis. Na osnovu uspješnosti poziva se poziva tijelo funkcija „successCallback“ ili „errorCallback“, u kojima mogu da se obrađuju dohvaćeni podaci.

Pri analizi ovog kontrolera, primjećuje se i kod koji manipulira sa mapom. Kao što je već napomenuto, korišten je modul "NgMap" koji je prilagodba Google mapa za rad sa AngularJS-om. Sav kod u kontroleru je klasičan JavaScript kod, kojeg za ovu priliku, nije potrebno objašnjavati detaljno. Ostaje da se prikaže, na koji način se mape registruju u HTML-u i koriste i iscrtavaju, pa je za to, dat kod u tabeli ispod.

```
<ng-map center="44.409477, 18.524855" zoom="4" pan-control="false" zoom-  
control="false" scale-control="true">  
  <marker  
    centered="true"  
    position="44.409477, 18.524855"  
    draggable="true"  
    on-dragend="IspisiLokaciju()">  
  </marker>  
</ng-map>
```

Tabela 21 –Povezivanje ng-map modula iz Angular aplikacije sa HTML-om

U nastavku i tabeli ispod (tabela 22) je prikazan još jedan kontroler u kojem je prikazana logika korištenja biblioteke za vizualizaciju, NVD3.

```

Controllers.controller('topBankomatiPoBrojuTransakcijaCtrl',[
'$scope'
,'$http'
,function($scope, $http)
{
    $http({
        method:'GET'
    , url:'api/dajTopBankomatePoBrojuTransakcija'
    }).then(function successCallback(response){
        $scope.topBankomati = response.data;
    var podaciZaBarChart = [];
        podaciZaBarChart.key = "Podaci";
        podaciZaBarChart.values = $scope.topBankomati;
        console.log(podaciZaBarChart);
    var podaciNovi = [];
        podaciNovi.push(podaciZaBarChart);
        nv.addGraph(function(){
    var chart = nv.models.discreteBarChart().x(function(d){
    return d.label
    }).y(function(d){
    return d.value
    }).staggerLabels(true).tooltips(false).showValues(true)
        d3.select('#barChartTopBankomataPoTrasnakcijama
    svg').datum(podaciNovi).transition().duration(500).call(chart);
        nv.utils.windowResize(chart.update);
    return chart;
    });
    },function errorCallback(response){
        console.log(response);
    });
}
]);

```

Tabela 22 – Primjer kontrolera u kom se koristi biblioteka za vizualizaciju (NVD3)

Nakon što se podaci dohvate putem http poziva, postoji mogućnost da ih je potrebno obraditi i pripremiti za vizualizaciju, jer većina biblioteka za vizualizaciju zahtjeva da set podataka bude prilagođen, pa tako i NVD3. Nakon što se podaci pripreme, poziva se funkcija “adGraph()” u čijem se tijelu predaju podaci koji treba da se vizualiziraju.

Ovime je završena analiza koda i strukture projekta web aplikacije.

Kao što je već napomenuto, mobilna aplikacija u ovom radu služi kao simulator bankomata i realizirana je na Android[19] platformi. Pošto su svi servisi realizirani u NodeJS aplikaciji, mobilna aplikacija samo koristi te servise, pa će u nastavku biti prikazana dva isječka koda koji pokazuju način na koji se ti servisi pozivaju u mobilnoj aplikaciji.

U tabeli koja slijedi je prikazan kod klase koja služi za asinhrono pozivanje servisa.

```
public class CallAPI extends AsyncTask <String, String, String>{

    public CallAPI(){

    }

    private onCallApiDone pozivatelj;
    public CallAPI(onCallApiDone p){pozivatelj = p;};
    private digniNovac pozivateljDizeNovac;
    public CallAPI(digniNovac p){pozivateljDizeNovac = p;};
    private stanjeBankomata pozivateljStanjaBankomata;
    public CallAPI(stanjeBankomata p){pozivateljStanjaBankomata = p;};
    @Override
    protected String doInBackground(String... params){
        String urlString = params[0];

        String resultToDisplay = "";

        InputStream in = null;
        try{

            URL url = new URL(urlString);
            System.out.println(url.toString());
            HttpURLConnection urlConnection = (HttpURLConnection)
            url.openConnection();

                in = new BufferedInputStream(urlConnection.getInputStream());

        }catch(Exception e){

            System.out.println(e.getMessage());

            return e.getMessage();

        }

        try{
            resultToDisplay = IOUtils.toString(in, "UTF-8");
            System.out.println(resultToDisplay);
        }
        catch(IOException e){
            e.printStackTrace();
        }
    }
}
```



```

return resultToDisplay;
}

@Override
protected void onPreExecute(){
    super.onPreExecute();
}

@Override
protected void onPostExecute(String s){
    super.onPostExecute(s);
    if(pozivatelj != null){
        pozivatelj.onDone(s);
    }
    if(pozivateljDizeNovac != null){
        pozivateljDizeNovac.zavrsonoDizanje(s);
    }
    if(pozivateljStanjaBankomata != null){
        pozivateljStanjaBankomata.zavrsonoDobavljanjeStanjaBankomata(s);
    }
}

public interface onCallApiDone{
    public void onDone(String rez);
}

public interface digniNovac{
    public void zavrsonoDizanje(String rez);
}

public interface stanjeBankomata{
    public void zavrsonoDobavljanjeStanjaBankomata(String rez);
}
}

```

Tabela 23 – Klasa za asinhrono pozivanje servisa

Klasa "CallAPI" je nasljeđena iz klase "AsyncTask" i implementira potrebne metode za asinhrono pozive. Ti metodi su "doInBackground()", "onPreExecute" i "onPostExecute". Također, u klasi su implementirani i interfejsi, koji su potrebni da se implementiraju u klasama iz kojih se šalju pozivi.

Način na koji se pozivaju servisi je dat u tabeli ispod.

```

new
CallAPI((CallAPI.onCallApiDone)Pocetna.this).execute("http://10.0.2.2:3000/api/dajSveNaziveBankomata");

```

Tabela 24 – Primjer koda za pozivanje servisa

Nakon što se poziv obavi, poziva se metoda "onDone" koja je implementirana prilikom implementacije interfejsa. Kod implementacije metode "onDone" (za poziv iznad) je dat u tabeli ispod.

```

@Override
public void onDone(String rez) {
try
    {
        Spinner dropdown =
        (Spinner)findViewById(R.id.listaBankomata);
        ArrayList<String> nizBankomata = new ArrayList<>();
        JSONArray nizNazivaBankomata = new JSONArray(rez);
        for(int i = 0; i < nizNazivaBankomata.length(); i++)
        {
            JSONObject pojedinačniBankomat = (JSONObject)
            nizNazivaBankomata.get(i);

            nizBankomata.add(pojedinačniBankomat.getString("identifikator"));

            listaIdevaBankomata.add(pojedinačniBankomat.getString("idBankomat"));

        }
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_spinner_dropdown_item, nizBankomata);
        dropdown.setAdapter(adapter);
    }catch (JSONException e) {
        e.printStackTrace();
    }
}

```

Tabela 25 – Implementacija onDone metode

U kodu iznad se, nakon što su dohvaćeni, podaci, izlistavaju u dropdown listi. Svi servisi se pozivaju na isti način, samo u zavisnosti od potrebne logike se implementiraju različiti interfejsi.

Ovime je završena analiza koda i strukture projekta web aplikacije.

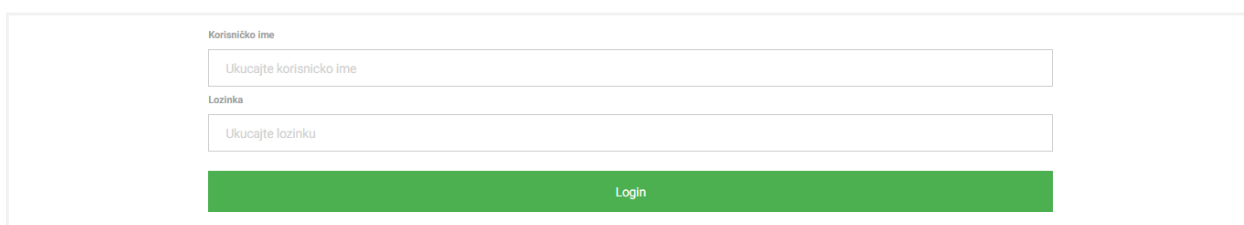
U tekstu iznad su prikazani reprezentativni dijelovi koda za aplikacije koja su razvijane, a kompletan kod se može pronaći na github repozitoriju, kao i u prilogu ovog rada.

Korisnički aspekt aplikacija

WEB aplikacija

Na samom pokretanju aplikacije, otvara se login panel, u kojem se unosi korisničko ime i lozinka korisnika. Korisnici koji su registrovani kao administratori imaju pristup i dozvolu da se loguju u sistem.

Visualise ATM Login

The screenshot shows a login interface for 'Visualise ATM'. It features two input fields: 'Korisničko ime' (Username) with the placeholder text 'Ukucajte korisničko ime' and 'Lozinka' (Password) with the placeholder text 'Ukucajte lozinku'. Below these fields is a prominent green button labeled 'Login'.

Slika 14 - Login

Nakon što se prijave, korisnicima se prikazuje dashboard panel.

The screenshot displays the 'Visualise ATM' dashboard. On the left is a red sidebar menu with the title 'Visualise ATM' and the user role 'Prijavljeni kao admin'. The menu includes options like 'Početna', 'Dodavanje bankomata', 'Dodavanje korisnika', 'Kreiranje računa', 'Punjenje bankomata', 'Izmjena korisnika', a 'Pregledi' (Views) section with sub-items like 'Pregled bankomata', 'Stanje novčanica', 'Udjeli korištenih kartica', 'Najkorištenije kartice', 'Pregled transakcija', and 'Top 5 transakcija'. The main content area has the title 'Visualise ATM' and a subtitle 'Završni rad prvog ciklusa studija na Elektrotehničkom fakultetu u Sarajevu na temu: Razvoj aplikacije za prikupljanje i vizualizaciju podataka sa bankomata koristeći Angular JS i Google Mape'. It also displays student and professor information: 'Student: Ramić Benjamin' and 'Profesor: Dr. Emir Buza, dipl.ing.el.'.

Slika 15- Početna stranica

Sa lijeve strane se nalazi meni za navigaciju kroz aplikaciju, dok se na desnoj strani nalazi prostor za funkcionalnosti.

Dodavanje bankomata se obavlja kroz sučelje koje je prikazano na sljedećoj slici.

Visualise ATM

Prijavljeni kao admin

Početna

Dodavanje bankomata

Dodavanje korisnika

Kreiranje računa

Punjenje bankomata

Izmjena korisnika

Pregledi

Pregled bankomata

Stanje novčanica

Udjeli korištenih kartica

Najkorištenije kartice

Pregled transakcija

Top 5 transakcija

Dodavanje bankomata

Lokacija

Količina novca

Kartice koje podržava:

☐ MasterCard
☐ Visa
☐ Maestro

Lozinka bankomata

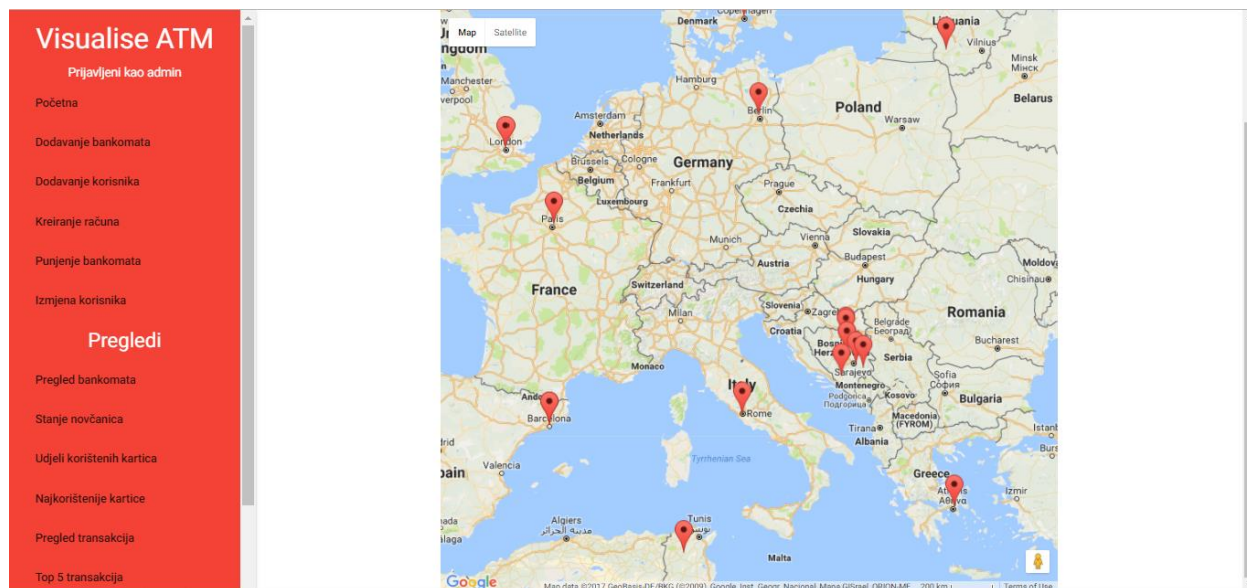
Slika 16- Dodavanje bankomata

Kada popunimo tražena polja, bankomat je potrebno locirati na mapi. Mapa koja se iscrta, zajedno sa jednim pomičnim pin markerom izgleda kao na slici ispod.



Slika 17 -Lociranje bankomata na mapi

Pin koji je prikazan na mapi je pomičan i pomicanjem mišem se locira bankomat na mapi. Za pregled svih bankomata na mapi se odabire opcija u meniju i sučelje za to izgleda kao na slici.



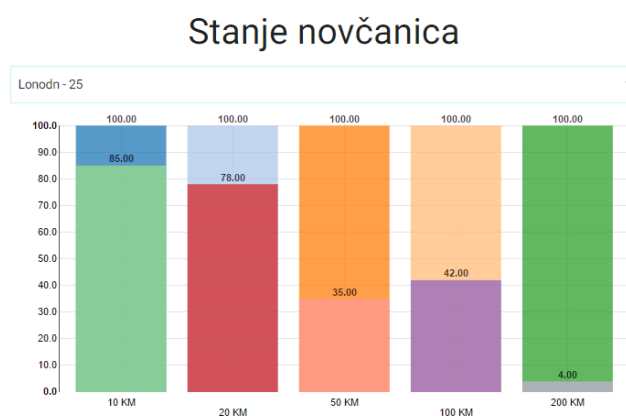
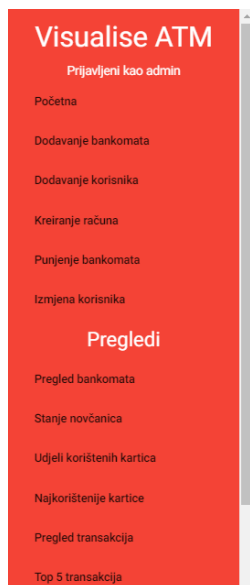
Slika 18-Pregled bankomata na mapi

Svi bankomati su označeni pinovima na mapi.

Od CRUD panela, reprezentativan je panel kreiranja računa. On je prikazan na slici ispod.

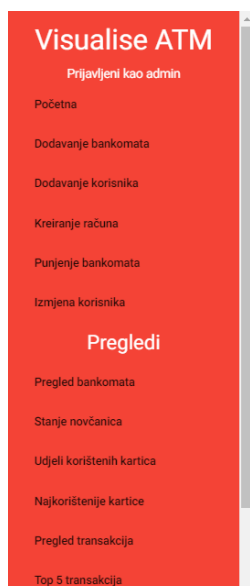
Slika 19-Kreiranje računa

Ono što je zapravo od interesa i što je glavna nit i tema ovog rada su vizualizacije unutar ove WEB aplikacije. U meniju se nalazi sekcija za preglede, u kojoj se odabire vizualizacija od interesa. U screenshotovima ispod će biti prikazane sve vizualizacije koje ova web aplikacija nudi.

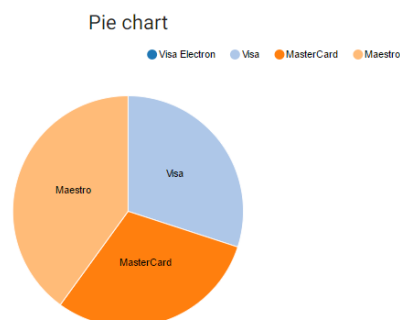


Na dijagramu su prikazani kapaciteti kasete za novčanice i trenutno stanje u kasetama.

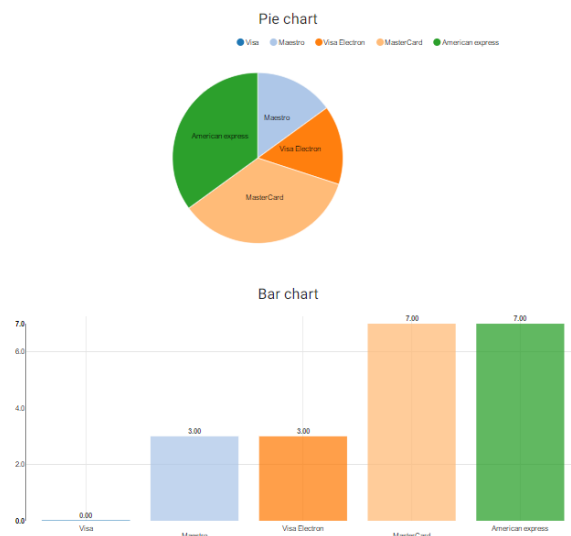
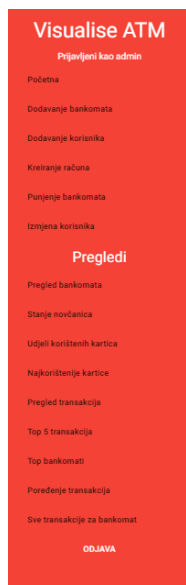
Slika 20 - Stanje broja novčanica u bankomatu



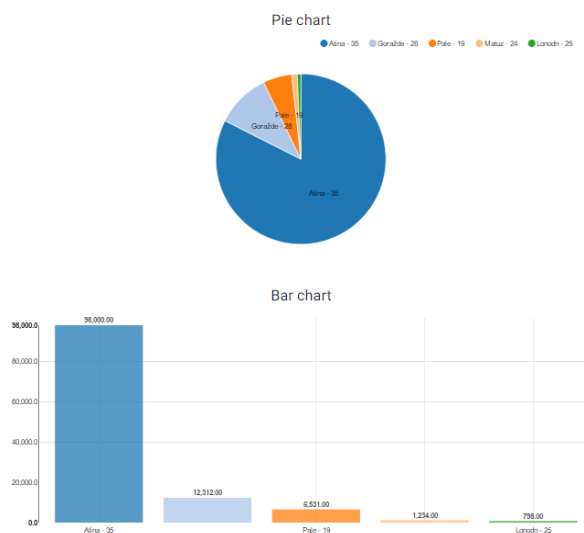
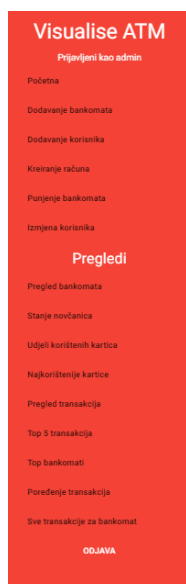
Kartice po broju bankomata koji ih primaju:



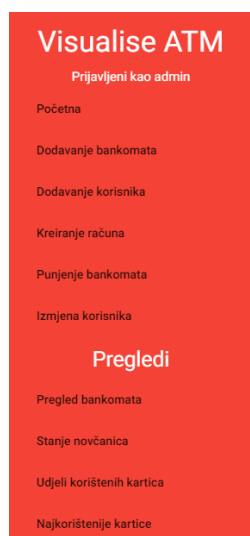
Slika 21- Udjeli korištenih kartica na bankomatima



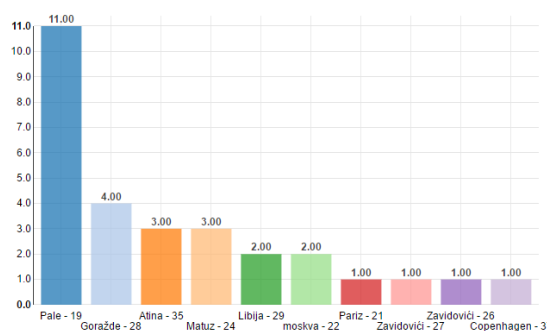
Slika 22 - Najkorištenije kartice iz perspektive korisnika



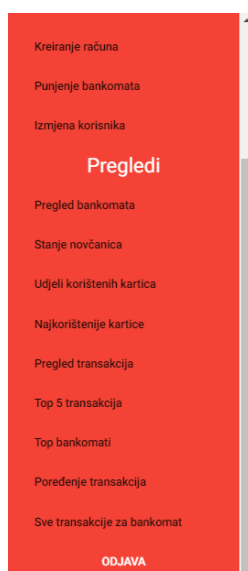
Slika 23 - Top transakcije



Top bankomati po broju transakcija

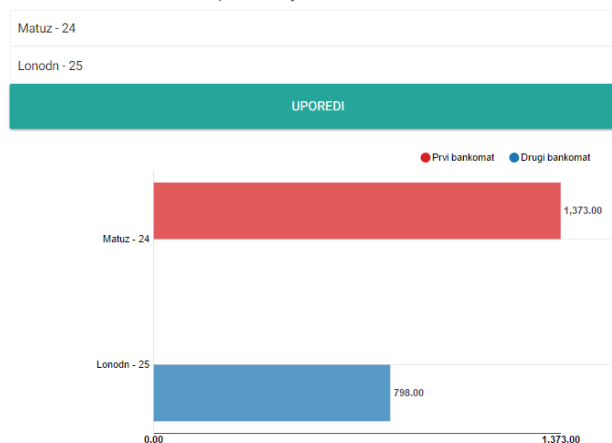


Slika 24-Top bankomati

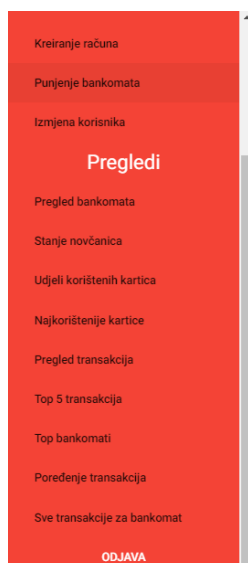


Uporedi dva bankomata

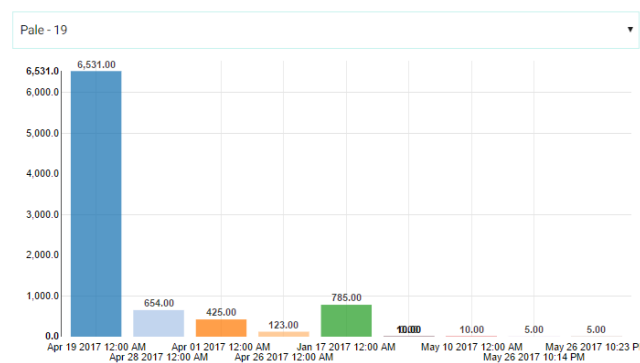
Odaberi bankomate za poređenje



Slika 25- Poređenje transakcija na dva odabrana bankomata



Transakcije po bankomatu:



Slika 26- Sve transakcije za odabrani bankomat

Bitno je napomenuti da se sve vizualizacije dinamički mijenjaju u skladu sa promjenama u bazi podataka. Primjetno je da su za većinu vizualizacija korišteni PieChart ili BarChart oblici vizualizacije. To je zato jer za postavljeni problem su ovi oblici grafova bili najpogodniji. Interesantno je izdvojiti graf za poređenje dva bankomata. Praktično, to je horizontalni BarChart koji prikazuje samo dvije vrijednosti koje se dinamički mijenjaju u zavisnosti od odabranih bankomata. Zbog modularnosti koda, moguće je dodati i poređenje više od dva bankomata.

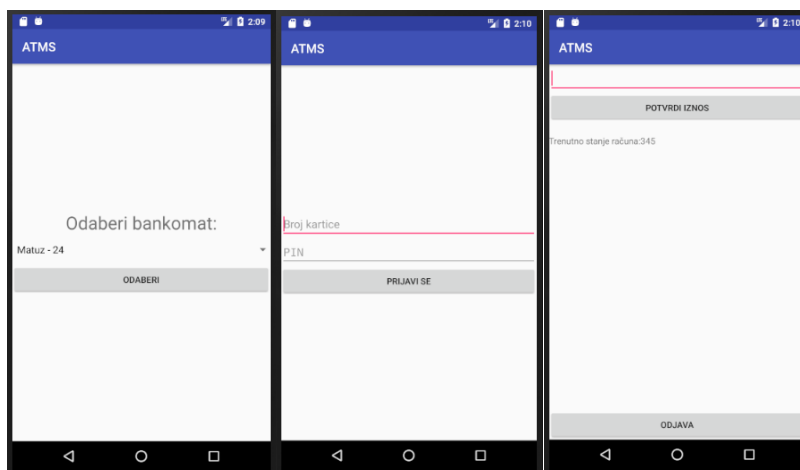
Ovim je završen dio prezentacije WEB aplikacije sa korisničkog aspekta. Aplikacija se može pronaći na github repozitoriju projekta, pokrenuti i isprobati sve njene funkcionalnosti.

Mobilna aplikacija

Mobilna aplikacija služi kao simulator stvarnih bankomata i preko nje se u bazu podataka šalju podaci o transakcijama, računima i bankomatima.

Aplikacija je razdvojena na tri ekrana, početni ekran gdje se bira sa kojeg bankomata se uzima novac, drugi ekran, gdje se ukucava broj kartice i PIN, i treći ekran koji nudi pregled stanja računa i sučelje za odabiranje količine novca za uzimanje sa bankomata. U slučaju da se odabere veća količina novca od one koja se nalazi na računu ili trenutno u bankomatu, ispisuje se greška transakcije.

Ispod su prikazani screenshotovi tri ekrana.



Slika 27-Ekrani mobilne aplikacije

Ovime je završena analiza aplikacija koje su kreirane unutar ovog rada. U sljedećem, posljednjem poglavlju je dat zaključak i osvrt na sve urađeno, kao i smjernica za eventualno buduće unaprjeđenje ovih aplikacija.

Zaključak

Kako bi se što bolje rezimiralo sve gore napisano, zaključak je podjeljen na tri dijela, i to dio vezan za vizualizaciju, dio vezan za implementaciju frontend sloja, backend sloja i baze podataka i dio vezan za eventualna unaprjeđenja ovog rješenja.

Kako je vizualizacija bila glavna tema ovog rada i nit oko koje se kompletan rad gradio, bitno je istaknuti uočene pojave prilikom procesa vizualizacije. Najbitnije i najvažnije za istaknuti je bitnost dva koraka u procesu vizualizacije, prikupljanje i priprema podataka. Do ovog zaključka se došlo pri procesu pripreme podataka za vizualiziranje tokom korištenja biblioteka. Naime, biblioteke koje su korištene, a koje su jako slične sa svim dostupnim, besplatnim, gore navedenim bibliotekama za vizualizaciju su prilično striktno u smislu oblika podataka koji treba da se vizualizira. Razvojni inženjeri, ako koriste ove biblioteke, imaju prilično mali spektar djelovanja, u smislu manipulacije oblika podataka. Naravno, korištenjem open source biblioteka, postoji mogućnost da razvojni inženjeri pokušaju prilagoditi ove biblioteke za svoj oblik i set podataka, međutim to nerijetko zna biti težak i mukotrpan posao, pa se više isplati kreirati svoje biblioteke za vizualizaciju. Ono što je bitno istaknuti, jeste da ove biblioteke imaju odlično implementirane osnovne grafove, kao što su bar chart, pie chart, line chart i slično, pa za neke manje aplikacije i manje – standardizovane podatke i vizualizacije, korištenje ovih biblioteka je i više nego opravdano i preporučljivo. U slučaju da se razvojni inženjeri odluče za implementaciju sopstvenih biblioteka za vizualizaciju, bitno je da prije toga naprave dobar plan i analizu podataka koji će se vizualizirati, i u kom obliku će podaci biti korišteni i vizualizirani, jer je implementacija onda znatno olakšana. Što se tiče biblioteka koje su komercijalno dostupne, one imaju znatno veću podršku, koja može da se iskoristi za neke popravke, prilagođavanja, i slično.

S druge strane, ako se gleda na implementacijski dio, konkretno za ovu aplikaciju, postoji veliki prostor za unaprjeđenje i promjene. NodeJS, sa Express frameworkom je prilično brz i intuitivan za korištenje, međutim postavlja se pitanje koliko je skalabilan i stabilan da podržava veće aplikacije, a naročito distribuirane aplikacije. Također, rad sa dijeljenom bazom podataka, znatno može da optereti i bazu i sloj koji radi sa njom. Ono gdje se ovdje može realizirati unaprjeđenje jeste da se koristi neki malo robustniji framework, kao što je .NET core ili Spring. Ova dva frameworka rade nad objektno orijentisanim programskim jezicima, pa i to daje dodatnu prednost prilikom olakšavanja pripreme i manipulacije podacima za vizualizaciju. Što se tiče dijeljene baze podataka, ona u ovom slučaju je sasvim korektan izbor, ali realizacija „pomoćnih“ baza podataka koje će čuvati neke podatke lokalno i sinhronizirati sa centralnom bazom podataka je mnogo pouzdaniji oblik čuvanja podataka. Ovo bi se moglo realizirati sa SQLite bazom podataka, koja dolazi kao defaultna baza podataka sa Android operativnim sistemom. U ovu bazu bi se mogli smiještati podaci o transakcijama lokalno na bankomatima, a zatim sinhronizirati sa centralnom bazom podataka. Ova sinhronizacija bi morala biti izuzetno brza zbog konzistentnosti podataka u centralnoj bazi podataka. Također, bitno je naglastiti da je implementacija backup sistem i sistem repliciranja centralizovane baze podataka veoma značajna, kako bi podaci bili očuvani u slučaju kraha baze podataka. Na frontend sloju, AngularJS je odlična opcija, međutim, kako je Angular 2 unaprjeđenje AngularJS-a, svakako da postoji prostor za prelazak na Angular 2. Korištene biblioteke su ispunile svoju misiju, međutim, za neke specifičnije analize i vizualizacije, bolja opcija bi bila kreiranje vlastitih rješenja za vizualizaciju.

U tekstu iznad su napisani prijedlozi za eventualna unaprjeđenja sa tehničkog aspekta. S druge strane, s korisničke tačke gledišta, mogu se razmotriti unaprijeđenja u smislu većeg broja vizualizacija i korisnijih vizualizacija. Također, postoji prostor za kreiranje platforme za vizualizacije koje su od interesa vlasnicima računa u nekoj banci. Naime vlasnici računa mogli pristupati pomoću jedinstvenog korisničkog imena i lozinke koja bi se mogla isporučivati na osnovu tokena, ili sličnih mehanizama za mobilno i web bankarstvo.

Reference

Literatura

- [1] - **Michael Friendly**, Introduction in A Brief History of Data Visualization, 4700 Keele Street, Toronto, ON, Canada M3J 1P3, 2006
- [2] - https://en.wikipedia.org/wiki/Representational_state_transfer
- [3] - <http://microservices.io/patterns/data/shared-database.html>
- [4] - <http://www.geekboots.tech/2017/04/what-is-rest-api-how-does-it-work.html>
- [5] - <http://optimalbi.com/blog/2016/07/21/whats-the-best-restful-web-api-framework-part-2/>
- [6] - <http://data-informed.com/top-5-business-benefits-using-data-visualization/>
- [7] - <https://www.sisense.com/blog/10-useful-ways-visualize-data-examples/>
- [8] - **Maila Hardin, Daniel Hom, Ross Perez, & Lori Williams**, Which chart or graph is right for you?
- [9] - **Ben Fry**, The Seven Stages of Visualizing Data in Visualizing Data, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2007
- [10] - <https://blog.hubspot.com/marketing/data-visualization-choosing-chart>
- [11] - <https://thenextweb.com/dd/2015/06/12/20-best-javascript-chart-libraries/>
- [12] - <https://www.mysql.com/>
- [13] - <https://nodejs.org/en/>
- [14] - <https://expressjs.com/>
- [15] - <https://angularjs.org/>
- [16] - <http://nvd3.org/>
- [17] - <https://ngmap.github.io/>
- [18] - <https://developers.google.com/maps/>
- [19] - <https://developer.android.com/index.html>

Slike

Slika 1 - Šema dijeljene baze podataka	5
Slika 2 - Pie i Bar chart gore, tabelarno, prikazanih podataka	11
Slika 3 - Troslojna arhitektura sistema	20
Slika 4 - Struktura komunikacije baze podataka i frontenda preko NodeJS-a	21
Slika 5 - Component dijagram	23
Slika 6 - Sequence dijagram	23
Slika 7 - UseCase dijagram	24
Slika 8 - Podizanje novca sa bankomata	25
Slika 9 - Dodavanje korisnika	26
Slika 10 - Dodavanje bankomata	27
Slika 11 - Dopuna novca u bankomat	28
Slika 12 - Pregled statistika	29
Slika 13 - ERD	29
Slika 14 - LogIn	43
Slika 15 - Početna stranica	43
Slika 16 - Dodavanje bankomata	44
Slika 17 - Lociranje bankomata na mapi	44
Slika 18 - Pregled bankomata na mapi	45
Slika 19 - Kreiranje računa	45
Slika 20 - Stanje broja novčanica u bankomatu	46
Slika 21 - Udjeli korištenih kartica na bankomatima	46
Slika 22 - Najkorištenije kartice iz perspektive korisnika	47
Slika 23 - Top transakcije	47
Slika 24 - Top bankomati	48
Slika 25 - Poređenje transakcija na dva odabrana bankomata	48
Slika 26 - Sve transakcije za odabrani bankomat	48
Slika 27 - Ekran mobilne aplikacije	49

Slika 1 je preuzeta sa sljedećeg linka:

<http://www.enterpriseintegrationpatterns.com/patterns/messaging/SharedDataBaseIntegration.html>

Sve ostale slike, ilustracije, screenshotovi su kreirani od strane autora.

Tabele

Tabela 1 – Odgovor u JSON formatu sa više objekata.....	7
Tabela 2 – Odgovor u JSON formatu sa jednim objektom.....	8
Tabela 3 – Pregled najpoznatijih frameworka za izradu WEB servisa [5].....	9
Tabela 4 – Pregled polugodišnjih rashoda po mjesecima	11
Tabela 5 – Pregled najreprezentativnijih oblika vizualizacije podataka [7][8].....	14
Tabela 6 – Pitanja i savjeti vezani za odabir oblika vizualizacije [10].....	16
Tabela 7 – Pregled biblioteka za vizualizaciju [11].....	18
Tabela 8 – package.json.....	31
Tabela 9 – Kreiranje NodeJs aplikacije i povezivanje i korištenje ExpressJS frameworka.....	32
Tabela 10 – Konekcija na bazu podataka.....	32
Tabela 11 – Servis sa GET metodom.....	32
Tabela 12 – Servis sa POST metodom	33
Tabela 13 – Servisiranje rutiranja Angular aplikaciji	33
Tabela 14 – Inizijalizacija Angular aplikacije.....	34
Tabela 15 – Primjer registriranja ruta	34
Tabela 16 – Registracija angular modula za kontrolere	35
Tabela 17 – Primjer realizacije kontrolera	36
Tabela 18 – Povezivanje HTML-a sa Angular aplikacijom.....	37
Tabela 19 – Bind varijable iz Angular aplikacije sa HTML tagom.....	37
Tabela 20 – Bind funkcije iz Angular aplikacije sa HTML tagom.....	37
Tabela 21 – Povezivanje ng-map modula iz Angular aplikacije sa HTML-om	38
Tabela 22 – Primjer kontrolera u kom se koristi biblioteka za vizualizaciju (NVD3).....	39
Tabela 23 – Klasa za asinhrono pozivanje servisa.....	41
Tabela 24 – Primjer koda za pozivanje servisa	41
Tabela 25 – Implementacija onDone metode.....	42