Sustav za evidenciju studenata korištenjem NFC komunikacije i Raspberry Pi ugradbenog računalnog sustava

Petković, Mihovil

Master's thesis / Diplomski rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:165612

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-19**

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

MIHOVIL PETKOVIĆ

SUSTAV ZA EVIDENCIJU STUDENATA KORIŠTENJEM NFC KOMUNIKACIJE I RASPBERRY PI UGRADBENOG RAČUNALNOG SUSTAVA

Diplomski rad

Pula, lipanj, 2018.

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

MIHOVIL PETKOVIĆ

SUSTAV ZA EVIDENCIJU STUDENATA KORIŠTENJEM NFC KOMUNIKACIJE I RASPBERRY PI UGRADBENOG RAČUNALNOG SUSTAVA

Diplomski rad

JMBAG: 0313007716, redoviti student

Studijski smjer: Informatika

Predmet: Izrada informatičkih projekata

Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske tehnologije
Znanstvena grana: Informacijski sustavi i informatologija

Mentor: doc. dr. sc. Siniša Sovilj

Pula, lipanj, 2018.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Mihovil Petković, kandidat za magistra informatike ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

U Puli, 20. lipnja 2018. godine

Student

IZJAVA

o korištenju autorskog djela

Ja, Mihovil Petković dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom „Sustav za evidenciju studenata korištenjem NFC komunikacije i Raspberry Pi ugradbenog računalnog sustava“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 20. lipnja 2018. godine

Student

SVEUČILIŠTE JURJA DOBRILE U PULI
ODJEL ZA INFORMACIJSKO-KOMUNIKACIJSKE TEHNOLOGIJE

Pula, 1. ožujka 2017.

DIPLOMSKI ZADATAK

Pristupnik:	**Mihovil Petković (0313007716)**
Studij:	Sveučilišni diplomski studij Informatike
Naslov (hrv.):	**Sustav za evidenciju studenata korištenjem NFC komunikacije i Raspberry Pi ugradbenog računalnog sustava**
Naslov (eng.):	System for tracking student attendance using NFC communication and Raspberry PI embedded computer system
Opis zadatka:	Zadatak je realizirati sustav koji će omogućiti kontrolu prisutnosti studenata putem beskontaktnih kartica i/ili pametnih telefona. Potrebno je proučiti način rada Android Things i Raspbian operacijskog sustava na Raspberry Pi računalu te eksperimentom usporediti njihove karakteristike kao što su količina prometa, kašnjenje, trajanje uspostave veze, zauzeća memorije i opterećenja procesora. Proučiti način rada vanjskog NFC čitača, povezati čitač s Raspberry Pi računalom te omogućiti prepoznavanje NFC beskontaktnih kartica i kartica emuliranih od strane Android pametnog telefona. Osmisliti web servis za primanje i slanje poruka prema Raspberry računalu, povezati servis s bazom podataka te oblikovati grafičko sučelje za korisnike.

Zadatak uručen pristupniku: 1. ožujka 2017.

Mentor:

Siniša Sovilj

_____

doc.dr.sc. Siniša Sovilj

Sadržaj

Uvod 1

1. Informacijski sustav visokih učilišta 2

2. Studentske iskaznice 6

3. RFID identifikacija 10

4. NFC komunikacija 12

5. NFC čitač kartica 14

6. Raspberry Pi ugradbeni sustav 16

7. Raspbian operacijski sustav 18

8. Android Things operacijski sustav 20

9. REST Web servis 22

9.1. Spring web servis 24

10. MySQL baza podataka 28

11. Angular web razvojni sustav 29

12. Arhitektura sustava za evidenciju 33

13. Implementacija sustava za evidenciju 35

13.1. Baza podataka 46

13.2. Opis web aplikacije 47

13.2.1. Prijava u sustav 48

13.2.2. Korisnici 49

13.2.3. Studenti 50

13.2.4. Kolegiji 51

13.2.5. Predavanja 52

13.2.6. Predavaonice i odjeli 53

13.2.7. Evidencija predavanja 54
13.2.8. Evidencija studenata 55
Zaključak 57
Literatura 58
Popis slika 60
Sažetak 62
Summary 63
Privitak 64

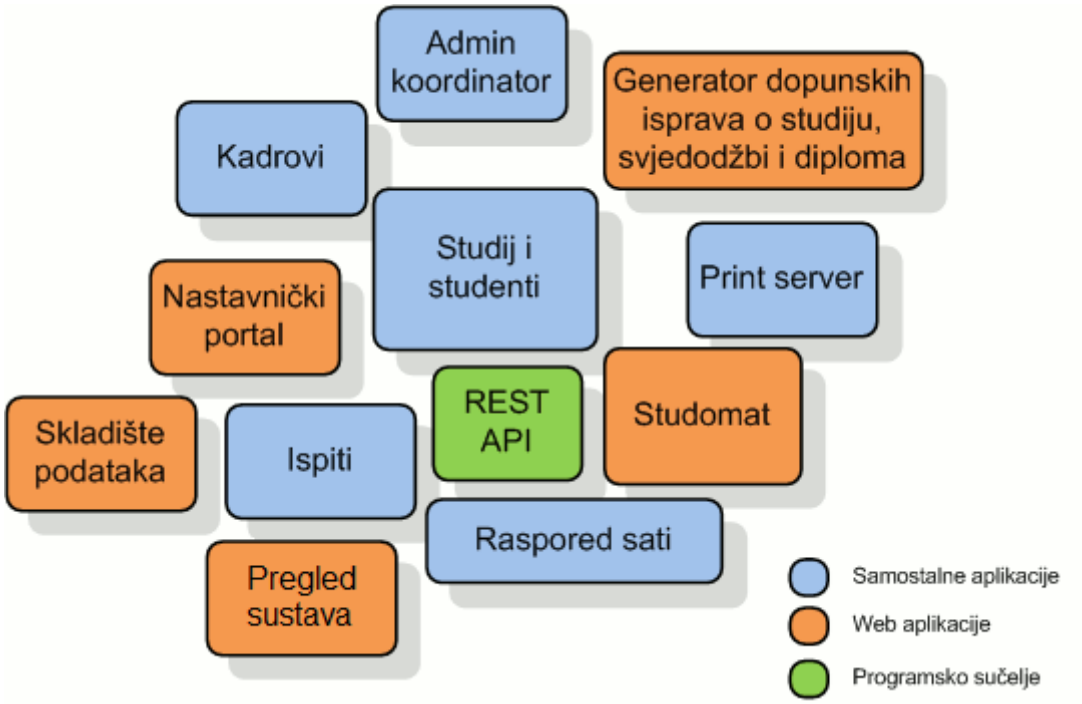
Uvod

Evidencija prisutnosti studenata predstavlja samo jedan dio svih aktivnosti koje obavlja visoko učilište, te čini sastavni dio svakog studentskog predavanja. Svrha evidencije je pratiti polaznost kolegija što predstavlja jedan od osnovnih uvjeta za ostvarivanje prava polaganja ispita. Trenutne metode evidencije se uglavnom odnose na pojedinačno popisivanje studenata kroz evidencijski list predavanja. Predavač mora čekati dok se svi studenti potpišu i zatim brojati ili prozivati studente da bi se uvjerio u vjerodostojnost evidentiranih podataka. Takav način evidencije odvraća pažnju studenata i dodatno usporava sam početak predavanja kojeg fokusira na proces evidencije. Dodatni problem predstavlja činjenica da je za administraciju i kontrolu evidentiranih dolazaka potrebno ručno prepisivati podatke s evidencija na papiru i unositi ih u računalo kroz program za obradu podataka. Ovakav način evidencije je dugotrajan i neefikasan, te se u konačnici može uvelike ubrzati i olakšati primjenom informacijskog sustava evidencije. Svrha ovog rada je istražiti mogućnosti automatizacije procesa evidencije studenata, te obuhvatiti i opisati tehnologije koje bi dovele do realizacije takvog sustava. Cilj rada je realizirati i opisati koncept sustava evidencije i svih njegovih komponenti. U radu su opisane RFID i NFC bežične tehnologije za prijenos podataka, principi rada NFC čitača kartica, način rada i korištenje Raspberry Pi ugradbenog računalnog sustava, korištenje web servisa, te klijentskih web razvojnih okruženja (engl. *web frameworks*).

1. Informacijski sustav visokih učilišta

Vodeći se pretpostavkom da visoko učilište već koristi određenu vrstu informacijskog sustava, tijekom planiranja načina realizacije sustava za evidenciju studenata, potrebno je razmišljati o što lakšoj integraciji sa postojećim informacijskim sustavom jednog visokog učilišta. Sustav za evidenciju je nužno implementirati na način da je u stanju razmjenjivati informacije s drugim informacijskim sustavima, te da je u što većoj mjeri fleksibilan i jednostavan za buduću nadogradnju. Tada je podatke o studentima, predavačima i njihovim kolegijima moguće administrirati kroz vlastitu aplikaciju unutar sustava za evidenciju ili dobivati takve podatke iz već postojećeg informacijskog sustava kojeg visoko učilište koristi. Za komunikaciju među sustavima nužno je implementirati određenu vrstu web servisa koji je u stanju razmjenjivati informacije putem standardiziranih HTTP protokola. Na taj način sustav za evidenciju može u bilo kojem trenutku dobiti podatke o svim bitnim parametrima za što točnije praćenje i kontrolu prisutnosti studenata na predavanjima. Za informatizaciju visokih učilišta u Republici Hrvatskoj zaduženo je Srce, odnosno Sveučilišni računalni sustav Sveučilišta u Zagrebu. Srce je središnja ustanova cjelokupnog sustava znanosti i visokog obrazovanja Republike Hrvatske. Ono se brine o izgradnji, razvoju i održavanju informacijske infrastrukture, te omogućuje interoperabilnost s drugim visokim učilištima u državi. Usluge koje Srce nudi uključuju podršku za pristup Internetu, nadzor i sigurnost IT sustava, razne aplikacije za obrazovanje, te nekoliko informacijskih sustava. Da bi visoko učilište započelo s informatizacijom potrebno je zatražiti uvođenje informacijskog sustava visokih učilišta – ISVU. Prema informacijama Sveučilišnog računalnog centra oko 70% visokih učilišta u Republici Hrvatskoj koristi ISVU sustav. ISVU je besplatan informacijski sustav financiran od strane ministarstva znanosti i obrazovanja. Njegova uloga je povezivanje različitih službi i odjela unutar samog visokog učilišta. Takva informatička poveznica olakšava međusobnu komunikaciju što za posljedicu ima smanjivanje količine administrativnih poslova od strane stručnih službi, te povećanje točnosti informacija unutar raznih odjela. Izbjegava se upisivanje istih podataka više puta i pruža se bolji nivo automatizacije procesa.

Olakšan je pristup do svih podataka, jer su one tada dostupne korisnicima putem Interneta u svakom trenutku. Uvođenje informacijskog sustava u konačnici omogućuje integriranje drugih sustava u rad učilišta, pa tako i sustava za evidenciju prisutnosti studenata. (Srce, Djelatnost srca, 2018). ISVU sustav je podijeljen u nekoliko programskih modula koji su prikazani na slici Sl. 1.1. Moduli visokom učilištu omogućuju automatizaciju raznih aktivnosti kao npr. upis i praćenje studenata, upisivanje nastavnog plana, rezervaciju dvorana i izradu satnice. Kroz modul skladišta podataka omogućeno je pregledavanje raznih statistika o uspješnosti studenata, prolaznosti na ispitima, prosjeku ocjena na ispitnim rokovima i sl. Studentima je, putem web aplikacije Studomat, olakšana prijava i odjava ispita, pregled statusa i ocjena pojedinih kolegija, te dobivanje raznih potvrda. Studenti zauzvrat ne moraju čekati u redu i uvelike se smanjuje opterećenost studentske referade (Srce, 2018). Integracija s ISVU sustavom olakšala bi administriranje matičnih podataka o studentima. Predavači bi i dalje koristili već postojeći ISVU sustav, dok bi sustav za evidenciju dobivao samo one informacije koje su mu potrebne za što točniju kontrolu prisutnosti studenata na predavanjima.



Sl. 1.1 Prikaz modula ISVU sustava

(izvor: http://www.isvu.hr/javno/hr/index.shtml)

Uz ISVU bitno je spomenuti i informacijski sustav studentskih prava – ISSP, te informacijski sustav akademskih kartica – ISAK. Informacijski sustav studentskih prava je sustav putem kojeg visoka učilišta mogu evidentirati i pratiti studentska prava. Sustav se u praksi najviše koristi za provjeru i ostvarivanje prava na subvencioniranu prehranu u raznim studentskim restoranima. Njegova uloga je pružanje osnovnih informacija o studentima, visokim učilištima, raznim restoranima studentske prehrane, te provjeri podataka prema broju studentske iskaznice (Srce, 2018) .

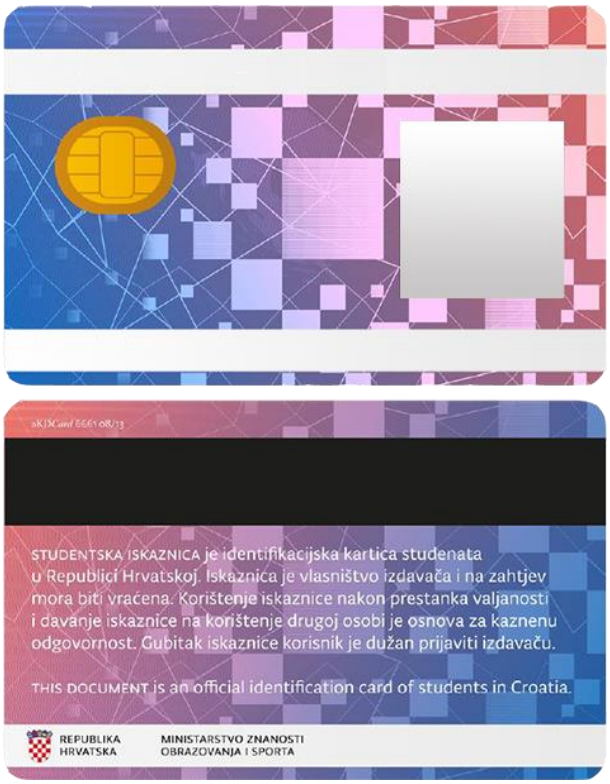
Informacijski sustav akademskih kartica je sustav koji prvenstveno služi za provjeru statusa akademskih iskaznica. Akademska iskaznica je službena isprava studenata u Republici Hrvatskoj. Ona, kao javna isprava, dokazuje status studenta i služi za njegovu identifikaciju. Pravilnikom o studentskoj ispravi točno je definiran izgled, sadržaj, izrada i izdavanje studentske iskaznice. ISAK osigurava podatke potrebne za tiskanje iskaznica i sprema podatke o statusu procesa tiskanja. Podatci o pravu na iskaznicu kao i sami matični podatci studenata ISAK prikuplja iz ISSP sustava. Izrada iskaznica, distribucija i naplata istih nije prepuštena ministarstvu, već je povjerena tvrtci AKD iz Zagreba (Srce, ISAK, 2018). Određenim podatcima iz ISSP sustava moguće je pristupiti putem službenih stranica Sveučilišnog računalnog centra Srce, dok je ostatak podataka dostupan kroz ISSP REST API programsko sučelje. Programsko sučelje omogućuje pristup informacijama kroz HTTP URI adrese i standardne HTTP metode: GET, PUT, POST i DELETE. Sve što vanjski sustav, kao npr. sustav za evidenciju studenata, treba napraviti jest poslati HTTP zahtjev i obraditi odgovor REST programskog sučelja. ISVU sustav također ima svoj vlastiti REST API putem kojeg mu je omogućena integracija s drugim vanjskim sustavima koji koriste HTTP protokol za razmjenu informacija. Trenutno su definirane dvije varijante REST API sučelja: probni i produkcijski. Dvije verzije olakšavaju vanjskim sustavima razvoj i testiranje komunikacije putem REST API-a. Funkcionalnosti se mogu provjeriti na probnom okruženju s testnim podatcima, te relativno jednostavno preusmjeriti na produkcijski ako su sve funkcionalnosti zadovoljene. Podatci su zaštićeni na način da jedno visoko učilište može pristupati samo svojim podatcima i procedurama. Ono ima mogućnost, kroz ISVU modul Koordinator, dodijeliti pravo pristupa vlastitim podatcima i nekom vanjskom sustavu. Pristup se definira kreiranjem

korisničkog imena i lozinke koji onda vanjski sustav koristi za autorizaciju prema ISVU sustavu. Autorizacija je tipa *Basic Authentication* koja se šalje u zaglavlju svakog HTTP zahtjeva. Pristup je moguće i dodatno ograničiti po jednoj ili određenom rasponu IP adresa. Kako bi se spriječilo pretjerano opterećivanje sustava postavljeno je i ograničenje na dnevni broj zahtjeva prema ISVU sustavu po formuli *200 * broj kolegija + 200 * broj studenata* (Srce, 2018).

Sustav za evidenciju prisutnosti studenata koji je opisan u diplomskom radu koristi vlastitu bazu podataka i vlastiti REST web servis za spremanje matičnih podataka o studentima. Web servis je realiziran na način da se, unatoč tome što nije direktno povezan s ISVU i ISSP REST API sučeljima, može relativno jednostavno integrirati s njima u budućnosti. Tada bi za povezivanje na ISVU i ISSP sustave bilo potrebno zatražiti odobrenje od vlasnika sustava, u ovom slučaju Sveučilišnog računalnog centra, te visokog učilišta koje takav sustav i koristi. Kroz web aplikaciju, koja je realizirana u diplomskom radu, omogućeno je administriranje podataka o studentima, predavačima, njihovim kolegijima i predavanjima. Administrator sustava može dodavati nove korisnike, dodati im uloge, definirati kolegije, povezati ih s predavačima i definirati mjesto i vrijeme izvođenja pojedinog predavanja. Na taj način sustav može povezati predavača s njegovim predavanjem, prema mjestu i vremenu izvođenja odrediti kojem kolegiju predavanje pripada, te dobiti popis studenata koji su upisali takav kolegij. Praćenje i kontrola prisutnosti se tada može odraditi korištenjem studentskih iskaznica tzv. X-ica koje bi služile kao sredstvo identifikacije studenata. Sustav za evidenciju bi na temelju podataka sa studentske iskaznice povezao studenta s njegovim kolegijem te evidentirao njegovu prisutnost na predavanju. Popis evidentiranih studenata bi tada mogao pregledavati i nositelj ili asistent na kolegiju, te na temelju toga odrediti koji studenti imaju pravo izaći na ispit i položiti kolegij. Za takvu implementaciju potrebno je realizirati određenu vrstu čitača kartica koji je u stanju čitati podatke s studentske iskaznice i prosljeđivati ih prema web servisu.

2. Studentske iskaznice

Da bi se student uspješno evidentirao na predavanjima sustav za evidenciju mora biti u stanju pročitati broj njegove studentske iskaznice, te na temelju njega autorizirati i povezati studenta s njegovim kolegijima. Studentska iskaznica je identifikacijski dokument temeljem kojeg student dokazuje svoj akademski status. Iskaznicu, način rada i sve njezine podatke propisuje Ministarstvo znanosti, obrazovanja i sporta. Iskaznica ima oblik kreditne kartice i na prednjoj strani sadrži broj iskaznice, informacije o vlasniku, njegovo ime i prezime, te fotografiju i naziv ustanove koju student, odnosno vlasnik iskaznice pohađa. Svaka studentska iskaznica ima tri načina prijenosa podataka: magnetskom trakom, čipom ili beskontaktno. Na poleđini kartice nalazi se magnetska traka u kojoj su upisani podatci o imenu i prezimenu studenta, njegovom OIB-u, te broju studentske iskaznice. Podatke s magnetskog zapisa je moguće pročitati korištenjem magnetskog čitača kartica. Takav zapis se sve manje koristi u praksi, ali je i dalje aktualan na raznim kreditnim i karticama korištenim u svrhu javnog prijevoza. Slično kao i kod kreditnih kartica, osim magnetskog zapisa, svaka iskaznica ima ugrađeni čip čiju vrstu i strukturu određuje Ministarstvo. U čip se upisuju svi podatci o studentu kao i kod magnetskog zapisa uz dodatak nekoliko dodatnih informacija vezanih uz usluge pojedinog visokog učilišta. U diplomskom radu istražene su mogućnosti beskontaktnog prijenosa podataka putem antene ugrađene u samu studentsku iskaznicu. Sam proces evidencije je jednostavniji i lakši samo prislanjanjem iskaznice na beskontaktni čitač naspram provlačenja kroz magnetski čitač kartica. Čitači beskontaktnih kartica predstavljaju korak u smjeru korištenja novih tehnologija autentifikacije, te kao takvi čine dobar izbor kod implementacije sustava za evidenciju. Na slici Sl. 2.1 prikazan je izgled studentske iskaznice s prednje i stražnje strane. Važno je napomenuti da je svaka iskaznica neprenosiva, odnosno može ju koristiti samo osoba na koju se odnose podatci zapisani na iskaznici. To osigurava njezinu autentičnost i jedinstvenost među studentima (Srce, 2018).



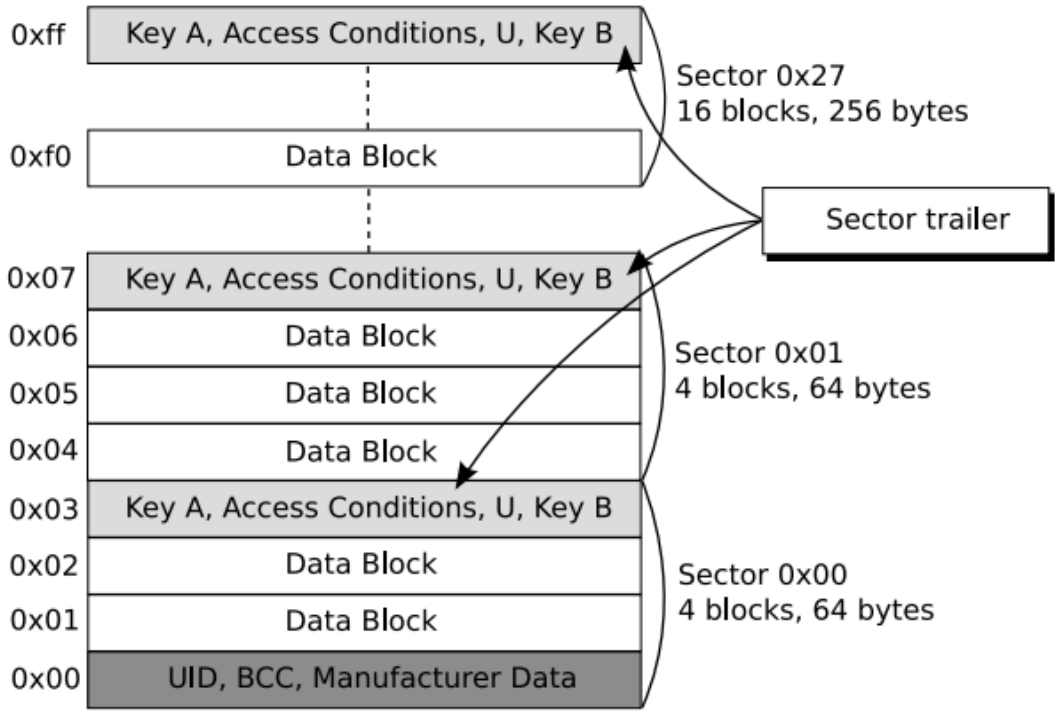
Sl. 2.1 Prikaz studentske iskaznice

(Izvor: http://www.srce.unizg.hr/isak/studentska-iskaznica)

Studentske iskaznice spadaju u vrstu beskontaktnih kartica pod nazivom MIFARE Classic 4K (engl. *Mikron FARE Collection System*). MIFARE je oznaka serije mikročipova proizvedenih od tvrtke NXP, globalnog proizvođača poluvodiča smještenog u Nizozemskoj. NXP je jedna od vodećih tvrtki na području identifikacijske tehnologije čija rješenja za evidenciju je moguće naći u bankama, javnom prijevozu i transportu, te automobilskoj i komunikacijskoj industriji (NXP, 2018). MIFARE Classic kartice su jedne od najpoznatijih beskontaktnih kartica trenutno na tržištu. Prisutne su u više od 750 gradova svijeta s preko 10 milijardi prodanih primjeraka. Zbog svoje pouzdanosti i niske cijene trenutno drže više od 80% udjela tržišta beskontaktnih kartica. Najviše su korištene kao identifikacijske iskaznice u raznim okolinama, primjerice za identifikaciju radnika u tvrtkama ili putnika u javnom prijevozu, gledatelja na sportskim natjecanjima i sl. MIFARE Classic kartice, kao tehnologiju prijenosa podataka, koriste radiofrenkventnu identifikaciju (engl. *Radio Frequency Identification*), odnosno RFID, čiji način rada je detaljno opisan u sljedećem poglavlju.

MIFARE oznaka pokriva 4 tipa beskontaktnih kartica koje se baziraju na RFID ISO/IEC 14443 standardu komunikacije i funkcioniraju na frekvenciji od 13.56 Mhz. Standard je opisan u 4 dijela u kojima se definiraju fizičke karakteristike kartica, korištena radijska frekvencija i signalno sučelje između kartice i čitača, inicijalizacija i protokoli anti-kolizije tj. situacija kada je na istom čitaču prisutno više kartica, te protokoli prijenosa podataka. MIFARE Classic kartice ne prate u potpunosti ISO specifikaciju, već za prijenos podataka koriste vlastiti protokol. MIFARE protokol se razlikuje u tome što komunikacija započinje s autentifikacijom, a potom je u potpunosti kriptirana. Za kriptiranje podataka na kartici zaslužan je privatni algoritam šifriranja Crypto-1 razvijen od tvrtke NXP, s 48 bitnim simetričnim ključevima. Studentske iskaznice se odlikuju oznakom 4K koja označava količinu memorije od 4096 bajta koju je moguće iskoristiti za spremanje podataka. Svaka kartica dolazi s tvornički upisanim UID brojem (engl. *Unique identifier*) dužine 7 bajta koji osigurava jedinstveni identifikacijski broj kartice. UID identifikator je pohranjen unutar dijela memorije koji je rezerviran za tvorničke podatke o kartici. Takav dio kartice se ne može prepisati ili izbrisati i moguće ga je pročitati bez potrebe za simetričnim tajnim ključevima i dekriptiranjem.

Brzina prijenosa podataka s kartice na RFID čitač iznosi 106 kbit/s, dok je maksimalna udaljenost prijenosa podataka ograničena na oko 100 milimetara, ovisno o tipu čitača kartica (NXP, 2018). Memorija kartice je podijeljena u blokove od 16 bajta. Takvi blokovi se zatim grupiraju u sektore od kojih se prvih 32 sektora sastoje od 4 bloka, dok ostalih 8 imaju po 16 blokova. Prvi blok prvog sektora sadrži podatke o jedinstvenom identifikacijskom broju kartice, te podatcima o proizvođaču i njegov sadržaj je moguće samo čitati. Zadnji blok svakog sektora nije moguće iskoristiti za spremanje podataka, već se on koristi za spremanje kriptografskih ključeva. Ključevi su dužine 6 bajta i tvornički su postavljeni na određenu vrijednost. Tajni ključevi A i B su konfigurirani na način da se ključ A nikad ne može pročitati, a ključ B se može konfigurirati za dozvoljeno čitanje ili zaključati kao ključ A. Ključevi se koriste za autentifikaciju bilo kakve memorijske operacije čitača nad pojedinim sektorom, osim sektora s tvornički podatcima. Slika Sl. 2.2 opisuje raspored memorije kod MIFARE Classic 4K kartice.



Sl. 2.2 Mifare Classic raspored memorije

(izvor: http://www.cs.ru.nl/~flaviog/publications/Attack.MIFARE.pdf)

Kriptiranje sadržaja kartice Crypto-1 algoritmom više se ne smatra sigurnim zbor raznih napada koji su prezentirani još 2008.g. kada su istraživači sa Sveučilišta Radboud u Nizozemskoj uspjeli dokazati uspješno probijanje enkripcije. Tvrtka koja je proizvela kartice, NXP je u međuvremenu spriječio zlouporabu i hakiranje podataka izdavanjem novih varijanta kartica imena MIFARE Classic EV1 i MIFARE DESfire s DES i AES algoritmima enkripcije. DES i AES su algoritmi koji koriste enkripciju simetričnog ključa koji omogućuju bolji nivo zaštite podataka zapisanih na samoj kartici. Nove kartice su dizajnirane da istovremeno pružaju kompatibilnost s čitačima prethodnih generacija kartica, te da prijelaz njih bude što lakši za trenutne korisnike MIFARE Classic kartica (Gerhard Gans, 2008).

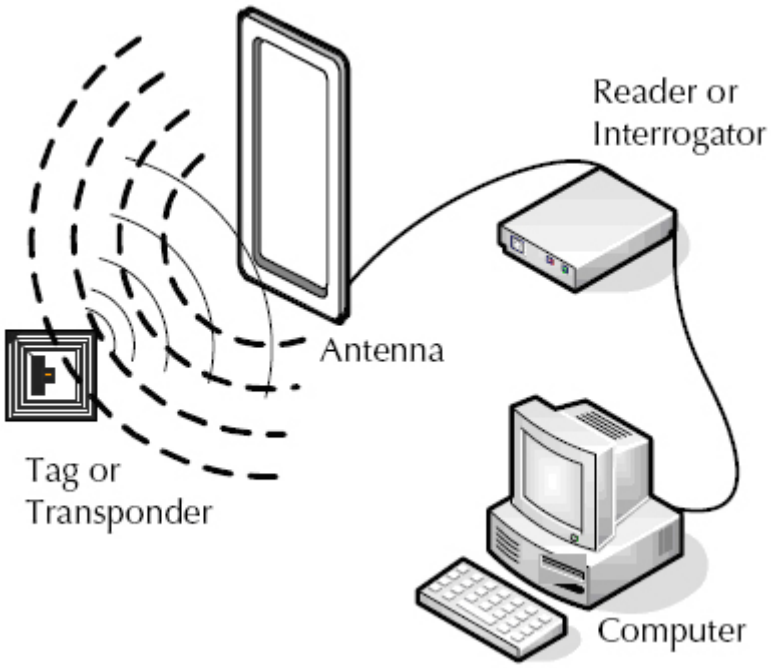
3. RFID identifikacija

Radiofrenkventna identifikacija je tehnologija koja se koristi za identifikaciju i razmjenu podataka između čitača i beskontaktnih kartica, odnosno transpondera. Transponder je uređaj koji sadrži mikročip i antenu, te se može proizvesti u raznim veličinama i oblicima, kao naljepnice ili kartice i sl.. Transponderi se generalno razlikuju po protokolu kojem komuniciraju i veličini memorije u kojoj su pohranjeni podatci. Količina memorije varira po vrstama transpondera, ali se generalno kreće od nekoliko desetaka bitova do nekoliko desetaka kilobajta. Studentske iskaznice, odnosno MIFARE Classic kartice, koriste RFID standard ISO-14443. ISO (engl. *International Organization for Standardization*) je međunarodna organizacija za normalizaciju koja, u ulozi posredničkog tijela, pomaže u razvoju internacionalnih standarda. Takvi standardi pomažu kod bolje interoperabilnosti između različitih proizvođača na svjetskom tržištu. RFID kao tehnologija identifikacije čini bazu za razumijevanje komunikacije između čitača i studentski kartica, ali i kao osnova za razumijevanje NFC standarda komunikacije, čiji čitač kartica je korišten u realizaciji sustava za evidenciju studenata.

RFID sustavi se generalno dijele u dvije vrste: aktivni i pasivni. Sustavi s pasivnim transponderima rade na način da komunikaciju započinje čitač emitiranjem snažnog radio signala niske frekvencije. Takav signal detektira transponder koji, ulaskom u polje radijske frekvencije, aktivira svoj mikročip i odašilje natrag kratku informaciju, odnosno svoj jedinstveni identifikator. Pasivni RFID sustavi omogućuju jednosmjernu komunikaciju koja, ovisno o veličini i osjetljivosti antene čitača, te jačini radio signala, može funkcionirati na udaljenosti od nekoliko centimetara do nekoliko metara. Čitači kratkog dometa se obično dijele u niskofrekventne čitače koji rade na frekvenciji od 125 Khz. Takvi čitači su najmanjeg dometa i najslabiji, ali samim time i s najnižom prodajnom cijenom. Visokofrekventni čitači funkcioniraju na frekvenciji od 13.56 Mhz i omogućuju čitanje i pisanje podataka na udaljenostima od nekoliko centimetara. Čitači većih frekvencija spadaju u ultra visokofrekventne i obično su većeg dometa i bržeg protoka podataka, ali su zato i znatno skuplji (Igoe, 2012). NFC čitač kartica,

korišten u realizaciji diplomskog rada, spada u visokofrekventne čitače s frekvencijom rada od 13.56 Mhz. Na takvoj frekvenciji komunikacije postoje nekoliko ISO standarda, od kojih je za komunikaciju s studentskom iskaznicom važan samo ISO-14443 standard. Problem je što unutar samog standarda postoje nekoliko implementacija ovisno o proizvođaču kartice, stoga je bitno odabrati čitač koji odgovara tipu beskontaktne kartice. Na slici Sl. 3.1 prikazan je način rada pasivnog NFC sustava.

Sustavi s aktivnim transponderima primaju signale od čitača putem antene, ali koriste vlastito napajanje putem interne baterije. Aktivni RFID sustavi mogu odašiljati na puno većoj udaljenosti, ali im je zato prodajna cijena znatno veća. Primjer korištenja aktivnog RFID sustava se može naći u načinu naplate cestarine. Automobili koji dolaze na naplatne kućice, uz pomoć aktivnog RFID transpondera i smanjenu brzinu kretanja, mogu obaviti novčanu transakciju i platiti cestarinu bez zaustavljanja na naplatnim kućicama.

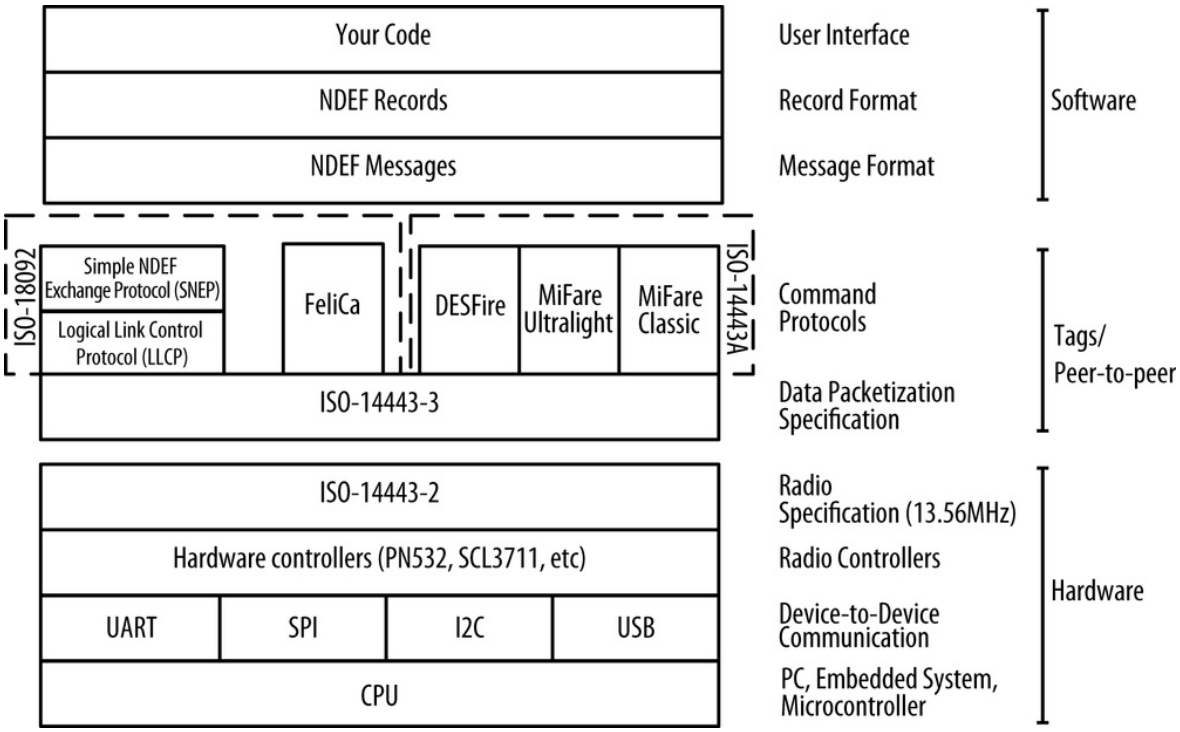


Sl. 3.1 Pasivni RFID sustav identifikacije

(izvor: https://www.epc-rfid.info/rfid)

4. NFC komunikacija

NFC (engl. *Near Filed Communication*) je komunikacija bliskog polja koja spada u bežične tehnologije za komunikaciju na malim udaljenostima. NFC komunikacija ne definira nove protokole, već se dobrim dijelom bazira na frekvencijama i standardima definiranim kod RFID identifikacije. Na fizičkom sloju NFC radi na ISO-14443 specifikaciji i frekvenciji od 13.56 MHz. Udaljenost unutar koje NFC može prenositi podatke iznosi svega nekoliko centimetara i ograničena je protokolom, a ne snagom i vrstom čitača ili kartice. NFC čitači imaju tri moda rada. Mogu se koristiti za čitanje/pisanje prema pasivnom transponderu, mogu emulirati NFC transponder, odnosno ponašati se kao NFC oznaka kada se nalaze u blizini drugog čitača, te raditi u *peer-to-peer* modu u kojem dvosmjerno razmjenjuju podatke prema drugom čitaču. Dvosmjerna komunikacija prati ISO-18092 standard koji koristi SNEP (engl. *Simple NDEF Exchange Protocol*) protokol za razmjenu poruka. Kao što je prikazano na slici Sl. 4.1 postoji nekoliko slojeva arhitekture u načinu rada NFC komunikacije. Na fizičkom sloju, kojeg definira standard ISO-14443-2, mikrokontroler komunicira s NFC čitačem putem različitih sabirnica (UART, SPI, I2C, USB), te odašilje frekvenciju od 13.56 MHz prema drugim pasivnim transponderima ili aktivnim uređajima. Na srednjem sloju NFC prati ISO-14443A protokol za komunikaciju s pasivnim oznakama i ISO-18092 protokol za *peer-to-peer* komunikaciju s drugim čitačima. Treći sloj NFC arhitekture definira NDEF format podataka (engl. *NFC Data Exchange Format*), koji predstavlja univerzalni format za razmjenu podataka između NFC čitača kartica (Igoe, 2012).

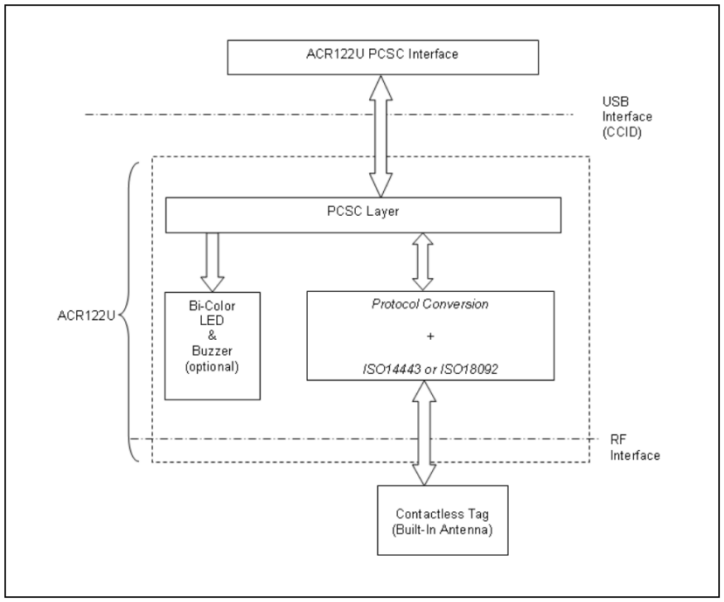


Sl. 4.1 NFC arhitektura

(izvor: Igoe, figure 2-1)

5. NFC čitač kartica

U izradi diplomskog rada korišten je NFC čitač kartica *ACR122U* kineskog proizvođača imena ACS (engl. *Advanced Card Systems*). ACS je globalna tvrtka prisutna u preko 100 zemalja svijeta koja je specijalizirana za proizvodnju raznih čitača kartica, te beskontaktnih kartica. *ACR122U* radi na frekvenciji od 13.56 MHz s brzinom čitanja od 424 kbit/s. U potpunosti podržava ISO-18092 i ISO-14443 standarde koji mu omogućuju komunikaciju s NFC i RFID beskontaktnim karticama. Podržani su svi tipkovi MIFARE, te FeliCa i sva četiri tipa NFC kartica. Čitač ima mogućnost čitanja i pisanja na udaljenosti do maksimalno 5 cm, ovisno o tipu kartice. *ACR122U* je prvi NFC čitač kartica u skladu s CCID (engl. *chip card interface device*) specifikacijom koja mu omogućuje povezivanje s računalom putem USB 2.0 priključka. Proizvođač garantira rad čitača na većini operacijskih sustava kao što su Windows, Linux, Mac OS, te Android. Na Windows sustavu je korištenje čitača dodatno olakšano činjenicom da čitač ne zahtjeva instaliranje upravljačkih programa zato što oni već dolaze ugrađeni u sam operacijski sustav. Čitač na prednjoj strani ima led lampicu koja signalizira uspješno ili neuspješno čitanje koje je praćeno zvučnim signalom (ACS, 2018).



Sl. 5.1 Komunikacijski dijagram *ACR122U* čitača kartica

(izvor: https://www.acs.com.hk/en/products/3/acr122u-usb-nfc-reader)

Na slici Sl. 5.1 prikazan je komunikacijski dijagram *ACR122U* NFC čitača kartica. Dijagram opisuje PC/SC (engl. *Personal Computer/Smart Card*) sučelje preko čije specifikacije NFC čitač kartica komunicira s vanjskim uređajima. Tvrtka Microsoft je PC/SC specifikaciju ugradila u svoje Windows operacijske sustave, dok je besplatna verzija za Linux distribucije dostupna pod nazivom *pcsc-tools*. Nakon što se alat instalira na Linux operacijski sustav, čitač kartica i podatci s studentske iskaznice moguće je provjeriti naredbom *pcsc_scan*. Skeniranje kartice prikazuje osnovne informacije kao što je tip i vrsta kartice, te ATR (engl. *Answer To Reset*). Slika Sl. 5.2 prikazuje ACR122U čitač kartica korišten u izradi diplomskog rada.

Sl. 5.2 ACR122U NFC čitač kartica

(izvor: autor)

6. Raspberry Pi ugradbeni sustav

Raspberry Pi je serija popularnih *single-board* računala dizajnirana u Velikoj Britaniji od strane Raspberry Pi zaklade. Zaklada je neprofitna organizacija osnovana 2009. godine s ciljem promoviranja računalnih znanosti i programiranja u školama. Raspberry Pi modeli su integrirani sustavu na čipu (engl. *System on Chip*) koji integriraju sve komponente računala na jednu pločicu veličine kreditne kartice. Postoji nekoliko generacija Raspberry Pi ugradbenih sustava, od kojih je u diplomskom radu korištena zadnja, treća serija. Svaka serija dolazi sa različitim karakteristikama sklopovlja i prodajne cijene. Sklopovlje treće generacije ovog ugradbenog sustava uključuje 64-bitni četverojezgreni ARM procesor brzine 1.2 GHz, ugradbenu grafičku karticu brzine 400 MHz, te radnu memoriju od 1 GB. Raspberry ugradbeni sustav nema tvrdi disk, kao osobna računala, već operacijski sustav pokreće sa mikro SD kartice (engl. *Secure Digital*). Osim kompaktnog dizajna Raspberry Pi 3 ima izvrsne mogućnosti povezivanja s drugim uređajima. Kao što je prikazano na slici Sl. 6.1 ugradbeni sustav ima 4 USB 2.0 priključka, 40 GPIO pinova, HDMI video izlaz, Ethernet priključak, te ugrađeni Wi-Fi i Bluetooth 4.1. Osim za učenje, moguće ga je koristiti i za multimediju, igrice, kao zamjenu za web server, te za povezivanje s raznom elektronikom. Na Raspberry Pi se preporučuje instalirati Raspbian, optimiziranu inačicu Linux operativnog sustava temeljenu na Debian distribuciji. Raspbian omogućuje pisanje programa u nekoliko vrsta programskih jezika, te pruža podršku za višedretveni rad, odnosno izvođenje više procesa paralelno (Schmidt, 2012). Raspberry podržava i mnoge druge operacijske sustave kao što su Ubuntu Mate, Windows 10 IoT Core, Android Things i sl. U diplomskom radu istražene su mogućnosti povezivanja NFC čitača kartica s Raspberry Pi ugradbenim sustavom kroz Raspbian i Android Things operacijske sustave. Njegova uloga je povezati se s čitačem kartica, te komunicirati s udaljenim web servisom preko Interneta. Raspberry se na Internet može povezati direktno Ethernet priključkom ili koristeći Wi-Fi vezu.

Sl. 6.1 *Raspberry Pi* ugradbeni računalni sustav

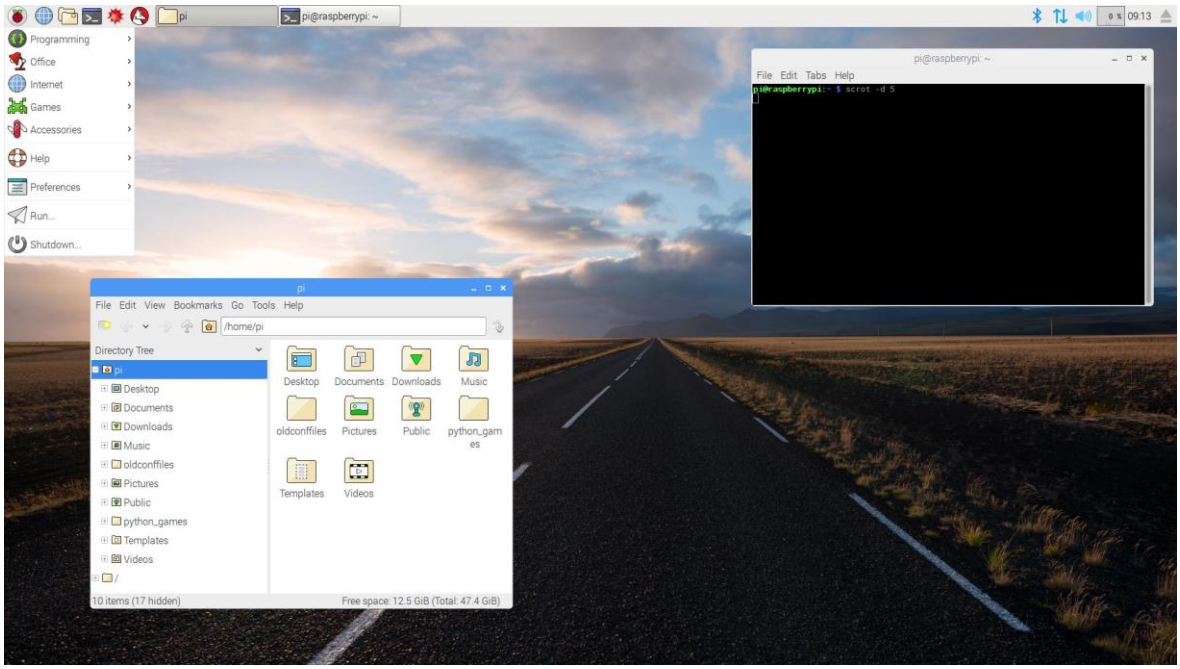
(Izvor: autor)

7. Raspbian operacijski sustav

Raspbian je besplatni operacijski sustav konfiguriran specifično za korištenje na Raspberry Pi ugradbenim sustavima. Njegova arhitektura se temelji na Linux distribuciji Debian Wheezy, koja je prilagođena ARM-v6 procesorima koje koristi Raspberry Pi. Raspbian se sastoji od preko 35 000 paketa koji uključuju razne predefinirane programe poput: Linux terminala, uređivača teksta Leafpad, Libre Office paket programa za obradu teksta, preglednik za PDF dokumente, Firefox i Chrome web preglednike, programski jezik Python itd. (Raspberry, 2018).

Prednost Raspberry Pi ugradbenog sustava, naspram pločica s mikrokontrolerima, kao što su Arduino pločice, je u tome što podržavaju izvršavanje čitavog Linux operacijskog sustava. Operativni sustav daje korisniku mnoge funkcionalnosti na visokom nivou kao što su konfiguracija Internet mreže, korisničkog sučelja, pristup datotečnom sustavu itd. Raspbian, kao i većina Linux distribucija, dolazi s predinstaliranim programskim jezikom Python. Python je moderan skriptni jezik s puno programskih biblioteka, te mnoštvom značajki za programiranje na visokoj razini. Na Raspbian je moguće instalirati i koristiti i druge programske jezike kao: C++, Java, JavaScript, PHP itd. Ovakav širok raspon jezika omogućuje programiranje raznih aplikacija u različitim programskim jezicima (London, 2015).

Podrška za razvoj aplikacija na Raspbian operacijskom sustavu olakšava činjenica da je Raspbian temeljen na Debian Linux distribuciji. Jednostavan pronalazak različitih upravljačkih programa (engl. *drivers*), te mnoštvo primjera koda za spajanje na razne uređaje omogućuje izvrsna podrška zajednice na Internetu. Na slici Sl. 7.1 prikazano je početno sučelje Raspbian operacijskog sustava.



Sl. 7.1 Raspbian operacijski sustav

(izvor: https://www.techrepublic.com/pictures/screenshots-raspberry-pi-gets-a-new-look-desktop/)

Operacijski sustav ima grafičko sučelje slično onome na drugim Linux distribucijama. Na glavnoj alatnoj traci pruža se prikaz otvorenih prozora, te glavni meni s aplikacijama koje su kategorizirane u raznim kategorijama: Programming, Office, Internet, Games i sl.. Na slici Sl. 7.1 je prikazan i preglednik datoteka koji s lijeve strane ima popis svih direktorija, a s desne strane sadržaj odabranog direktorija s svim pripadajućim datotekama. U gornjem desnom kutu Raspbian prikazuje trenutno vrijeme, te status Wi-Fi i Bluetooth konekcija.

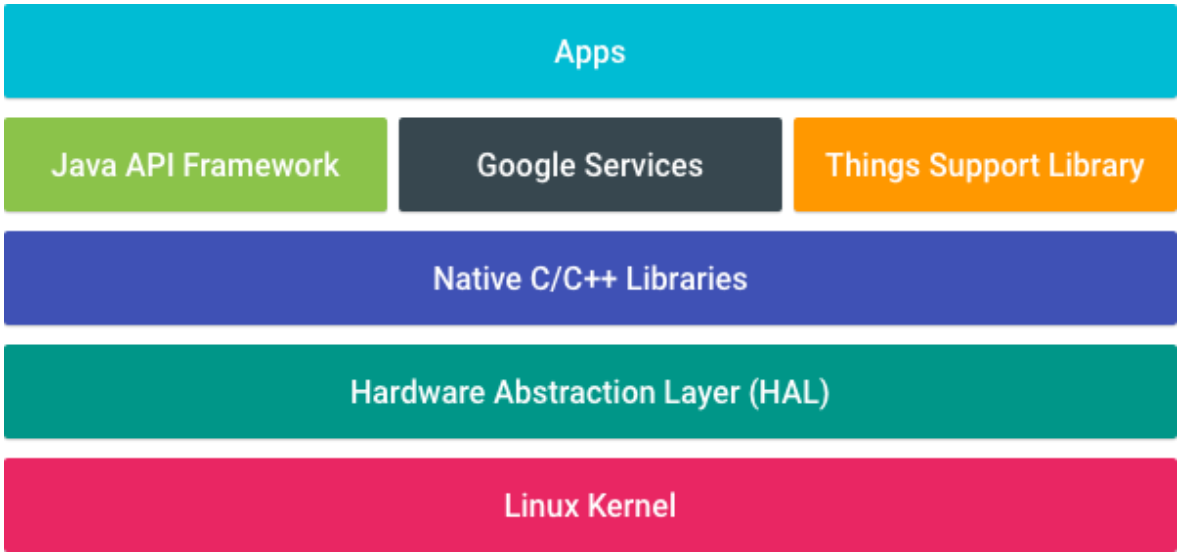
8. Android Things operacijski sustav

Android Things je operativni sustav baziran na Android platformi. Android Things, prvobitno nazvan projekt Brillo, prvi put je prezentiran na Google I/O konferenciji 2015. godine. Njegova namjena je proširiti korištenje Android platforme na uređaje za Internet stvari (engl. *Internet of Things*). Takvi uređaji obično rade na slabim procesorima ili mikrokontrolerima, te s ograničenom količinom radne memorije. Android Things iz tog razloga inicijalno dolazi s jezgrom Android razvojnog okruženja koja je smanjena za oko 50%. Takvo drastično smanjenje količine koda omogućuje Android Things operacijskom sustavu rad na uređajima s 32 ili 64 MB radne memorije. Povezivanje drugih uređaja, putem raznih perifernih priključaka, izrazito je bitno kod razvoja Internet stvari, stoga Android Things nudi podršku za Bluetooth Low Energy, te Wi-Fi bežični prijenos podataka (Amadeo, 2015).

Razvoj aplikacija za Android Things funkcionira na sličan način kao i razvoj mobilnih aplikacija za Android platformu. Za samo programiranje koristi se Java programski jezik kroz Google-ov besplatni IDE (engl. *Integrated Development Enviroment*) Android Studio. Jedina razlika u odnosu na razvoj mobilnih aplikacija je uvođenje Things Support biblioteke. Biblioteka pruža različita programska sučelja za pristup vanjskim uređajima, senzorima i upravljačkim programima koji inače nisu prisutni na mobilnim telefonima. Android Things jednom instaliranu aplikaciju automatski pokreće tijekom svakog pokretanja operacijskog sustava. Ako se aplikacija pokrene na Raspberry Pi uređaju, operacijski sustav automatski pokreće aplikaciju i spaja uređaj na Internet putem Wi-Fi veze.

Za razliku od Raspbian operacijskog sustava, Android Things nema podršku za grafičko sučelje slično onome na Linux i Windows operacijskim sustavima. Sve što se korisniku prezentira kod prvog pokretanja operacijskog sustava je početni ekran dobrodošlice, te status Wi-Fi i Ethernet konekcija. Aplikacije koje se pokreću mogu imati jedan ekran, odnosno aktivnost, te na njoj sve komponente slične onima koje je moguće naći na mobilnim aplikacijama. Takve komponente uključuju razne tekstualne kontrole, tipke, prikaze i sl.

Razvoj aplikacija na Android Things platformi olakšava rad onim osobama koje već imaju nekog iskustva u razvoju Android mobilnih aplikacija. Takve aplikacije koriste Java programski jezik i većina programera za razvoj koristi Android Studio IDE. Android se može vrlo jednostavno povezati na uređaj na kojemu se izvršava aplikacija u Android Things operacijskom sustavu. Sve što je potrebno je pozvati Android Debug Brigde naredbu: *adb connect* s točnom IP adresom uređaja na kojemu je aplikacija pokrenuta. U tom trenutku Android Studio IDE dobiva pristup takvoj aplikaciji. Daljnji razvoj aplikacije oslanja se na korištenje Android Things Support biblioteke. Kao što je prikazano na slici Sl. 8.1 Things biblioteka se oslanja na već postojeće Android razvojno okruženje. Komunikacija s različitim vanjskim senzorima i aktuatorima omogućena je kroz Peripheral I/O API koji koriste GPIO, I2C, SPI, te UART kao standardne protokole i sučelja za komunikaciju. Ako se operacijski sustav ne može povezati na određeni uređaj, osigurana je podrška za dodavanje novih upravljačkih programa kroz User Driver API. Aplikacija može pristupiti nativnom sklopovlju na uređaju kroz podršku za pisanje C++ koda koristeći Android NDK (engl. *Native Development Kit*) (Google, 2018).



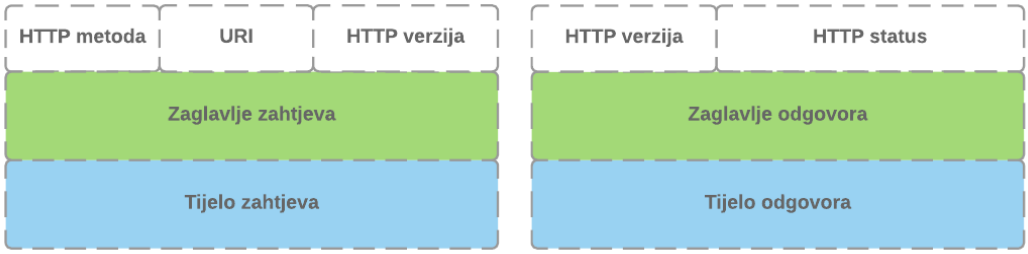
Sl. 8.1 Things Support biblioteka

(izvor: https://developer.android.com/things/get-started/)

9. REST Web servis

Web servis je program koji se izvršava na serveru i moguće mu je pristupiti preko Interneta. Njegova uloga je omogućiti komunikaciju između nekoliko aplikacija koje mogu biti napisane u različitim programskim jezicima i koje se mogu izvršavati na različitim uređajima, u različitim programskim okruženjima i na raznim operacijskim sustavima. Interoperabilnost između takvih aplikacija moguće je postići samo kroz predefinirane i standardizirane protokole, te formate za razmjenu podataka. Web servisi se generalno dijele u dvije kategorije: SOAP i REST. REST generalno podrazumijeva arhitekturu izrade web servisa, dok SOAP predstavlja protokol za razmjenu informacija. Premda se web protokol i web arhitektura ne mogu direktno usporediti, postoje bitne razlike između implementacija web servisa kroz obje tehnologije. Glavna razlika se očituje u stupnju povezanosti između klijenta i servera. SOAP protokol zahtjeva strogu poveznicu između klijenta i servera koja onemogućava fleksibilne izmjene na serveru bez ažuriranja klijenta i obrnuto. S druge strane, REST se oslanja na princip *stateless* komunikacije, odnosno komunikacije koja je neovisna o bilo kakvom stanju na serveru (Werneck, 2017). Druga glavna razlika se očituje u tome što SOAP servisi koriste XML format za razmjenu poruka, dok se REST oslanja na HTTP protokol i JSON format razmjene poruka. XML je puno robusniji i ima više mogućnosti opisa strukture i tipova podataka, dok je JSON kompaktan i brz format za čitanje kojeg je jednostavno integrirati s raznim klijentskim *JavaScript* web aplikacijama (Kling, 2016). Implementacija web servisa kroz SOAP protokol stavlja fokus na operacije nad podatcima, omogućava ACID transakcije, te je generalno namijenjen za poslovne sustave i njihove aplikacije (Wodehouse, 2018). U realizaciji diplomskog rada korištena je REST arhitektura za izradu web servisa. REST nije strogo vezan uz klijenta koji koristi web servis, već njemu samo prezentira API sučelja, odnosno URI reference za pristup raznim resursima. *URI* reference predstavljaju tekstualne adrese koje pružaju identifikaciju različitih resursa koji su prisutni na web servisu.

Komunikacija između klijenta i servisa započinje iniciranjem zahtjeva od strane klijenta. Servis tada obrađuje zahtjev i šalje odgovor klijentu. Na slici Sl. 9.1 prikazan je format HTTP *request* i *response* poruka.



Sl. 9.1 HTTP *request* i *response*

(izvor: autor)

Zahtjev se sastoji od HTTP metoda: GET, PUT, POST, DELETE, verzije HTTP protokola, te URI identifikatora. Zaglavlje poruke sadrži meta-podatke u *key-value* parovima koje služe za dodatno tijela HTTP poruke. Koristeći standardne HTTP metode klijentska aplikacija može slati zahtjeve prema web servisu. Svaka od metoda omogućuje uniformno sučelje za CRUD operacije, odnosno akcije dohvaćanja, dodavanja, uređivanja ili brisanja podataka. U tijelu poruke nalaze se podatci koje klijentska aplikacija želi poslati prema servisu. Nakon obrade zahtjeva servis vraća odgovor (engl. *response*), te HTTP status odgovora. Status je obično 3-znamenkasti kod koji pruža informacije o tome da li je zahtjev uspješno obrađen, te da li ga je server u potpunosti razumio. Da bi servis bio što fleksibilniji i lakši za održavanje, te imao dobre performanse nužno je da bude *stateless*, odnosno neovisan o bilo kakvom stanju na serveru. Takav servis podrazumijeva da svi zahtjevi koji se obrađuju imaju sve potrebne podatke unutar zaglavlja ili tijela HTTP poruke i da nisu ovisni o bilo kakvom kontekstu ili stanju na servisu. Na taj način izbjegava se potreba za sinkronizacijom podataka iz različitih sesija, servis je jednostavno testirati, te smanjiti opterećenje koje dolazi s povećanim brojem upita tokom vremena. REST servis stvara mogućnost brzog i fleksibilnog prezentiranja podataka raznim aplikacijama kroz standardni JSON format za razmjenu poruka (Vaqqas, 2014). Ovakav način komunikacije omogućuje modernim *JavaScript* razvojnim okruženjima, poput onog opisanog u realizaciji diplomskog rada, jednostavan pristup podatcima koje tada mogu prezentirati korisniku kroz sučelje web aplikacije.

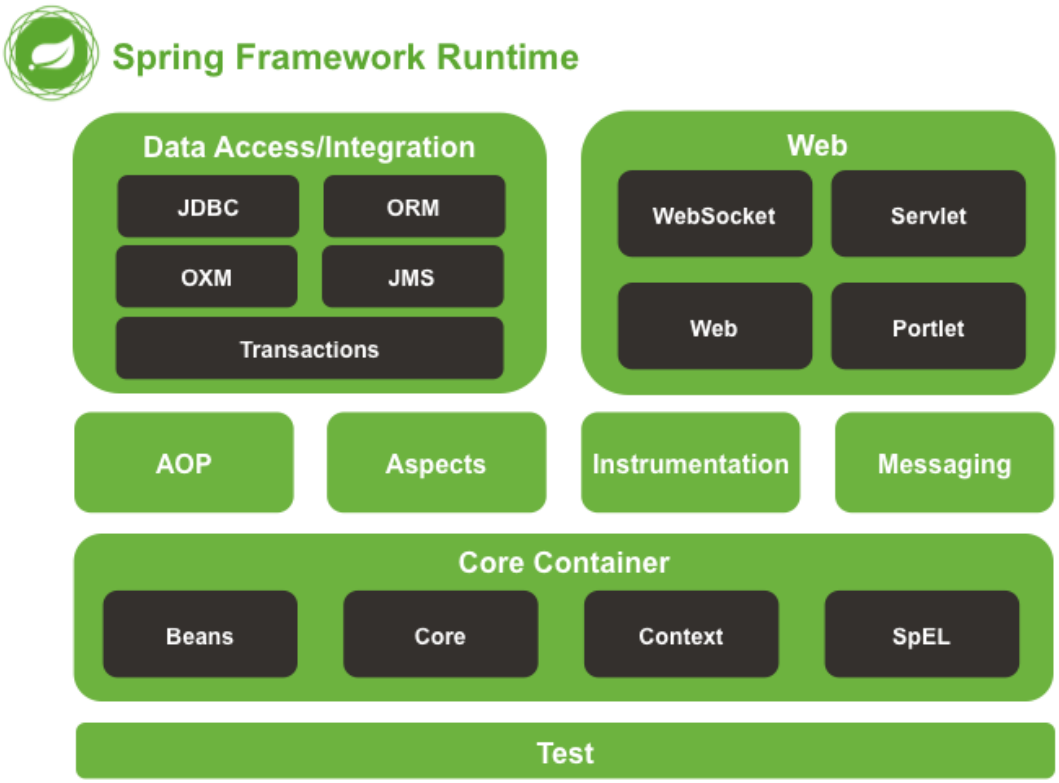
9.1. Spring web servis

REST web servis je najbolje implementirati koristeći neki od gotovih web razvojnih okruženja. Svaki se od razvojnih okruženja dostupnih na Internetu bazira na nekom programskom jeziku: Python, Ruby, PHP, JavaScript, Java, C# i sl. Osobne preference i nivo poznavanja određenog programskog jezika bitno utječu na odabir razvojnog okruženja. Kod izrade diplomskog rada odabrana je Java kao baza za izradu REST web servisa. Koristeći Javu kao programski jezik, na Internetu se mogu pronaći nekoliko web razvojnih okruženja: Spark, Play, Spring i sl. Spark je razvojno okruženje otvorenog koda, koje se bazira na Jetty web serveru. Trenutna verzija podržava Javu i Kotlin kao jezike za razvoj, te se bazira na MVC obrascu dizajna. Upravo zbog svoje malene veličine, razvojno okruženje je jednostavno za korištenje, fleksibilno i idealno za malene web aplikacije ili REST API-je. Vrlo jednostavno se može proširiti novim bibliotekama, pa je tako moguće umjesto zadanog Jetty web servisa koristiti nešto drugo. S druge strane Spark zbog svoje veličine nije idealno rješenje za sve situacije. Nedostaje mu robusnost i mnoštvo funkcionalnosti koje pruža puno veće razvojno okruženje kao Spring. Ali unatoč tome zbog svoje jednostavnosti korištenja i dalje uživa popularnost među Java programerima (Shelajev, 2016).

Play je razvojno okruženje otvorenog koda koje je prati MVC obrazac dizajna. Napisan je u Scala programskom jeziku, ali ga je moguće koristiti s Javom. U mnogo slučajeva uspoređuje se izravno s Spring-om zbog njihove veličine i količine funkcionalnosti koje oba razvojna okruženja nude. Play dolazi s uključenim Akka ili Netty web serverom koji omogućuje obavljanje asinkronih ulazno/izlaznih operacija. Nativno je podržana *stateless* REST API komunikacija, što znači da *framework* ne čuva informacije o sesiji kod svake konekcije. Briga o vanjskim bibliotekama i izrada aplikacije prepuštena je *Simple Build Tool*-u, Scala alatu za automatizaciju izrade. Povezivanje na bazu podataka i apstrakcija podataka omogućena je kroz objektno-relacijski maper *Eban*. Play ima podršku za API validacije s anotacijama, testiranje koda kroz *JUnit* biblioteku, injekciju ovisnosti koristeći *Guice* biblioteku, te mogućnost *hot reloadinga* tijekom programiranja (Play, 2018). Play ima vrlo dobru dokumentaciju i veliku zajednicu korisnika na Internetu s kojom predstavlja pravu konkurenciju Spring razvojnom

okruženju. Premda je Play moguće koristiti s Java programskim jezikom, većina materijala i referenci na Internetu preferira Play u sklopu programiranja s Scala programskim jezikom. Upravo zato je u realizaciji diplomskog rada odabran Spring kao razvojno okruženje za razvoj web servisa.

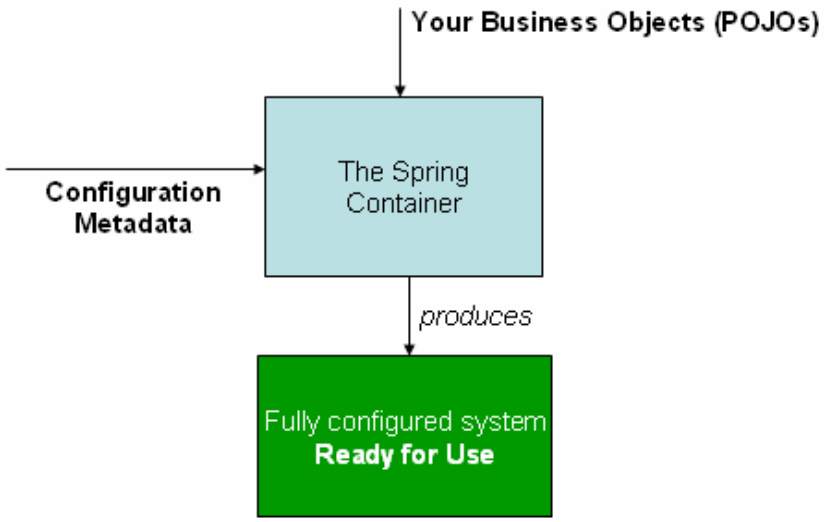
Spring je Java razvojno okruženje otvorenog koda u vlasništvu tvrtke Pivotal. Spring nastoji pomoći u strukturiranju Java aplikacije kroz značajke kao *container* inverzije kontrole, aspektno orijentirano programiranje, integraciju s bazama podataka i REST web servisima. Ove ključne komponente Springa, koje će biti objašnjene u nastavku poglavlja, omogućuju da arhitektura aplikacije bude čim više slojevita i modularna, te da kod bude što više odvojen i neovisan od ostatka aplikacije (Chand, 2017). Kao što je prikazano na slici Spring se sastoji od nekoliko kolekcija modula: *Core Container*, *Integration/Data Access*, *Web*, *AOP, Instrumentation, Messaging* i *Test*.



Sl. 9.2 Arhitektura Spring razvojnog okruženja

(izvor: https://docs.spring.io/spring/docs/5.0.0.RC2/spring-framework-reference/overview.html)

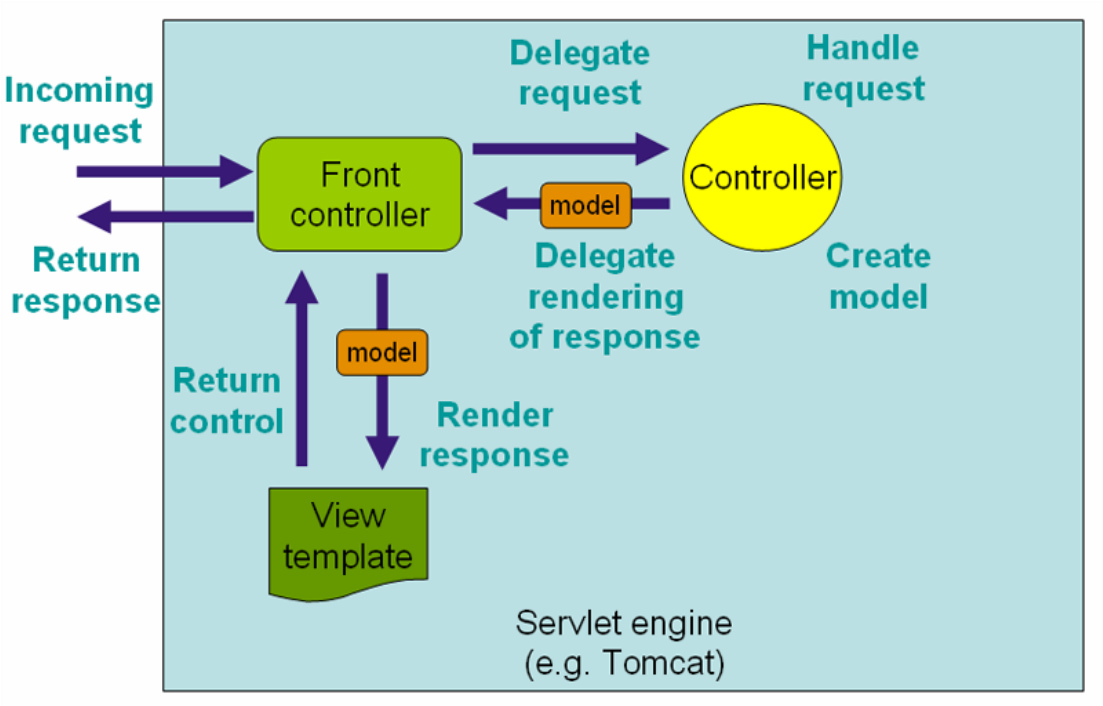
Core Container kolekcija sadrži glavne module razvojnog okruženja: *spring-core* i *spring-beans*. Ovi moduli omogućuju glavnu značajku razvojnog okruženja: *container* inverzije kontrole, odnosno injekciju ovisnosti. To je proces u kojem objekti sami definiraju svoje ovisnosti prema drugim objektima, tj. prema onim objektima o kojima ovise. Spring *container* tada ubacuje (engl. *injects*) objekte kada kreira pojedini *bean*. *Bean* je tada sam zadužen za kontrolu instanciranja svojih ovisnosti, pa stoga i naziv inverzija kontrole. U Springu se objekti o kojima se brine *container* inverzije kontrole nazivaju *beans*. *Bean* predstavlja samo jedan od nekoliko objekata koji tvore srž aplikacije. Svi *beanovi* i njihove međusobne ovisnosti se odražavaju u konfiguracijskim meta-podatcima koje koristi Spring *container*. Takav *container* je u Springu definiran kroz *ApplicationContext* sučelje. *Container* na temelju konfiguracijskih meta-podataka odlučuje koje će objekte kreirati, te kako će ih konfigurirati. Meta-podatci su obično prezentirani kroz XML konfiguracijske datoteke, Java anotacije ili Java kod. Na slici Sl. 9.3 je prikazan Spring *IoC container* koji, koristeći aplikacijske klase i konfiguracijske meta-podatke, kreira aplikaciju koja je spremna za korištenje (Spring, 2018).



Sl. 9.3 Spring *container* inverzije kontrole

(izvor: https://docs.spring.io/spring/docs/5.0.0.RC2/spring-framework-reference/core.html)

Implementacija REST web servisa je moguća kroz *spring-web* i *spring-webmvc* servlet module. Ovi moduli čine web razvojno okruženje bazirano na MVC (engl. *model-view-controller*) arhitekturi. Spring MVC je realiziran oko centralnog servlet objekta koji se brine o svim HTTP zahtjevima na temelju kojih kreira odgovarajući HTTP odgovor. Na slici Sl. 9.4 prikazano je procesiranje HTTP zahtjeva kroz Spring MVC razvojno okruženje. Spring MVC je u potpunosti integriran s *IoC container*-om, te kao takav nudi sve mogućnosti Spring razvojnog okruženja (Spring, 2018).



Sl. 9.4 Spring MVC procesiranje HTTP zahtjeva

(izvor: https://docs.spring.io/spring/docs/5.0.0.RC2/spring-framework-reference/web.html)

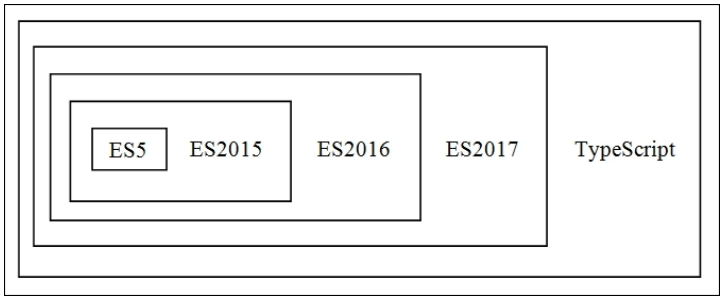
Spring uključuje podršku za JDBC konekcije koja olakšava povezivanje i izvršavanje upita prema bazi podataka, te pruža sloj apstrakcije koji olakšava s upravljanjem transakcija prema bazi podataka. Objektno-relacijsko mapiranje moguće je korištenjem *spring-orm* modula koji pruža integraciju s JPA i *Hibernate* objektno-relacijskim maperima. Na taj način korisnik Spring razvojnog okruženja može na jednostavan način pretočiti relacije i entitete iz baze podataka u POJO (engl. *Plain Old Java Objects*) objekte. Testiranje programskog koda i poslovne logike moguće je kroz *spring-test* modul koji uključuje podršku za *unit* i integracijske testove (Spring, 2018).

10. MySQL baza podataka

Mogućnost perzistiranja podataka, odnosno trajnog pohranjivanja na disku servera nije moguće koristeći samo java web servise. U rješavanje takvog problema potrebno je uključiti bazu podataka, čija uloga je rukovanje i spremanje različitih kolekcija informacija. Sustavi za upravljanje bazama podataka dolaze u različitim konfiguracijama i načinima rada. Oni se prvenstveno baziraju na raznim podatkovnim modelima koji se koriste da bi se stvorila logična struktura samih podataka. Najpoznatiji takav model naziva se relacijski model koji služi da bi se uz pomoć relacija opisale veze među podatcima. Podatci se čuvaju u tablicama koje predstavljaju entitete, a opisuju ih njihovi atributi. Ako relacije među podatcima nisu potrebne i podatke je moguće pohraniti nepovezano i slučajno, tada je bolje koristiti NoSQL baze podataka. NoSQL baze ne koriste relacije i nastoje što više maknuti ograničenja s načina na koji se podatci formiraju. Podatci su dostupni kao jedna kolekcija koja predstavlja čitavi podatkovni objekt. Relacijske baze su iznimno pouzdane i efikasne, te kroz vrlo strogi, matematički pristup strukturiranja podataka garantiranju njegovu jednostavnost i točnost (Tezer, 2014). U realizaciji aplikacije opisane u diplomskom radu korištena je relacijska baza MySQL. MySQL je jedna od najpoznatijih i najpopularnijih baza podataka otvorenog koda. U vlasništvu je tvrtke Oracle, te se može koristiti na većini operacijskih sustava: Window, Mac I Linux, te ima podršku za povezivanje s većinom programskih jezika: Java, C++, Python, PH, Node.JS i sl. SQL relacijske baze zahtijevaju definiranje sheme podataka koja predstavlja čitavu strukturu podataka. Tek nakon definiranja strukture podatci mogu početi pohranjivati u bazu podataka. To znači da svi podatci moraju pratiti istu predefiniranu strukturu, te svaka nova izmjena u strukturi zahtjeva ponovno promišljanje čitave sheme podataka. NoSQL baza podataka ima dinamičnu shemu ne strukturiranih kolekcija podataka. Nema potrebe za unaprijed definiranjem strukture podataka, svaki dokument može imati svoju strukturu, moguće je dodavati nove atribute i polja po potrebi. Mana ovakvih baza sintaksa za pristup podatcima, koja može varirati od baze do baze. Podatci mogu biti složeni u kolone, kao jedan dokument ili po principu *key-value* (Xplenty, 2017).

11. Angular web razvojni sustav

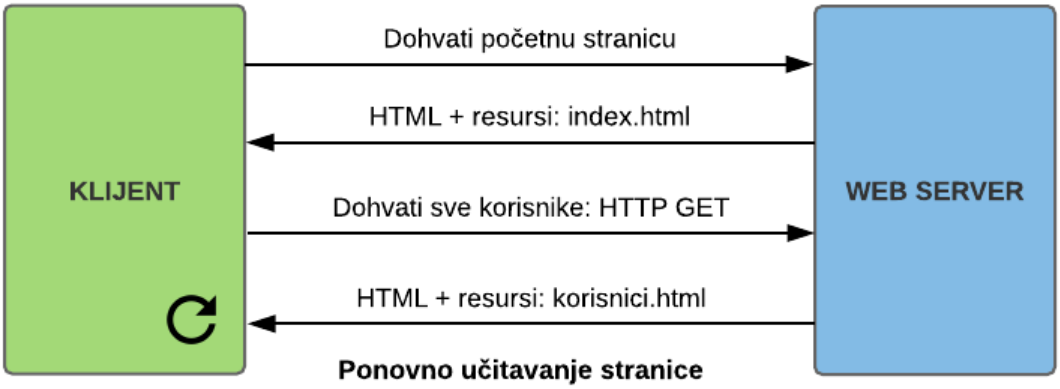
Angular je web platforma otvorenog koda (engl. web *framework*) namijenjena za izradu interaktivnih *single page* web aplikacija. Na razvoju i održavanju Angular *frameworka* radi tvrtka Google koja, uz veliku zajednicu programera na Internetu, pomaže u osiguravanju kvalitetne i dugoročne podrške svim sadašnjim i budućim korisnicima ovog Angular *frameworka*. Angular je napisan u programskom jeziku Typescript, na čijem razvoju radi tvrtka Microsoft. Typescript je programski jezik otvorenog koda koji predstavlja sintaktički nadskup JavaScript jezika. JavaScript, uz HTML i CSS, čini jednu od tri ključne tehnologije korištene u izradi današnjih web aplikacija. Uloga JavaScripta je prvenstveno obogaćivanje ponašanja web aplikacije kroz bolju interakciju između korisnika i same aplikacije. Osim brojnih prednosti, JavaScript ima i određene mane. Jedna od mana je nedostatak statične provjere tipova, te različitih mehanizama objektno orijentiranih jezika koje omogućuju korištenje klasa, nasljeđivanja, sučelja i sl. Upravo su takvi nedostatci doveli su do nastanka TypeScript programskog jezika. TypeScript pomaže u razvoju poslovnih web aplikacija svih nedostataka koje se mogu naći u JavaScriptu. Dodana je statična provjera tipova, klase, moduli, generički tipovi, enumeratori, sučelja i sl. Da bi svi Internet preglednici mogli razumjeti TypeScript kod potrebno ga je provesti kroz jezični prevoditelj (engl. *compiler*). Prevoditelj TypeScript kod pretvara u JavaScript i tokom tog procesa pomaže programeru u otkrivanju pogrešaka kroz kod koristeći provjeru tipova i sintakse u samom trenutku prevođenja. Slika Sl. 11.1 prikazuje sintaktički nadskup TypeScript programskog jezika naspram različitih verzija JavaScript-a.



Sl. 11.1 Prikaz nadskupa TypeScript programskog jezika

(izvor: Web development with Bootstrap 4 and Angular 2, Radford)

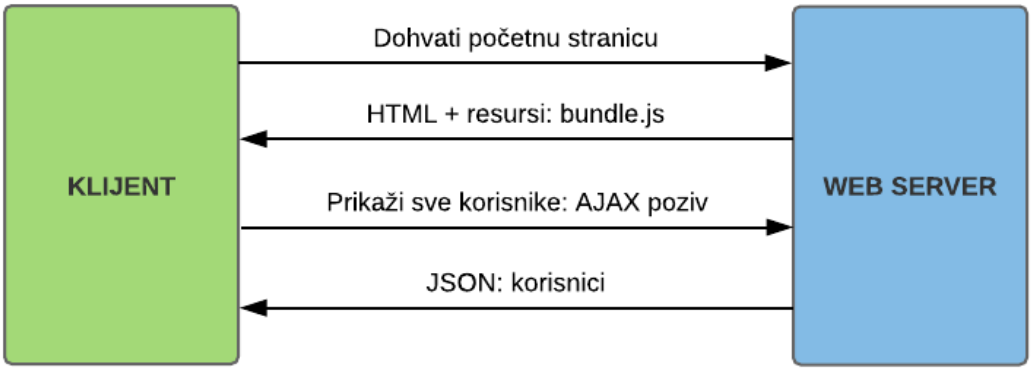
Glavni razlog korištenja Angular *frameworka*, kod izrade diplomskog rada, jest pokušaj implementacije novog načina izrade web aplikacija. Naime, standardne *round trip* web aplikacije prilikom interakcije s korisnikom uvijek iznova šalju zahtjeve prema serveru za novim HTML dokumentom. To znači da svaki klik na sučelju aplikacije kreira novi upit prema web serveru. Prilikom odgovora servera preglednik mora svaki put iznova renderirati čitavi HTML dokument da bi osvježio sučelje korisniku aplikacije. Takav dokument tada mora u sebi sadržavati čitavu poslovnu logiku i sve podatke potrebne za ispravno prikazivanje web aplikacije. Ovaj proces ne zahtjeva pisanje poslovne logike na klijentu, odnosno web pregledniku, ali zauzvrat traži od klijenta čekanje dok se HTML dokument preuzme i prikaže na sučelju. Dobar primjer predstavljaju gotovo sve web aplikacije pisane u programskom jeziku PHP. Slika Sl. 11.2 prikazuje komunikaciju *round trip* web aplikacije.



Sl. 11.2 *Round trip* web aplikacija

(izvor: autor)

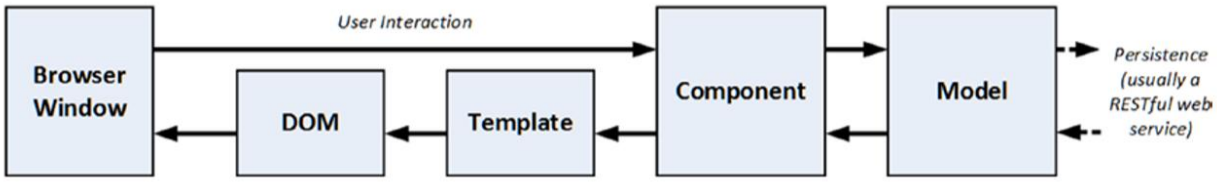
Kao što je prikazano na slici Sl. 11.3 *single page* aplikacije rade na drugačiji način. Prilikom inicijalnog učitavanja web aplikacije šalje se zahtjev za HTML dokumentom. Nakon što je dokument dostavljen on se više ne učitava ponovno, već prilikom interakcije s korisnikom aplikacija osvježava samo onaj dio dokumenta koji je relevantan korisniku. Svi se zahtjevi prema serveru obavljaju asinkrono kroz AJAX pozive što znači da korisnik može nastaviti s radom na aplikaciji dok preglednik ne dobije odgovor od web servera. Na taj način korisnik dobiva brzo, responzivno i fluidno sučelje, te iskustvo slično onome na *desktop* aplikacijama.



Sl. 11.3 *Single page* web aplikacija

(izvor: autor)

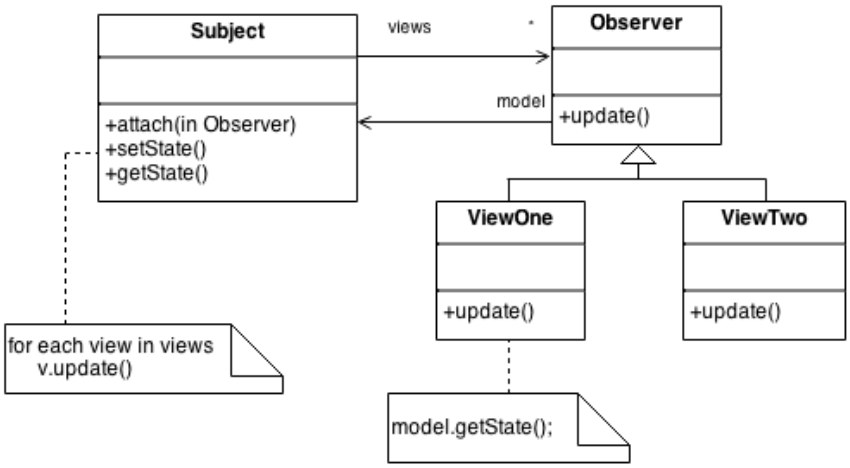
Arhitektura izrade Angular aplikacija može se opisati kroz MVC (engl. *Model View Controller*) oblikovni obrazac. Glavna ideja korištenja MVC obrasca podrazumijeva razdvajanje dužnosti raznih dijelova web aplikacije. U takvom razdvajanju zamisao je razdvojiti podatkovni model od implementacije poslovne logike i prezentacijskog dijela web aplikacije. Prezentacijski dio u slučaju web aplikacije podrazumijeva HTML i CSS elemente koji služe za prikaz podataka na sučelju web preglednika. Kao što je prikazano na slici Sl. 11.4 Angular razvojno okruženje obično generira podatkovni model na temelju REST web servisa. On vrši komunikaciju s bazom podataka i zadužen je za perzistiranje podataka na serveru. Komponente i predlošci (engl. *template*) obavljaju operacije nad modelom podataka i na temelju njih vode brigu o DOM-u i HTML elementima s kojim korisnik stupa u interakciju. Razne akcije korisnika tada putuju natrag prema komponentama, te na taj način zatvaraju petlju toka podataka prema modelu. Naposljetku, MVC je samo dogovoreni stil izrade web aplikacija. Web aplikacije ga nisu dužne poštivati, ali on nastoji pomoći u olakšavanju održavanja i testiranja, omogućiti brže dodavanje novih komponenti, te smanjiti broj mogućih grešaka tijekom razvoja web aplikacije (Freeman, 2017).



Sl. 11.4 Angular MVC oblikovni obrazac

(izvor: Pro Angular – Adam Freeman)

Kao što je prikazano na slici Sl. 11.3 komunikacija između Angular web aplikacije i REST web servisa odvija se asinkrono putem AJAX poziva. Angular može AJAX pozive obavljati korištenjem *Promise* ili *Observable* implementacija. *Promise* implementacija radi s jednim događajem koji signalizira kada se asinkrona operacija završi uspješno ili neuspješno. Observable implementacija koristi RxJS (engl. *Reactive Extensions for JavaScript*) biblioteku. RxJS je biblioteka za reaktivno programiranje koju je izradila tvrtka Microsoft. Pod pojmom reaktivnog programiranja podrazumijeva se korištenje *Observable* tipova podataka koji pomažu u upravljanju asinkronim upitima i podatcima, kao što su oni koji na web aplikaciju pristižu od strane web servisa. RxJS biblioteka pruža implementaciju za *Observable* tipove korištenjem observer oblikovnog obrasca. Oblikovni obrazac nalaže da jedan subjekt može imati listu ovisnih objekata, koji se nazivaju promatrači (engl. *observers*). Subjekt tada obavještava svakog promatrača ako u nekom trenutku promjeni stanje (Zöchbauer, 2017). Na slici Sl. 11.5 prikazan je dijagram *observer* oblikovnog obrasca.



Sl. 11.5 Observer oblikovni obrazac

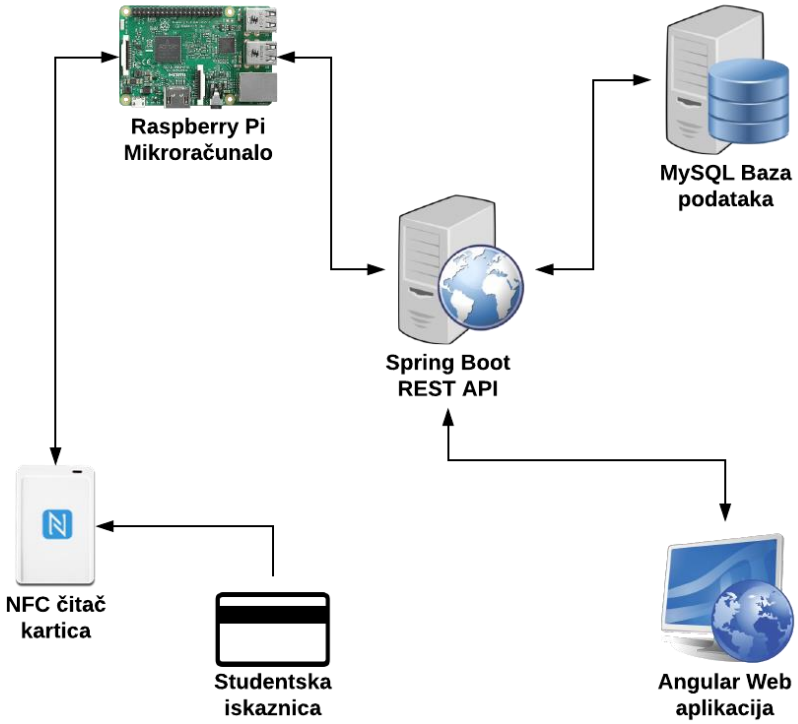
(izvor: https://sourcemaking.com/design_patterns/observer)

12. Arhitektura sustava za evidenciju

Osnovna funkcionalnost sustava za evidenciju je evidentiranje studenata koji su prisutni na predavanjima. Da bi se student uspješno evidentirao potrebno ga je povezati s njegovim kolegijima i vremenu izvođenja njihovih predavanja. Takva poveznica je omogućena kroz web aplikaciju kojoj visoko učilište pristupa putem Interneta. Web aplikaciju mogu koristiti studenti, predavači i djelatnici visokog učilišta s ulogom administratora. Studenti imaju mogućnost pregledavanja upisanih kolegija i popisa evidencija koji prikazuju kada su bili prisutni na predavanju. Predavači imaju mogućnost praćenja i kontrole prisutnih studenata za sve kolegije na kojima imaju ulogu nositelja ili asistenta. Predavači na taj način mogu na vrijeme upozoriti studente s previše izostanaka na pojedinim kolegijima.

Sustav za evidenciju funkcionira na način da nastoji u svakom trenutku raspolagati s što točnijim informacijama vezanim uz predavanja vlastitih studenata. Upravo iz tog razloga visoko učilište treba, prije početka akademske godine, unijeti matične podatke o studentima, predavačima, predavanjima i njihovim kolegijima kroz sučelje web aplikacije. Predavač tada, na osnovu unesenih kolegija, može započeti evidenciju prisutnih studenata na temelju mjesta i vremena izvođenja vlastitih predavanja. Predavač je ujedno i inicijator evidencije iz razloga što evidencija za studente započinje tek nakon što predavač prisloni vlastitu iskaznicu na čitač kartica. Na taj način sustav za evidenciju može vrlo jednostavno povezati predavača sa njegovim predavanjem i na temelju toga odrediti o kojem kolegiju se radi. Kada sustav raspolaže s informacijom o kojem se točno kolegiju radi može, sa visokom sigurnošću, evidentirati dolaske za sve prisutne studente.

Kod dizajniranja arhitekture sustava za evidenciju glavna ideja je mogućnost povezivanja sa drugim vanjskim sustavima. Kod takvog povezivanja se prvenstveno misli jednostavnu integraciju s ISVU i ISSP informacijskim sustavima. Informacijski sustavi imaju podršku za REST komunikaciju, pa je upravo iz tog razloga sustav za evidenciju realiziran na način u kojem koristi vlastiti REST web servis.



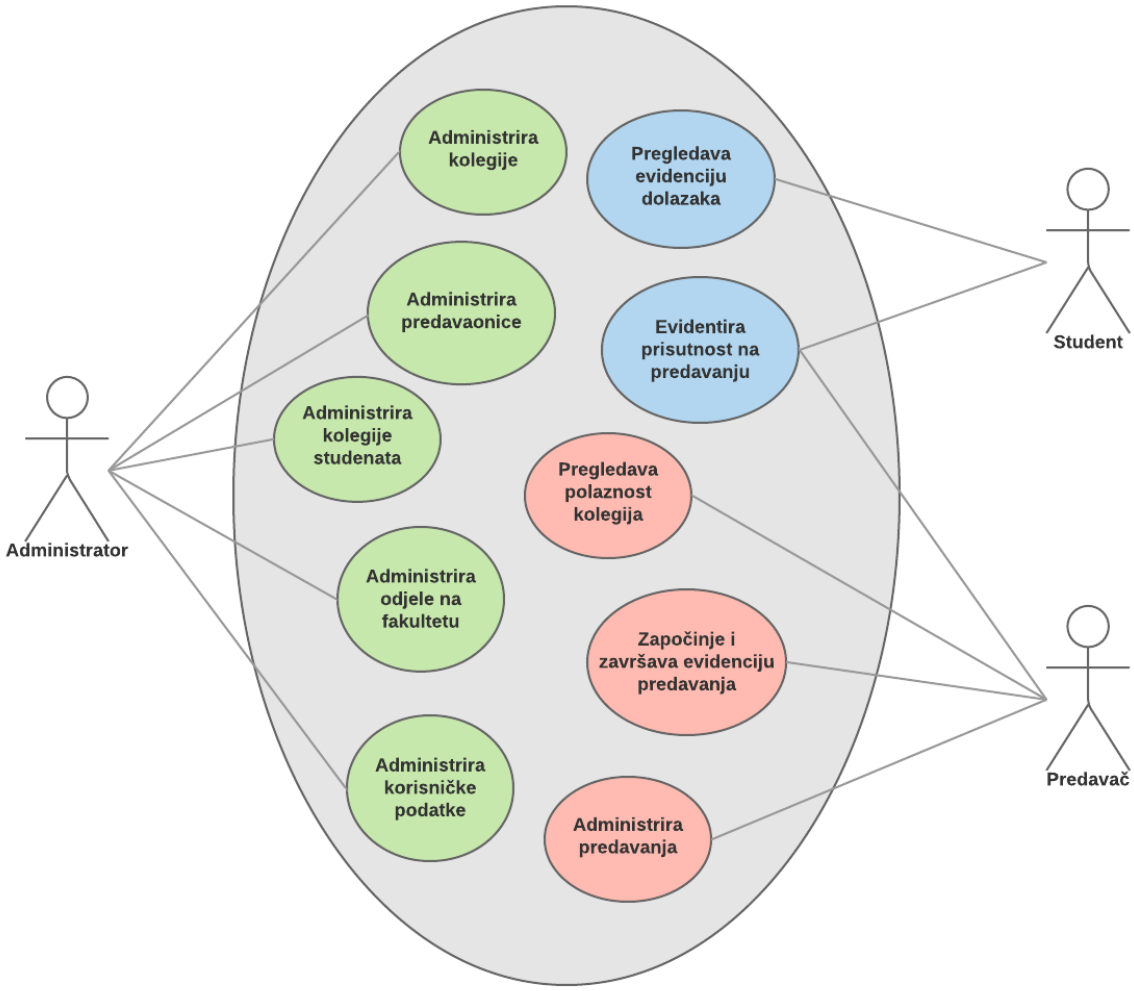
Sl. 12.1 Prikaz arhitekture sustava za evidenciju

(izvor: autor)

Web servis, koji je vidljiv na slici Sl. 12.1. ima ulogu središnje komponente sustava koja prikuplja podatke o svim aktivnostima koje obavljaju predavači i administratori preko web aplikacije. Web aplikacija koristi web servis da bi dodavala i uređivala podatke koji se nalaze u MySQL bazi podataka. Sve akcije prema web servisu prate HTTP protokol i njegove metode kroz JSON format. JSON služi da web aplikacija može jednostavno čitati i parsirati relevantne informacije i prikazivati ih na sučelju web aplikacije. Web aplikacija tada nema nikakvu poslovnu logiku, već samo služi da bi konzumirala podatke s web servisa. Na taj način je arhitektura čitavog sustava dovoljno fleksibilna i razdvojena da je podatcima moguće pristupiti s bilo kojeg mjesta i uređaja koji ima Internet preglednik. Sama evidencija započinje slanjem podataka s studentske iskaznice, koji putuju iz čitača kartica prema Raspberry Pi ugradbenom računalnom sustavu. Raspberry Pi tada podatke prosljeđuje prema web servisu koji ih validira, te obavlja novi unos u MySQL bazi podataka.

13. Implementacija sustava za evidenciju

Kao što je prikazano na slici Sl. 13.1 web aplikacija za evidenciju studenata podijeljena je kroz tri glavne uloge: administrator, student i predavač. Ulogu administratora mogu imati djelatnici visokog učilišta koji su zaduženi za administraciju podataka o studentima, njihovim kolegijima, predavačima i predavaonicama. Studentima je omogućeno evidentiranje i praćenje vlastite evidencije dolazaka, dok predavači mogu dodavati predavanja, pregledavati prisutnosti studenata na vlastitim kolegijima, te voditi računa o vremenu i mjestu izvođenja pojedinog predavanja.



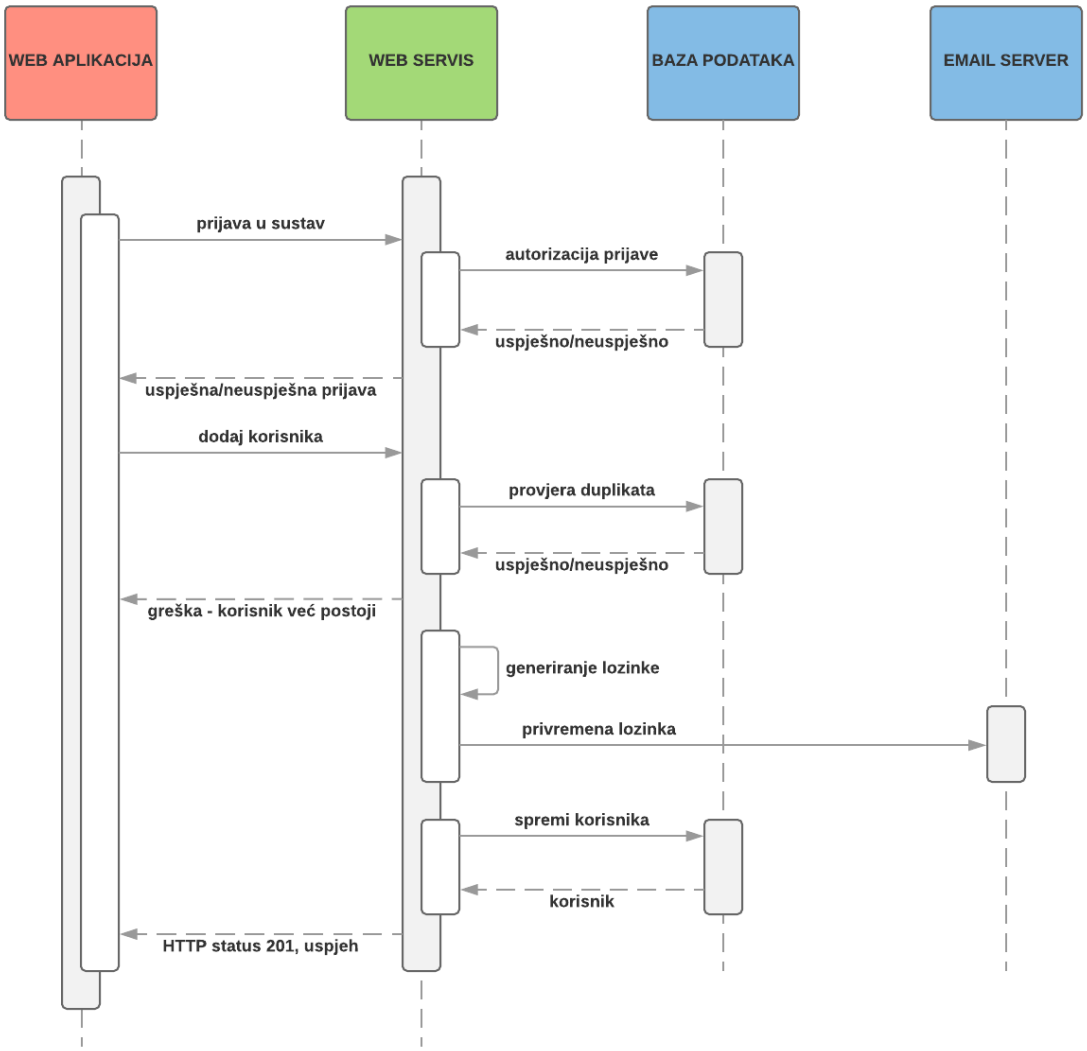
Sl. 13.1 Dijagram slučajeva korištenja

(izvor: autor)

Web aplikacija nema direktan pristup do baze podataka, već cijelo vrijeme komunicira i konzumira podatke s web servisa kroz JSON format za razmjenu podataka. Komunikacija se odvija korištenjem HTTP protokola i standardnih HTTP metoda: GET, PUT, POST i DELETE. Da bi REST web servisi pratili određena pravila i konvencije i bili što jednostavniji za korištenje potrebno se pridržavati određenih pravila. REST konvencija, prilikom pisanja web servisa, nalaže da sve URI adrese, odnosno URI rute budu odvojene prema određenim resursima s kojima takav web servis manipulira. Takvi se resursi, u slučaju sustava za evidenciju studenata, odnose na podatke o korisnicima, njihovim ulogama, studentima, predavačima, njihovim predavanjima, kolegijima, evidenciji dolazaka studenata, predavaonicama i odjelima. Konvencija nalaže da se CRUD operacije nad resursima moraju mapirati prema odgovarajućim HTTP metodama. Na taj način operacija čitanja nad resursom „*korisnici*“ mora biti mapirana na URI adresu koja počinje s „*users*“ i koja koristi HTTP GET metodu. Ako web servis ima mogućnost dohvaćanja pojedinog korisnika po jedinstvenom identifikatoru, tada URI adresa mora imati identifikator korisnika čiji podatci se žele pročitati. Operacija dodavanja novog korisnika mora pratiti HTTP POST metodu, dok operacija izmjene podataka korisnika mora koristiti HTTP PUT metodu. Operacija brisanja određenog korisnika mora biti mapirana na HTTP DELETE metodu koja, slično kao i dohvaćanje pojedinog korisnika, mora u parametru rute sadržavati identifikator korisnika čiji podatci se žele izbrisati. Da bi djelatnici visokog učilišta započeli s dodavanjem matičnih podataka o studentima, predavačima i korisnicima sustava nužno je da se u sustav prijave putem web aplikacije. Prijava u sustav funkcionira na način da web aplikacija prvo šalje HTTP POST zahtjev prema URI adresi */api/v1/authenticate* web servisa. U tijelu HTTP POST zahtjeva upisuje JSON poruku s korisničkim imenom i lozinkom korisnika. Prilikom primanja zahtjeva za autentifikaciju Spring Boot web servis mapira podatke iz JSON poruke u svoj interni klasni model, te započinje proces autentifikacije korisnika. Koristeći korisničko ime i lozinku koju je dobio od web aplikacije, web servis radi SQL upit prema bazi podataka u kojem provjerava da li takav korisnik postoji. Ako korisnik ne postoji, web servis vraća natrag HTTP odgovor s statusom *Unauthorized*. Ako je autorizacija uspješna i web servis je pronašao korisnika u bazi podataka, web aplikacija dobiva HTTP status *Success*.

Da bi studenti, predavači i administratori mogli započeti s korištenjem web aplikacije za evidenciju studenata potrebno ih je prvo dodati u sustav. Takve ovlasti ima korisnik s ulogom administratora koji jedini može dodavati nove korisnike. Sustav inicijalno dolazi konfiguriran s jednim korisničkim računom koji ima administratorska prava, ali korisnici mogu dodavati druge administratore prema vlastitim potrebama. Administrator, kao i svi korisnici sustava, treba prvo izvršiti prijavu u sustav da bi mogao dodavati nove korisnike. Nakon uspješne prijave u sustav, web aplikacija šalje HTTP GET zahtjev prema URI adresi */api/v1/users*. U tom trenutku web servis provjerava da li je zahtjev došao od korisnika koji ima prava pregledavati sve korisnike u sustavu i radi upit na bazu. Nakon uspješno izvršenog SQL upita, servis vraća listu korisnika u tijelu JSON poruke i HTTP status *Success*. Slanjem HTTP POST zahtjeva na URI adresu */api/v1/users* obavlja se dodavanje novog korisnika. Zahtjev mora u tijelu poruke sadržavati JSON model sa osnovnim podatcima kao što su korisničko ime, email adresa, šifra kartice i uloga korisnika. Nakon primanja i validacije zahtjeva web servis provjerava da li u sustavu već postoji korisnik sa istim korisničkim imenom. Ako korisnik već postoji web servis vraća HTTP status *Bad Request*. Ako korisnik s tim korisničkim imenom ne postoji sustav nastavlja dalje i generira privremenu lozinku za novog korisnika koju šalje na njegovu email adresu. Privremena lozinka se može iskoristiti samo kod prve prijave u web aplikaciju, nakon toga je korisnik dužan izmijeniti zbog sigurnosnih razloga. Nakon slanja privremene lozinke web servis sprema novog korisnika u bazu podataka i vraća HTTP status *Created* prema web aplikaciji. Slika Sl. 13.2 prikazuje sekvencijski dijagram koji opisuje proces dodavanja novog korisnika u sustav. Osim dodavanja, administrator ima mogućnost uređivanja podataka o postojećim korisnicima. Takav zahtjev na web aplikaciji prvo započinje slanjem HTTP GET zahtjeva koji kao parametar u URI adresu ima jedinstveni identifikator korisnika. Web servis tada dohvaća korisnika prema njegovom identifikatoru i vraća njegove podatke u JSON formatu prema web aplikaciji. Nakon što je administrator izmijenio podatke, web aplikacija šalje HTTP PUT zahtjev prema web servisu. PUT metoda signalizira web servisu da se radi o operaciji uređivanja podataka, pa web servis prilikom obrade zahtjeva provjerava da li su korisnički podatci ispravni, odnosno da li u sustavu postoji korisnik sa takvim jedinstvenim identifikatorom. Ako ne postoji, šalje se HTTP status *Bad Request* i operacija uređivanja podataka prestaje. Ako korisnik postoji

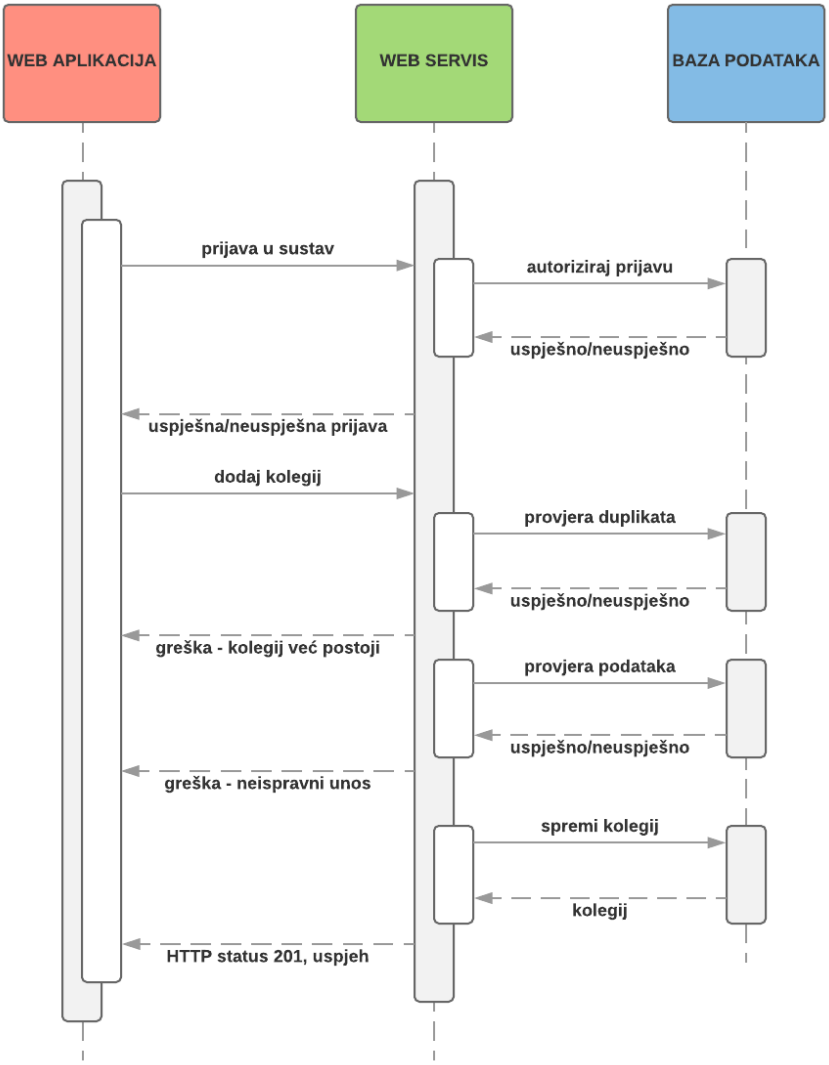
web servis prosljeđuje zahtjev i sprema podatke u bazu podataka. Nakon što su podatci uspješno spremljeni web servis vraća HTTP status *OK*. Operacija brisanja mora pratiti svoja pravila prema REST konvenciji, stoga web servis za takvu akciju šalje HTTP DELETE zahtjev prema web servisu. Akcija brisanja ne treba spremati podatke u tijelo poruke, već samo upisuje jedinstveni identifikator korisnika u parametre URI adrese. Nakon što web servis provjeri da li takav korisnik stvarno postoji u bazi podataka izvršava se akcija brisanja i vraća se HTTP status *OK* kao odgovor web aplikaciji.



Sl. 13.2 Dodavanje novog korisnika

(izvor: autor)

Osim korisničkih podataka administrator ima ovlasti pregledavati, dodavati, uređivati i brisati kolegije na visokom učilištu. Proces pregledavanje svih kolegija u sustavu započinje isto kao i kod pregleda svih korisničkih podataka. Web aplikacija šalje HTTP GET zahtjev prema web servisu koristeći URI adresu */api/v1/courses*. Nakon što je web servis autorizirano zahtjev koristeći *Basic Autentification* metodu i ustanovio da se radi o korisniku koji ima prava pregledavati sve kolegije u sustavu, web aplikaciji se vraća popis svih kolegija i HTTP status *OK*. Dodavanje novog kolegija funkcionira na sličan način kao i kod dodavanja novog korisnika, web aplikacija šalje HTTP POST zahtjev gdje u tijelu poruke upisuje JSON model s odgovarajućim podatcima za kolegij koji želi kreirati. Za svaki kolegij je bitno definirati nositelja i asistenta, te odjel kojemu taj kolegij pripada. Administrator odabire nositelja i asistenta na temelju popisa unesenih korisnika s ulogom predavača. Na taj način sustav definira jasnu poveznicu između kolegija i predavača. Ona služi da bi predavač, po dolasku na predavanje, mogao započeti evidenciju prisutnih studenata. Web servis i ovdje obavlja validaciju primljenih podataka, pa tako prije nego što spremi novi kolegij u bazu podataka, provjerava da li u sustavu već postoji kolegij s istim imenom, te da li postoje odabrani nositelj, asistent i odjel za taj kolegij. Ako postoji vraća HTTP status *Bad Request* i odbija zahtjev. Ako je validacija prošla u redu, web servis nastavlja s izvođenjem programa, sprema podatke u bazu i vraća odgovor da je operacija obavljena uspješno HTTP statusom *OK*. Uređivanje podataka o pojedinom kolegiju se obavlja na način da web aplikacija prvo radi HTTP GET zahtjev u kojem kao parametar upita šalje jedinstveni identifikator kolegija čije podatke želi uređivati. Nakon što administrator izmijeni željene podatke, web aplikacija šalje, prema REST konvenciji, HTTP PUT zahtjev. Web servis tada provjerava da li takav kolegij postoji u sustavu. Ako kolegij postoji, podatci se spremaju u bazu podataka i vraća je HTTP status *OK* prema web aplikaciji. Ako validacija nije prošla uspješno web servis odustaje od operacije spremanja podataka i vraća HTTP status 400 koji signalizira da zahtjev nije uspješno obrađen. Brisanje kolegija se odvija na isti način kao i kod resursa sa korisnicima. Web servis provjerava jedinstveni identifikator HTTP DELETE zahtjeva od web aplikacije, ako takav kolegij postoji, web servis ga trajno briše iz baze podataka i vraća odgovor da je operacija uspješno obavljena. Na slici Sl. 13.3 opisan je postupak dodavanja novog kolegija u sustav za evidenciju studenata.

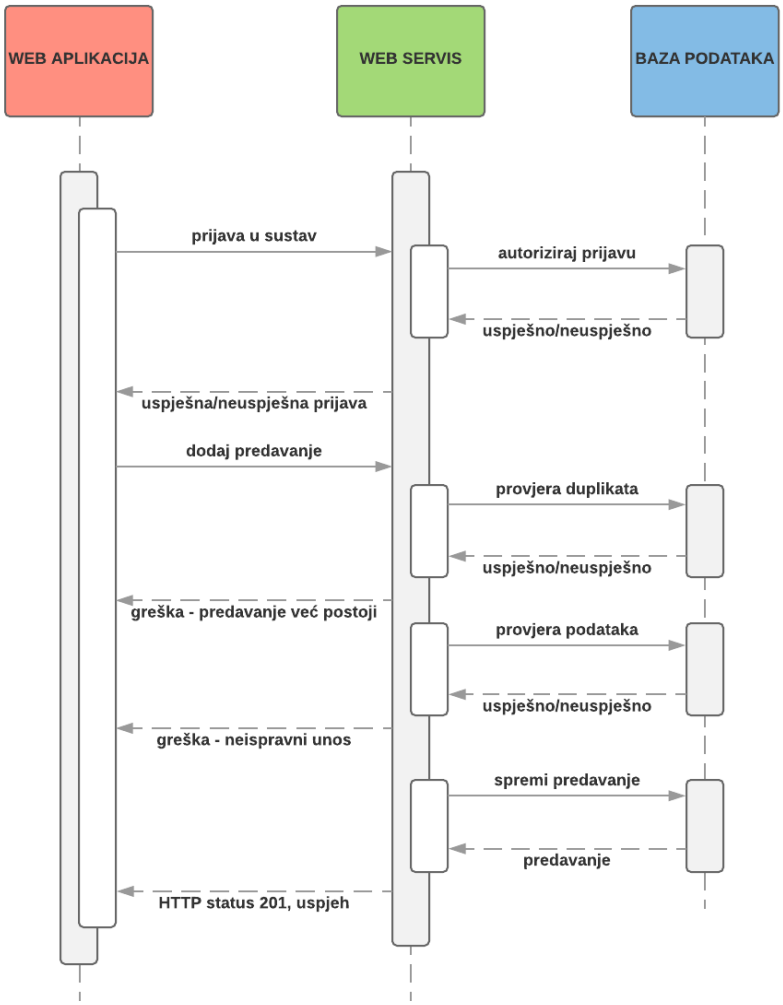


Sl. 13.3 Dodavanje novog kolegija

(izvor: autor)

Da bi se studenti mogli što preciznije i točnije evidentirati potrebno je povezati kolegije i njihova predavanja. Predavanja za tekuću akademsku godinu definira nositelj kolegija koji u sustavu evidencije ima ulogu predavača. Svako predavanje treba imati definirano vrijeme i mjesto izvođenja, kolegij, predavaonicu u kojoj se izvodi, te definiran status predavanja. Status predavanja govori o tome da li je predavanje započelo ili završilo, te da li je otvoreno za evidenciju dolazaka. Predavač također ima i ovlasti promijeniti status predavanja u završeno i time onemogućiti evidenciju studentima koji nisu došli na vrijeme. Neovisno o tome sustav ima određeni level automatike, pa tako evidentira dolazak studenta unutar deset minuta od vremena kad je predavač započeo evidenciju dolazaka.

Pregled svih predavanja u sustavu za evidenciju studenata započinje slanjem HTTP GET upita na web servis s URI adresom: */api/v1/lectures*. Ako je korisnik autoriziran kroz *Basic Autentification* metodu web servis radi SQL upit te odgovara s listom svih predavanja koji su upisani u bazi podataka. Dodavanje novog predavanja zahtjeva od web aplikacije slanje HTTP POST zahtjeva s dva objekta u tijelu JSON poruke. Prvi objekt predstavlja predavanje sa svim njegovim podatcima kao što su naziv, kolegij za koji se veže predavanje, predavaonica, te status predavanja. Drugi objekt je lista svih datuma tijekom kojih će se takvo predavanje održavati. Kao što je vidljivo na slici Sl. 13.4 nakon što je web servis autorizirao zahtjev od web aplikacije, započinje validacija podataka. Web servis provjerava da li postoji kolegij za koji je definirano predavanje, te da li postoji takva predavaonica u kojoj će se održati predavanje. Isto tako provjerava da li postoji neko predavanje koje je već definirano u isto vrijeme i u istoj predavaonici. Ako jedan od provjera nije uspješan, web servis vraća HTTP status *Bad Request* koji označava da zahtjev nije uspješno obrađen. Ako su sve provjere prošle uspješno web servis započinje dodavanje novih predavanja prema vremenu i datumu iz liste koje je dobio u HTTP zahtjevu. Nakon što su sva predavanja i njihova mjesta i datumi izvođenja dodani, web servis vraća HTTP status *Created* koji označava da je zahtjev uspješno obrađen. Uređivanje podataka o predavanjima započinje slanjem HTTP GET zahtjeva koji u parametru URI adrese ima oznaku koja govori za koje predavanje se zahtijevaju podatci. Nakon što predavač izmijeni podatke o predavanju, web aplikacija šalje PUT zahtjev prema web servisu koji u tijelu poruke ima dva objekta: predavanje i listu datuma i vremena koji opisuju kada se predavanje treba održati. Web servis i tu vrši validaciju, prvo provjerava da li takvo predavanje postoji u sustavu, a zatim provjerava da li su podatci o kolegiju i predavaonici točni. Provjera da li već postoji predavanje za istu predavaonicu u isto vrijeme se odrađuje i ovdje. Ako su sve validacije prošle uspješno web servis sprema izmjene u bazu podataka i vraća HTTP status *OK* prema web aplikaciji. CRUD operacije nad predavanjem završavaju HTTP DELETE metodom koja validira da li takvo predavanje postoji, te ako postoji briše isto iz baze podataka.

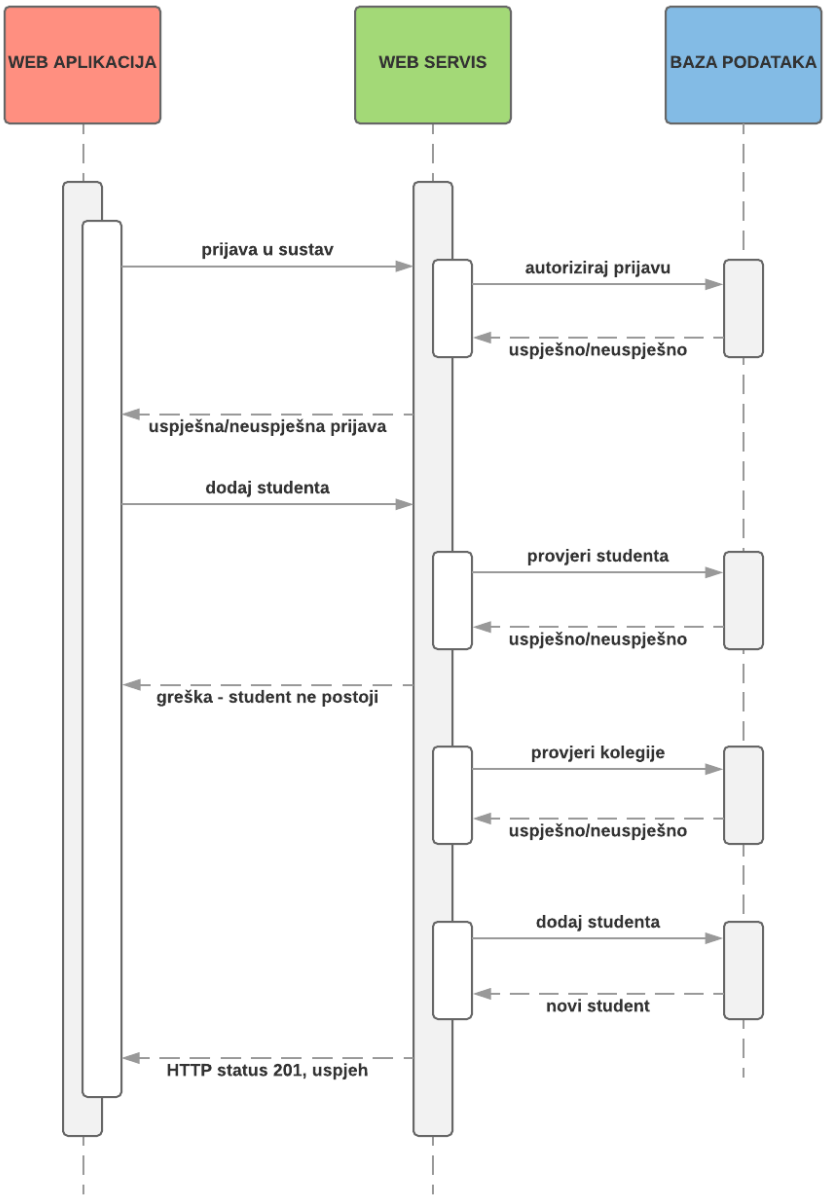


Sl. 13.4 Dodavanje novog predavanja

(izvor: autor)

Proces dodavanja novog studenta započinje slanjem HTTP POST zahtjeva iz web aplikacije prema web servisu. Zahtjev u tijelu poruke sadrži model studenta i listu svih njegovih kolegija. Model studenta mora imati definirane korisničke podatke koji definiraju svakog studenta i kao korisnika sustava. Osim korisničkih podataka, student mora imati definirane podatke o godini studija, odjelu, te kolegije na čija predavanja namjerava dolaziti. Nakon što je web servis dobio zahtjev od web aplikacije, započinje validacija podataka. Servis prvo provjerava da li su poslani ispravni podatci za studenta, odnosno da li student postoji u sustavu kao registrirani korisnik. Ako student nije registriran kao korisnik sustava web servis prekida operaciju dodavanja i vraća HTTP status *Bad Request* prema web aplikaciji. Ako student postoji web servis nastavlja pozivanjem baze podataka i

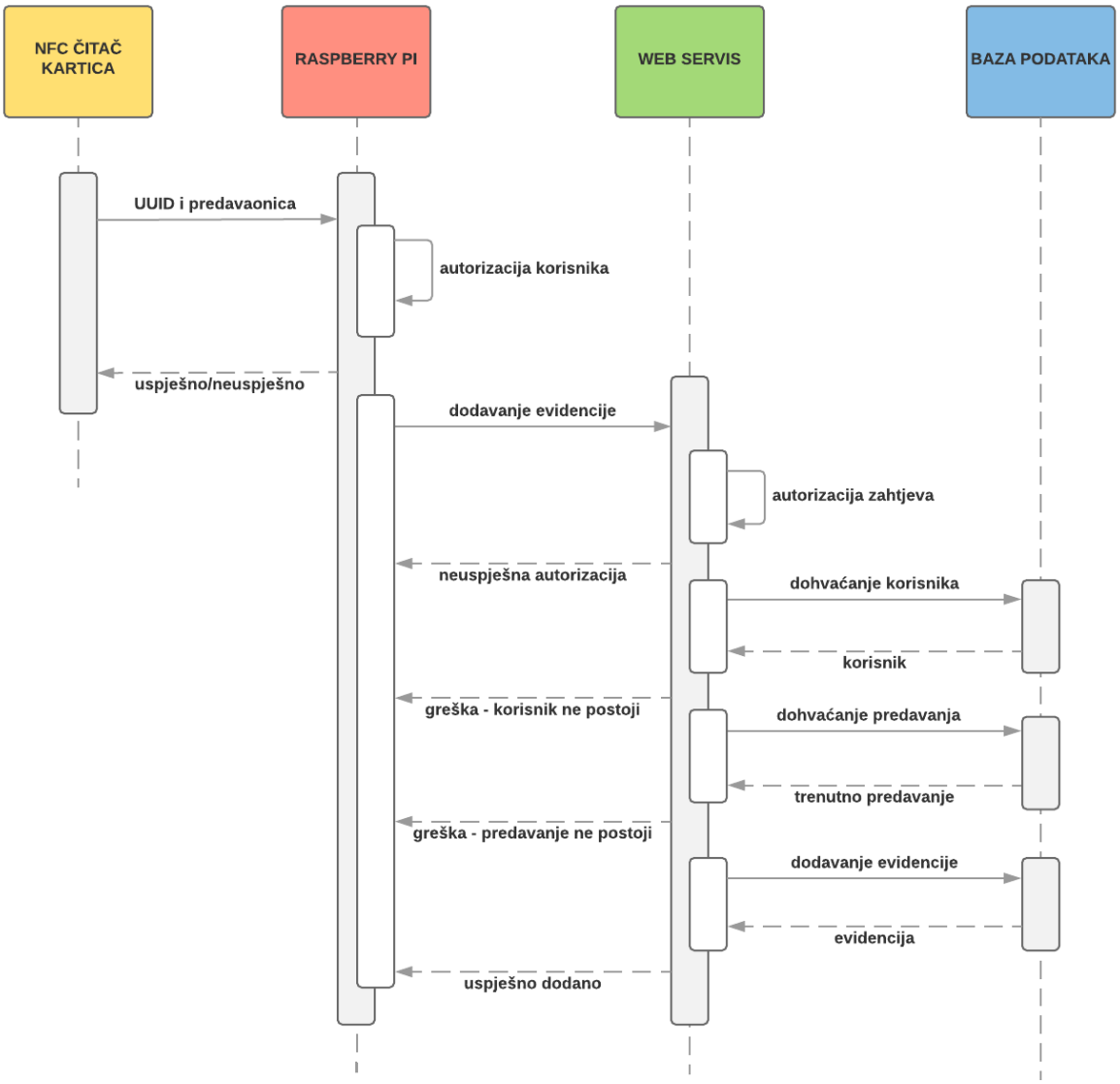
slanjem SQL naredbe koja sprema podatke u tablicu *Students*. Nakon što je dodavanje studenta uspješno završilo, web servis prolazi po listi kolegija, te za svaki radi provjeru da li je kolegij upisan u bazu podataka. Ako je validacija uspjela web servis radi novi unos u tablicu *Students_Courses*. Tablica služi da bi se studentima omogućilo pregledavanje evidencija na predavanjima po upisanim kolegijima. Na slici Sl. 13.5 opisan je proces dodavanja novog studenta s popisom njegovih kolegija. Web servis osim dodavanja studenta omogućuje web aplikaciji slanja upita za dohvat svih studenata u sustavu, dohvat jednog studenta prema njegovom jedinstvenom identifikatoru, te uređivanje i brisanje studenata.



Sl. 13.5 Dodavanje novog studenta

(izvor: autor)

Glavni proces u sustavu za evidenciju studenata je sama evidencija dolazaka. Ona se obavlja putem NFC čitača kartica koji služi da bi se studenti uspješno identificirali na osnovu vlastitih studentskih iskaznica. NFC čitač kartica nije u mogućnosti pohranjivati podatke interno već je povezan s Raspberry Pi ugradbenim sustavom. Raspberry je, kao što je opisano u diplomskom radu, funkcionalno vrlo sličan pravom računalu osim po svojim dimenzijama. Njegove karakteristike čine ga idealnim sredstvom za slanje podataka s čitača kartica na Internet prema udaljenom web servisu. Proces evidencije dolazaka započinje dolaskom predavača na predavanje. Predavač je dužan pri dolasku prisloniti vlastitu iskaznicu na čitač kartica i time započeti evidenciju dolazaka za sve studente koji su upisali taj kolegij. Nakon što je prislonio iskaznicu, NFC čitač kartica će pokušati očitati jedinstveni identifikator koji je zapisan na kartici. Nakon uspješnog čitanja podatci se iz čitača šalju prema Raspberry Pi ugradbenom sustavu. Raspberry u tom trenutku radi HTTP PUT zahtjev prema web servisu na URI adresi: */api/v1/attendance/lecture*. Web servis tada provjerava podatke s kartice i dohvaća korisnika čiji se podatci poklapaju sa onima na kartici. Nakon što je dohvatio korisnika, web servis na temelju njegovih podataka i podataka o trenutnom datumu i vremenu pokušava odrediti o kojem predavanju se radi. Ako predavanje ne postoji web servis vraća HTTP status *Bad Request* i odustaje od unosa evidencije. U suprotnom web servis dohvaća predavanje čiji status mijenja u otvoreno i vraća HTTP status *OK* prema Raspberry Pi ugradbenom sustavu. Time je predavanje promijenilo status iz zatvorenog u otvoreno, te je omogućeno evidentiranje studenata koji su došli na predavanje. Studenti se evidentiraju na način da web servis dobije HTTP POST zahtjev koji sadrži podatke s studentske iskaznice i identifikator predavaonice u kojemu se održava predavanje. Web servis tada pokušava na temelju podataka s iskaznice dohvatiti studenta. Ako student ne postoji proces se prekida i web servis vraća HTTP status *Bad Request*. Ako korisnik s tim identifikatorom postoji, web servis pokušava na temelju trenutnog datuma i vremena, te oznake predavaonice pronaći predavanje koje ima status otvoreno. Ako takvo predavanje postoji servis dodaje novi unos u tablicu evidencija predavanja za studenta i vraća HTTP status *Created*. Ako predavanje ne postoji web servis prekida proces dodavanja evidencije i odgovara Raspberry Pi ugradbenom sustavu s HTTP statusom Bad Request. Na slici Sl. 13.6 prikazan je proces dodavanja nove evidencije prisutnosti.



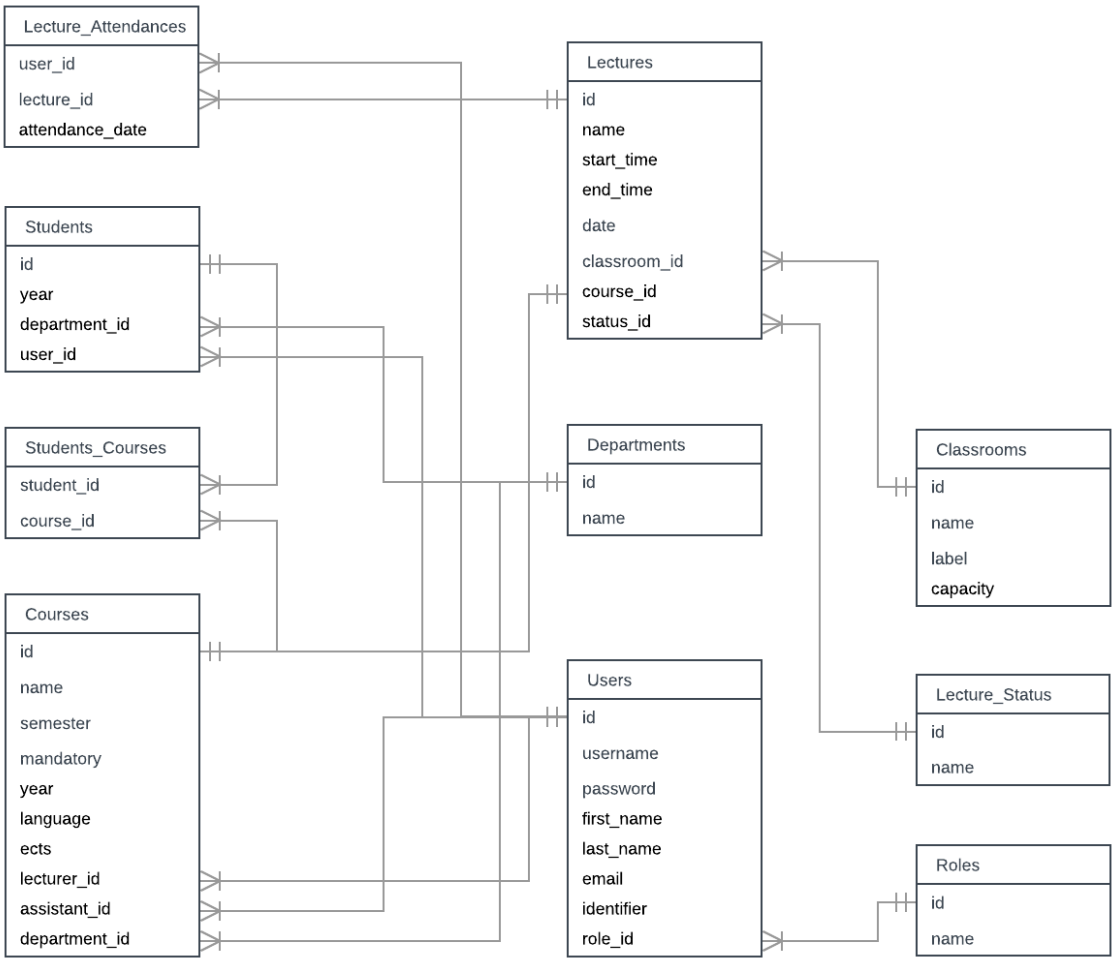
Sl. 13.6 Dodavanje evidencije prisutnosti

(izvor: autor)

Web aplikacija omogućuje predavaču pregled svih njegovih kolegija, te evidencija dolazaka studenata koji su prisutni na njegovim predavanjima. Na taj način predavač može vrlo jednostavno dobiti informacije koji studenti dolaze redovito, a koji često izostaju s njegovih predavanja. Svaki student ima svoju email adresu, pa ga predavač može kontaktirati vezano uz njegove izostanke. Uloga studenta je prvenstveno evidencija prisutnosti na predavanjima. Stoga mu sustav omogućuje brz i jednostavan pregled svih dolazaka i izostanaka sa predavanja. Na taj način student ne treba kontaktirati nositelja kolegija za količinu izostanaka, nego može dobiti te iste informacije u bilo koje vrijeme kroz sučelje web aplikacije.

13.1. Baza podataka

Baza podataka je realizirana oko nekoliko glavnih entiteta. Jedna od prvih zadaća web aplikacije je razdijeliti funkcionalnosti sustava na temelju različitih uloga korisnika. Upravo zato shema baze podataka sadrži entitet *Users* uz koji opisuje samog korisnika i entitet *Roles* s kojim opisuje uloge korisnika. Korisnici s ulogom studenta imaju vezu na entitet *Students*. Takva veza postoji iz razloga što sustav mora za svakog studenta znati njegovu godinu studija i njegov odjel na visokom učilištu, čiji podatci nisu namijenjeni za entitet korisnika. Da bi sustav znao koje je kolegije student upisao potrebno je imati tablicu koja će povezati dva entiteta. *Students_Courses* je tablica koja opisuje vezu *više-na-više* između entiteta *Students* i *Courses.* Čitajući podatke iz te tablice sustav zna točno koji kolegij pripada kojem studentu. Svaki kolegij, uz osnovne podatke, ima odjel kojemu pripada, te nositelja i asistenta. Korisnici s ulogom predavača ne zahtijevaju vlastitu tablicu zato što svi atributi koji ih opisuju mogu biti sadržani u tablici *Users*. Da bi se izvršila kontrola prisutnosti studenata na kolegijima bitno je imati popis njegovih predavanja. Upravo zato svaki kolegij ima vezu na entitet *Lectures*. Predavanja, osim poveznice na kolegije, imaju i oznaku predavaonice koja definira mjesto izvođenja tog predavanja. Bitna oznaka je i status u kojemu se predavanje trenutno nalazi, opisana entitetom *Lecture_Status*. Svako predavanje može imati tri statusa: započeto, završeno i zatvoreno. Statusi pružaju veći nivo kontrole prema predavanjima zato što definiraju točno vrijeme, između statusa započeto i završeno, kada predavanje može bilježiti evidencije. Kontrola prisutnosti studenata je naposljetku realizirana kroz entitet *Lecture_Attendances*. U njemu se bilježi svaki dolazak putem jedinstvenog identifikatora korisnika sustava, te točnog datuma i vremena kad je korisnik bio prisutan na predavanju. Na slici Sl. 13.7 prikazana je E-R (engl. *Entity-Relation*) shema baze podataka.



Sl. 13.7 Shema baze podataka

(izvor: autor)

13.2. Opis web aplikacije

Web aplikacija je realizirana uz pomoć Angular web razvojnog okruženja. Grafičko sučelje je izrađeno u obliku single page web aplikacije Kako bi djelatnici visokog učilišta mogli administrirati podatke što lakše. *Single page* aplikacije se izvode unutar jedne stranice i omogućuju korisniku iskustvo slično onome kao kod *desktop* aplikacija. Web aplikacija kao podlogu za grafički dizajn koristi *Bootstrap* biblioteku. *Bootstrap* je besplatna biblioteka koja uključuje HTML i CSS predloške, stilove, fontove, te mnoštvo dokumentacije koja pomaže kod dizajniranja web aplikacije. Glavna značajka ove biblioteke je podrška za responzivan dizajn što znači da će se aplikacija za evidenciju prilagoditi bilo kojoj rezoluciji i uređaju na kojemu se prikazuje. Biblioteka također pruža podršku većinu Internet preglednika.

13.2.1. Prijava u sustav

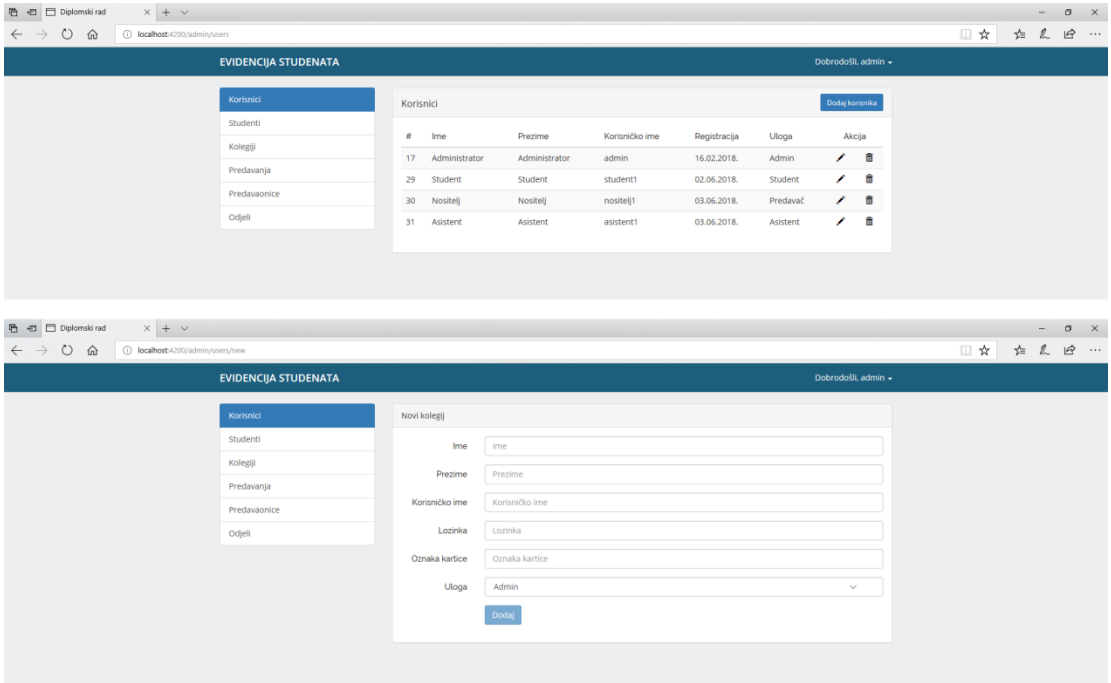
Prijava u sustav je početni zaslon web aplikacije. Sastoji se od dva tekstualna polja za unos, *checkbox* kontrole „*zapamti me*“, te gumba za prijavu. Korisnik se u sustav prijavljuje upisivanjem vlastitog korisničkog imena i lozinke. Ako korisnik unese pogrešno korisničko ime ili lozinku web servis mu tada odbija pristup prema ostatku aplikacije i vraća HTTP status *Unauthorized.* Tada se na sučelju web aplikacije prikazuje poruka o neuspješnoj prijavi: „*Greška! Korisnik nije autoriziran*“. Ako korisnik unese točne podatke i validacija na web servisu prođe uspješno s HTTP statusom *OK,* tada sustav preusmjerava korisnika na njegovu početnu stranicu. Preusmjeravanje se radi na različite rute ovisno o tome da li se radi o korisniku s ulogom administratora, studenta ili predavača. Ako se radi o administratoru korisnik se preusmjerava na rutu */admin/dashboard.* Korisnike s ulogom predavača sustav preusmjerava na */lecturer/dashboard*, dok one s ulogom studenta preusmjerava na */student/dashboard*. Korisnik naposljetku ne mora svaki put iznova raditi prijavu u sustav zato što web aplikacija kod svake uspješne prijave sprema korisničke podatke u lokalnu pohranu Internet preglednika. Na slici Sl. 13.8 prikazan je ekran za prijavu u sustav za evidenciju studenata.

Sl. 13.8 Prijava u sustav

(izvor: autor)

13.2.2. Korisnici

Sučelje koje vidi administrator započinje pregledom svih korisnika koji su trenutno upisani u sustavu. Administratoru se nudi pregled informacija kao što su ime, prezime, korisničko ime, datum registracije, te uloga korisnika. Korisnici su prikazani u tablici responzivnog dizajna koja se prilagođava rezoluciji uređaja na kojemu se nalazi. U zadnjoj koloni tablice nalaze se tipke sa uređivanje i brisanje korisnika. U zaglavlju panela s popisom korisnika administrator može odabrati tipku „*Dodaj korisnika*“ kojom mu se otvara novo sučelje s formom za unos korisnika. Odabirom tipke za uređivanje podataka, administrator dobiva isto sučelje kao i za dodavanje korisnika samo s već popunjenim podatcima. Prilikom svake akcije mijenja se ruta u adresi Internet preglednika. Odabirom opcije dodavanja korisnika ruta postaje */admin/users/new*, dok opcija uređivanja preusmjerava korisnika na rutu */admin/users/edit/30*. Na taj način administrator može vrlo jednostavno pristupati raznim sučeljima koristeći predefinirane rute kroz adresnu traku svog Internet preglednika. Administrator može i odabrati akciju brisanja korisnika koja otvara plutajući izbornik sa porukom upozorenja i tipkama za brisanje ili vraćanje na pregled svih korisnika u sustavu. Na slici Sl. 13.9 prikazano je sučelje za administraciju korisnika.

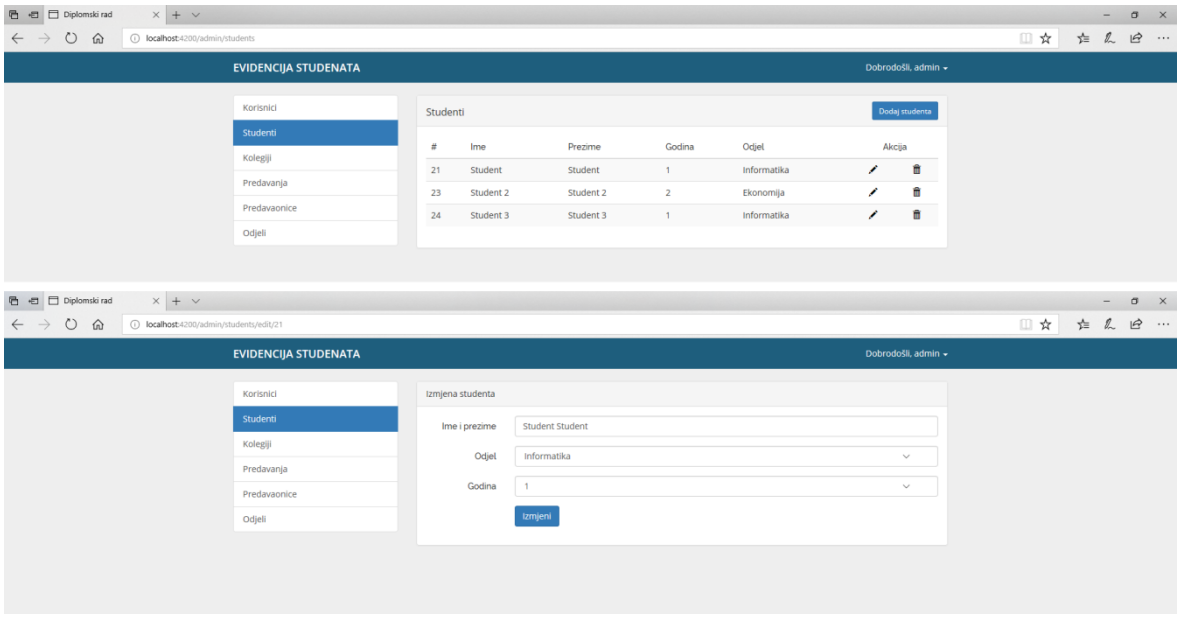


Sl. 13.9 Administracija korisnika

(izvor: autor)

13.2.3. Studenti

Sučelje za pregled studenata ima ulogu prikazivanja svih studenata, te onih korisnika s ulogom studenta za kojeg administrator još nije definirao podatke poput njegovog odjela, godine studija i kolegija koje je student upisao. Pritiskom na tipku „*Dodaj studenta*“ administrator otvara novo sučelje na ruti */admin/students/new*. Na tom sučelju administrator dobiva formu s nekoliko padajućih izbornika. Na prvom izborniku se nalazi popis svih korisnika u sustavu koji imaju ulogu studenta. Odabirom studenta, administrator može iz drugog padajućeg izbornika odabrati odjel kojemu student pripada, te njegovu godinu studija. Nakon što je odabrana godina studija, web aplikacija radi upit prema web servisu gdje traži popis kolegija koji odgovaraju odabranom odjelu i godini studija. Takav popis prikazuje administratoru na sučelju i čeka odabir svih ili samo dijela ponuđenih kolegija. Nakon što su kolegiji odabrani administrator može poslati zahtjev za dodavanjem novog studenta prema web serveru sustava za evidenciju. Na slici Sl. 13.10 prikazano je sučelje za administraciju studenata.

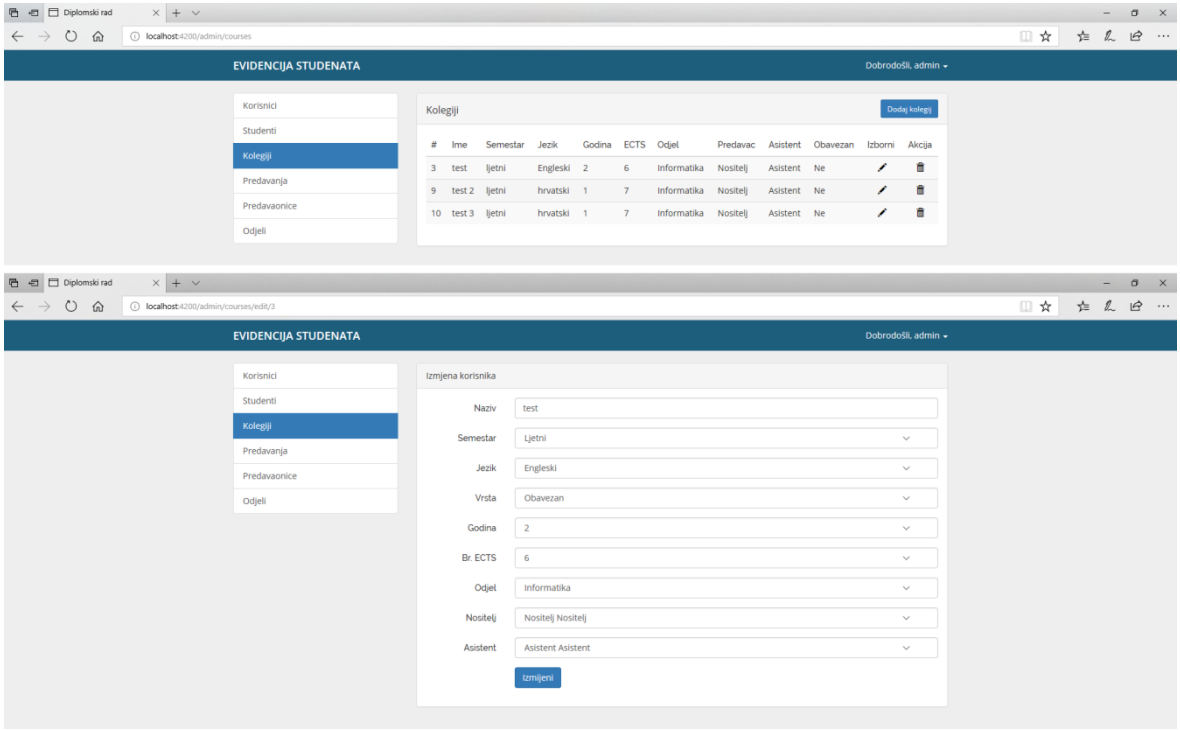


Sl. 13.10 Administracija studenata

(izvor: autor)

13.2.4. Kolegiji

Administrator može pregledavati sve kolegije u sustavu kroz sučelje kolegiji. Svaki kolegij opisuju nekoliko atributa: naziv, semestar, jezik na kojemu se izvodi, broj ECTS bodova, da li je kolegij izborni ili obavezni, te tko je nositelj i asistent na kolegiju. Odabirom tipke „*Dodaj kolegij*“ iz zaglavlja panela administrator otvara novo sučelje s formom za unos podataka o kolegiju. Svaki kolegij mora imati jednog nositelja i jednog asistenta. Svaki predavač u sustavu za evidenciju može imati definiranu ulogu nositelja ili asistenta, stoga forma za unos novog kolegija nudi padajući izbornik preko s popisom svih korisnika koji imaju takve uloge. Administrator može uređivati podatke o kolegiju: mijenjati jezik izvođenja, semestar, godinu, naziv, nositelja ili asistenta, te odjel. Na isti način kao i kod sučelja za korisnike, administrator odabirom akcijske tipke za brisanje otvara modalni prozor putem kojeg može trajno izbrisati kolegij iz sustava. Na slici Sl. 13.11 prikazano je sučelje za administraciju kolegija.

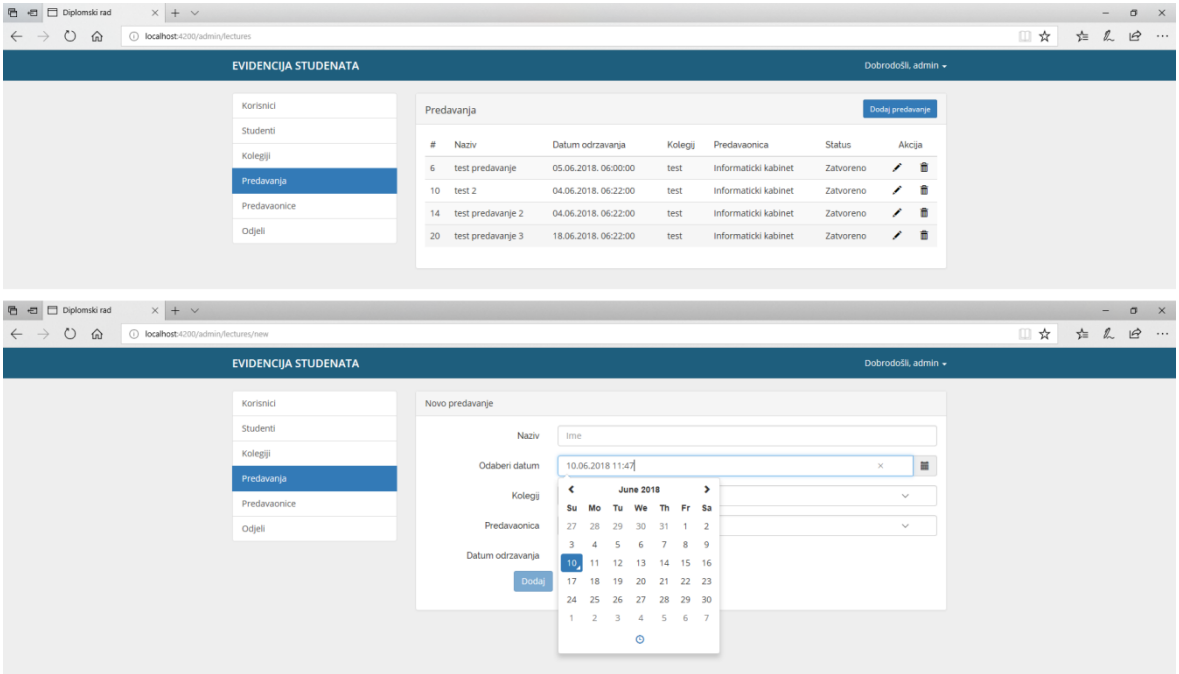


Sl. 13.11 Administracija kolegija

(izvor: autor)

13.2.5. Predavanja

Da bi se student uspješno evidentirao u sustavu za evidenciju studenata potrebno mu je definirati sva predavanja vezana uz kolegije koje je upisao. Iz tog razloga administrator ima ovlasti pregledavati, dodavati, uređivati i brisati sva predavanja u sustavu. Predavanja su prikazana na isti način kao i ostali matični podatci s akcijskim tipkama za uređivanje i brisanje. Odabirom tipke „*Dodaj predavanje*“ administrator otvara sučelje za unos novog kolegija. Sučelje se nalazi na adresi */admin/lectures/new* koje prikazuje formu sa tekstualnim poljima za unos svih podataka o predavanju. Administrator odabire naziv predavanja, kolegij kojemu predavanje pripada, te oznaku predavaonice u kojoj će se predavanje izvršiti. Osim mjesta izvođenja potrebno je odabrati i točan datum i vrijeme. Iz tog razloga forma za unos ima kalendar kontrolu, koja kroz plutajući izbornik prikazuje kalendar s datumima i tipkom za odabir točnog vremena u satima i minutama. Administrator može spremiti odabrane podatke i vratiti se na sučelje s prikazom svih predavanja. Podatke o već definiranim predavanjima, njihovim nazivima, kolegijima, te vremenu i mjestu izvođenja je moguće uređivati i brisati odabirom odgovarajuće akcijske tipke na tablici s prikazom svih predavanja. Na slici Sl. 13.12 prikazano je sučelje za administraciju predavanja.

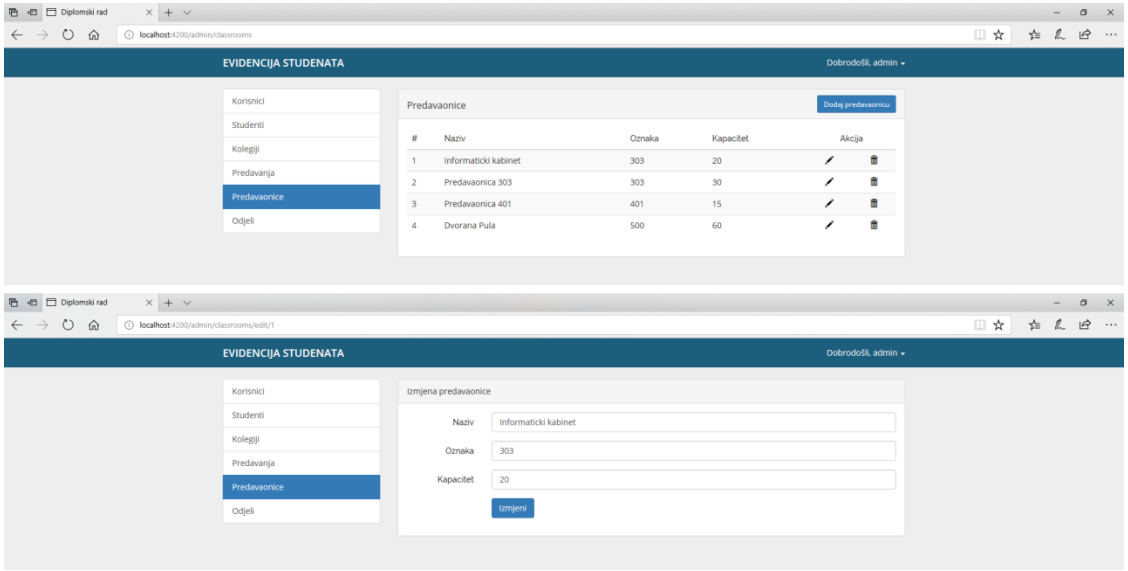


Sl. 13.12 Administracija predavanja

(izvor: autor)

13.2.6. Predavaonice i odjeli

Da bi se svakog studenta moglo uspješno evidentirati u sustavu za evidenciju potrebno je imati točan popis predavanja sa definiranim mjestom i vremenom izvođenja. Mjesto izvođenja pojedinog predavanja određuje oznaka predavaonice čiji se podatci mogu pregledavati, uređivati i brisati na administratorskom sučelju web aplikacije. Kao što je prikazano na slici Sl. 13.13 dodavanje nove predavaonice započinje odabirom tipke „*Dodaj predavaonicu*“ koja otvara sučelje s formom za unos. Forma sadrži tekstualna polja koja definiraju naziv predavaonice, njezinu oznaku te kapacitet sjedećih mjesta unutar nje. Nakon uspješno spremljenih podataka, administrator može proslijediti s akcijom dodavanja predavanja koje će se održati u novo unesenoj predavaonici. Na slici prikazano je administrativno sučelje za pregled i unos novih predavaonica u sustavu za evidenciju studenata.

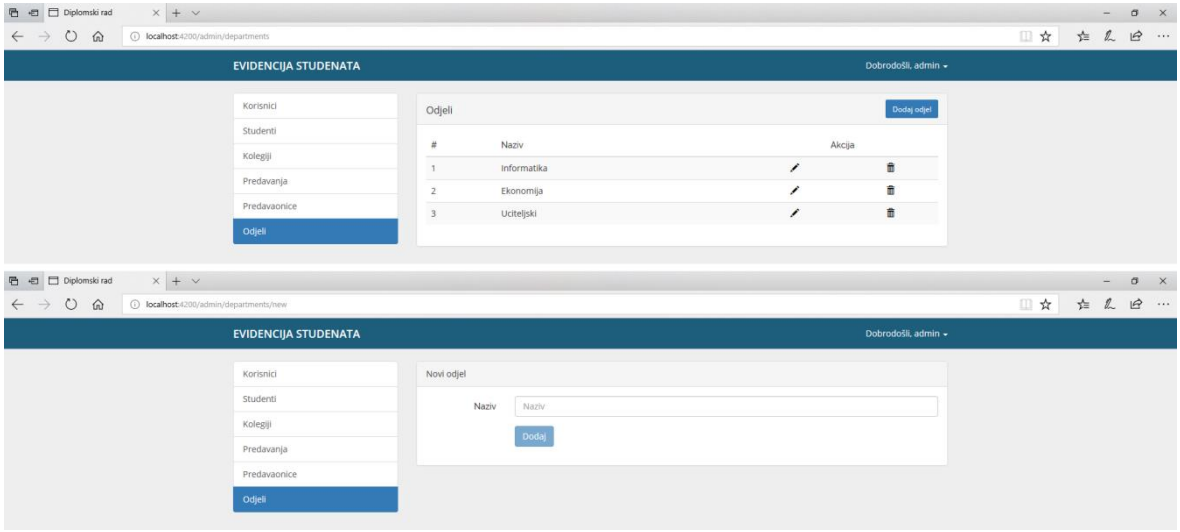


Sl. 13.13 Administracija predavaonica

(izvor: autor)

Sustav za evidenciju omogućuje administratoru pregled svih odjela na visokom učilištu. Odjeli su bitni iz razloga što pomažu kod povezivanja studenata s njihovim kolegijima. Da bi administrator dodao studenta u sustav potrebno mu je odrediti odjel i godinu studija, te na temelju toga dobiti informacije o kolegijima koje može

upisati. Na slici Sl. 13.14 je prikazano sučelje za administraciju podataka o odjelima.

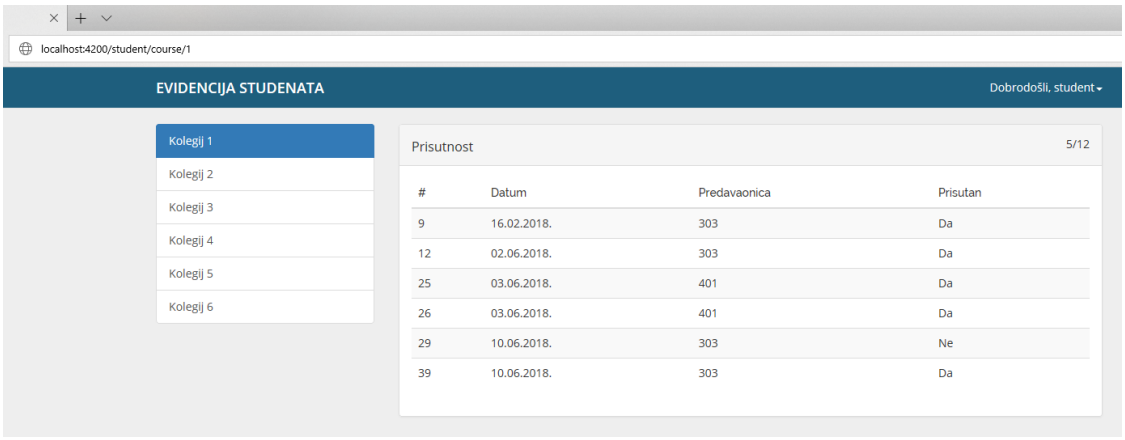


Sl. 13.14 Administriranje odjela

(izvor: autor)

13.2.7. Evidencija predavanja

Svaki student, prijavom u sustav za evidenciju, može pregledavati upisane kolegije. Odabirom određenog kolegija iz navigacije s lijeve strane sučelja web aplikacije, studentu se prikazuje popis svih predavanja koja su se održala za odabrani kolegij. Popis je prikazan u tabličnom formatu s podatcima koja opisuju datum izvođenja predavanja, predavaonicu u kojoj se predavanje održalo, te evidentirani status studenta. Ako status ima oznaku „*Ne*“, tada student može vidjeti na kojim točno predavanjima ima zabilježene izostanke. Na slici prikazano je sučelje za pregled evidencije dolazaka. Na slici Sl. 13.15 prikazan je pregled evidencije po kolegijima.

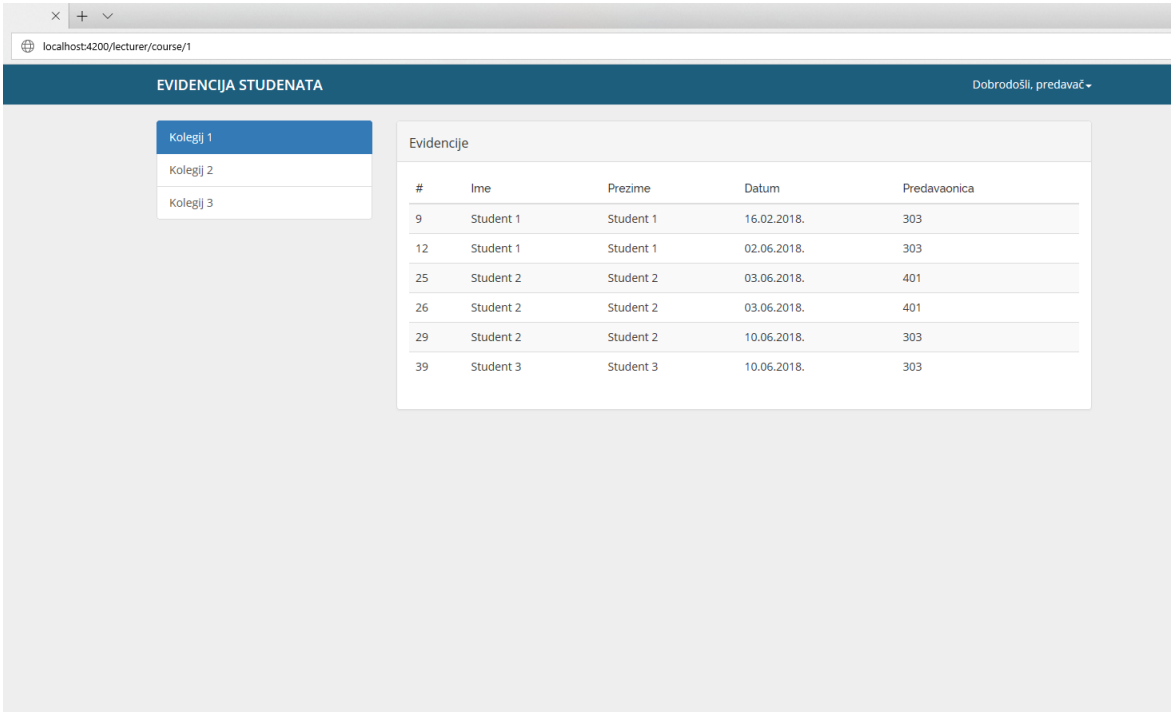


Sl. 13.15 Pregled evidencije studenata po kolegijima

(izvor: autor)

13.2.8. Evidencija studenata

Svi predavači mogu, kroz sustav za evidenciju, pregledavati kolegije na kojima su definirani kao nositelji ili asistenti. Odabirom kolegija s lijeve strane sučelja, predavač dobiva popis svih studenata koji su se uspješno evidentirali o svom dolasku. Popis evidencija prikazuje podatke o Imenu i prezimenu studenta, datumu i vremenu kada je evidencija odrađena, te oznaku predavaonice u kojoj se održalo predavanje. Uvidom u popis studenata predavač može ustanoviti koji studenti su prisutni na predavanju i koliko ih redovno pohađa kolegije. Na slici Sl. 13.16 prikazan je pregled evidencije dolazaka studenata po kolegijima.



Sl. 13.16 Pregled evidencije dolazaka po kolegijima

(izvor: autor)

Zaključak

Sustav za evidenciju studenata zahtjeva integriranje više različitih tehnologija u jednu smislenu cjelinu. Da bi se proces evidencije što više automatizirao u radu su istraženi načini rada studentskih iskaznica, kojeg su tipa, te kako pročitati podatke s njih. Čitač kartica, kojeg se opisuje kroz diplomski rad, predstavlja noviju verziju protokola za beskontaktno čitanje i komuniciranje s karticama. Novi protokoli i tehnologije omogućuju implementaciji ovog sustava veću fleksibilnost i više mogućnosti načina rada u budućnosti. Da bi se sustav za evidenciju što lakše implementirao kod jednog visokog učilišta rad istražuje i opisuje na koje sve načine visoka učilišta spremaju podatke o studentima i koriste ih u svakodnevnom radu. Istražene su mogućnosti povezivanja Raspberry Pi ugradbenog računalnog sustava i NFC ACR122U čitača kartica, te izvršavanje i način rada Raspbian operacijskog sustava naspram Android Things operacijskog sustava. Da bi naposljetku sve komponente sustava mogle međusobno komunicirati i razmjenjivati podatke opisan je način rada Spring web servisa. Web servis čini ključnu ulogu u sustavu za evidenciju zato što omogućuje komunikaciju između čitača kartica i web aplikacije preko čijeg sučelja korisnici prate dolaske i izostanke studenata s pojedinih kolegija. Rezultat rada je koncept sustava koji uspješno koristi studentske kartice, Raspberry ugradbeni sustav, web servis i *single-page* web aplikaciju za što jednostavniji proces evidencije studenata na jednom visokom učilištu.

Literatura

[1] ACS. (2018). *ACR122U usb nfc reader*. Dohvaćeno iz ACS: https://www.acs.com.hk/en/products/3/acr122u-usb-nfc-reader/

[2] Amadeo, R. (21. 5 2015). *Arstechnica*. Dohvaćeno iz Google Brillo: https://arstechnica.com/gadgets/2015/05/google-developing-brillo-internet-of-things-os-based-on-android/

[3] Chand, S. (25. 5 2017). *Why should we use Spring?* Dohvaćeno iz quora: https://www.quora.com/Why-should-we-use-Spring-What-benefits-does-Spring-provide-that-we-cannot-do-with-normal-Java-programming

[4] Freeman, A. (2017). *Pro Angular.*

[5] Gerhard Gans, J.-H. H. (2008). *A Practical Attack on the MIFARE Classic.* Dohvaćeno iz Radboud University: http://www.cs.ru.nl/~flaviog/publications/Attack.MIFARE.pdf

[6] Google. (2. 6 2018). *Android Things*. Dohvaćeno iz Android Developer: https://developer.android.com/things/get-started/

[7] Igoe, T. (2012). *Getting Started with RFID.*

[8] Kling, F. (2016). *JSON and XML comparison*. Dohvaćeno iz Stackoverflow: https://stackoverflow.com/questions/4862310/json-and-xml-comparison

[9] London, G. (9. 10 2015). *Quora*. Dohvaćeno iz Differences between Arduino and Raspberry Pi: https://www.quora.com/What-are-the-differences-between-Arduino-and-Raspberry-Pi

[10] NXP. (2018). *MIFARE Classic*. Dohvaćeno iz NXP: https://www.nxp.com/products/identification-and-security/mifare-ics/mifare-classic:MC_41863

[11] NXP. (2018). *NXP at a glance.* Dohvaćeno iz https://www.nxp.com/docs/en/supporting-information/NXPCORPORATE.pdf

[12] Play. (2018). *Documentation*. Dohvaćeno iz Playframework: https://www.playframework.com/documentation/2.6.x/JavaHome

[13] Raspberry. (2018). *Raspberry Pi documentation*. Dohvaćeno iz Raspberry Pi: https://www.raspberrypi.org/documentation/

[14] Schmidt, M. (2012). *Raspberry Pi A quick-start guide.*

[15] Shelajev, O. (23. 3 2016). *Spark framework*. Dohvaćeno iz Zeroturnaround: https://zeroturnaround.com/rebellabs/sparkjava-is-an-amazing-java-web-framework-do-you-really-need-it/

[16] Spring. (2018). *Core Technologies*. Dohvaćeno iz Spring IO: https://docs.spring.io/spring/docs/5.0.0.RC2/spring-framework-reference/core.html

[17] Spring. (2018). *Overview of Spring Framework*. Dohvaćeno iz Spring IO: https://docs.spring.io/spring/docs/5.0.0.RC2/spring-framework-reference/overview.html

[18] Spring. (2018). *The web*. Dohvaćeno iz Spring IO: https://docs.spring.io/spring/docs/5.0.0.RC2/spring-framework-reference/web.html

[19] Srce. (2018). *Djelatnost srca*. Dohvaćeno iz Srce: http://www.srce.unizg.hr/djelatnost-srca

[20] Srce. (2018). *ISAK*. Dohvaćeno iz Srce: http://www.srce.unizg.hr/isak

[21] Srce. (2018). *ISSP*. Dohvaćeno iz Srce: http://www.srce.unizg.hr/issp

[22] Srce. (2018). *REST API dokumentacija*. Dohvaćeno iz ISVU: https://www.isvu.hr/apiproba/dokumentacija/index.html#OvlastiZaKoristenjeServisa

[23] Srce. (2018). *Studentska iskaznica*. Dohvaćeno iz Srce: http://www.srce.unizg.hr/isak/studentska-iskaznica

[24] Srce. (2018). *Uvod u ISVU*. Dohvaćeno iz Wiki Srce: https://wiki.srce.hr/display/TUT/Uvod+u+ISVU

[25] Tezer, O. (21. 2 2014). *Understanding SQL And NoSQL Databases And Different Database Models*. Dohvaćeno iz Digitalocean: https://www.digitalocean.com/community/tutorials/understanding-sql-and-nosql-databases-and-different-database-models

[26] Vaqqas, M. (23. 9 2014). *RESTful web services*. Dohvaćeno iz Dobb's: http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069

[27] Werneck, P. (2017). *SOAP vs REST*. Dohvaćeno iz Stackoverflow: https://stackoverflow.com/questions/19884295/soap-vs-rest-differences

[28] Wodehouse, C. (2018). *SOAP vs REST: A look at two different API styles*. Dohvaćeno iz Upwork: https://www.upwork.com/hiring/development/soap-vs-rest-comparing-two-apis/

[29] Xplenty. (28. 9 2017). *The SQL vs NoSQL Difference: MySQL vs MongoDB*. Dohvaćeno iz Medium: https://medium.com/xplenty-blog/the-sql-vs-nosql-difference-mysql-vs-mongodb-32c9980e67b2

[30] Zöchbauer, G. (19. 1 2017). *Promise vs Observable*. Dohvaćeno iz Stackoverflow: https://stackoverflow.com/questions/37364973/promise-vs-observable

Popis slika

Sl. 1.1 Prikaz modula ISVU sustava 3

Sl. 2.1 Prikaz studentske iskaznice 7

Sl. 2.2 Mifare Classic raspored memorije 9

Sl. 3.1 Pasivni RFID sustav identifikacije 11

Sl. 4.1 NFC arhitektura 13

Sl. 5.1 Komunikacijski dijagram *ACR122U* čitača kartica 14

Sl. 5.2 ACR122U NFC čitač kartica 15

Sl. 6.1 *Raspberry Pi* ugradbeni računalni sustav 17

Sl. 7.1 Raspbian operacijski sustav 19

Sl. 8.1 Things Support biblioteka 21

Sl. 9.1 HTTP *request* i *response* 23

Sl. 9.2 Arhitektura Spring razvojnog okruženja 25

Sl. 9.3 Spring *container* inverzije kontrole 26

Sl. 9.4 Spring MVC procesiranje HTTP zahtjeva 27

Sl. 11.1 Prikaz nadskupa TypeScript programskog jezika 29

Sl. 11.2 *Round trip* web aplikacija 30

Sl. 11.3 *Single page* web aplikacija 31

Sl. 11.4 Angular MVC oblikovni obrazac 32

Sl. 11.5 Observer oblikovni obrazac 32

Sl. 12.1 Prikaz arhitekture sustava za evidenciju 34

Sl. 13.1 Dijagram slučajeva korištenja 35

Sl. 13.2 Dodavanje novog korisnika 38

Sl. 13.3 Dodavanje novog kolegija 40

Sl. 13.4 Dodavanje novog predavanja 42
Sl. 13.5 Dodavanje novog studenta 43
Sl. 13.6 Dodavanje evidencije prisutnosti 45
Sl. 13.7 Shema baze podataka 47
Sl. 13.8 Prijava u sustav 48
Sl. 13.9 Administracija korisnika 49
Sl. 13.10 Administracija studenata 50
Sl. 13.11 Administracija kolegija 51
Sl. 13.12 Administracija predavanja 52
Sl. 13.13 Administracija predavaonica 53
Sl. 13.14 Administriranje odjela 54
Sl. 13.15 Pregled evidencije studenata po kolegijima 55
Sl. 13.16 Pregled evidencije dolazaka po kolegijima 56

Sažetak

U ovom radu istražene su mogućnosti više različitih tehnologija s ciljem rješavanja jednog zadatka, a to je realizacija sustava za evidenciju studenata. Da bi što lakše evidentirao svakog studenta rad se bavi proučavanjem načina rada studentskih iskaznica, te beskontaktnih NFC čitača kartica. Nakon toga, rad istražuje razne mogućnosti povezivanja čitača kartica i Raspberry Pi ugradbenog računalnog sustava. Uspoređuju se Raspbian i Android Things operacijski sustavi, te se dalje nastavlja opisom komunikacije prema Spring Boot web servisu. Web servis, implementiran kroz Spring Boot razvojno okruženje, predstavlja središnju komponentu sustava koja integrira sve ostale dijelove u jednu smislenu cjelinu. Da bi korisnici mogli pregledavati podatke o evidenciji realizirana je i web aplikacija koristeći Angular web razvojno okruženje. Na kraju rada opisana je implementacija takvog sustava, način rada web servisa, MySQL baze podataka, te izgled web aplikacije. Sustav je realiziran samo kao prototip, odnosno koncept na koji način što više automatizirati evidenciju studenata koji su prisutni na predavanjima jednog visokog učilišta.

Ključne riječi: NFC, Spring Boot, Angular, Raspberry Pi, Android Things, Web

Summary

This paper explores the possibilities of serveral different technologies for solving one task, namely the realization of a student attendance system. To make it easier to record each student, the paper deals the way that student IDs works, and contactless NFC card readers. After that, the paper explores the various possibilities of connecting the card reader and the Raspberry Pi embedded computer system. Raspbian and Android Things operating systems are compared, and the paper continues with describing the communication with Spring Boot Web service. The Web service, implemented through the Spring Boot framework, represents the central component of a system that integrates all other parts into a single meaningful whole. In order to allow users to browse attendance records, web application has been developed using the Angular Web development enviroment. At the end of the paper, the implementation of such system, how web service works, how MySQL database works, and how does the web application look. The system is realized only as a prototype, that is, the concept of how to automate student attendances for those students who are present at the lectures of a faculty.

Keywords: NFC, Spring Boot, Angular, Raspberry Pi, Android Things, Web

Privitak

1. Izvorni kod REST web servisa
2. Izvorni kod web aplikacije
3. Izvorni kod za povezivanje s NFC čitačem kartica