IMPLEMENTACIJA SUSTAVA PRAĆENJA KRETANJA OSOBA PREKO RFID ČITAČA

Hlupić, Hrvoje

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

IMPLEMENTACIJA SUSTAVA PRAĆENJA KRETANJA OSOBA PREKO RFID ČITAČA

Hrvoje Hlupić

Zagreb, veljača 2020.

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada".

U Zagrebu, datum. 18.02.2020

Hrvoje Hlupic

Predgovor

Zahvaljujem svom mentoru Aleksanderu Radovanu na pomoći i promptnim odgovorima tijekom izrade ovog rada.

Zahvaljujem svojoj obitelji i prijateljima na potpori, pomoći i motivaciji tijekom mog studija.

Također, zahvaljujem stack overflow-u. Apes together strong.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

Cilj ovog završnog rada je napraviti sustav od nekoliko različitih hardverskih i softverskih komponenti pomoću kojeg bi se automatski pratio broj ulaza i izlaza osoba kroz štićeni prolaz. Iako takvi komercijalni sustavi postoje, trenutno su cjenovno nepristupačni i kompleksni.

Kroz ovaj rad pokazat će se kako je moguće kombinacijom različitih postojećih i jednostavnih tehnologija i hardverskih komponenti sastaviti jedan sustav koji bi imao takvu funkcionalnost, a bio bi cjenovno prihvatljiv i skalabilan. Praktični dio ovog rada temeljit će se na potpuno funkcionalnom opisanom sustavu koji će se sastojati od tri glavna dijela: .NET web aplikacije za upravljanje i pregled podataka, *remote* Python aplikacije za komunikaciju s RFID čitačem i .NET konzolne aplikacije koja će preko TCP protokola komunicirati s *remote* aplikacijom i bazom podataka.

Ključne riječi: RFID, protok, praćenje, prijava.

Summary

The goal of this thesis is to develop a system based on several different hardware and software components which will be used for automatic tracking of people flow through secured gate. Even though such systems developed for commericial use already exist, they are currently not affordable and usually very complex.

Development of this thesis will prove that it is possible to create an afordable and scalable system, by combining simple existing technologies and different hardware components. Practical part of this thesis will be a fully functional system with functionality as described, which will have three main components: .NET web application for management and data presentation, remote Python application for communication with RFID reader and .NET console application which will use TCP protocol to communicate with remote application and database.

Key words: RFID, flow, tracking

Sadržaj

1. Uvod

Trenutno na tržištu postoje različita gotova rješenja za automatsko praćenje toka ljudi/predmeta kroz označene prolaze, ali većina tih sustava cjenovno je nepristupačna, kompleksna i rađena za specifične potrebe. Kroz ovaj rad razvijen je sustav koji ima sličnu funkcionalnost, ali je jednostavniji, jeftiniji, skalabilniji i s mogućnošću različitih primjena.

Rad počinje s osnovnim pregledom tehnologija korištenih pri samom razvoju sustava. Nakon toga opisana je funkcionalnost i arhitektura sustava, gdje je dana logička podjela sustava kroz koju se vidi smisao arhitekturne podjele na slojeve. Nadalje je opisana i sama realizacija praktičnog dijela rada, gdje je u detalje objašnjena implementacija, realizacija i interoperabilnost sustava. U zadnjem poglavlju dana je kratka usporedba ovog sustava s postojećim industrijskim rješenjima.

2. Uvod u korištene tehnologije

Kroz ovo poglavlje ukratko će biti opisane glavne tehnologije korištene u ovom završnom radu i njihove osnovne značajke.

2.1. RFID tehnologija

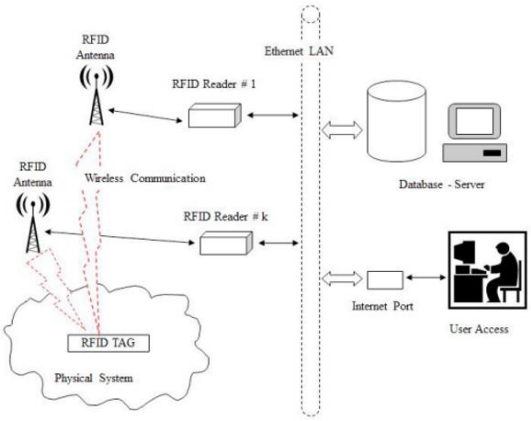
RFID (engl. *Radio Frequency Identification*) je tehnologija kojom se ostvaruje bežična komunikacija i prijenos podataka između subjekata (jedinica) koji na sebi imaju RFID tag i centralnog računala koji sakuplja i obrađuje te podatke.[1]

RFID sustav najčešće se sastoji od tri glavna dijela: kartica (tagova), antena i kontrolera. No, u nekim inačicama sustava antena i kontroler su integrirani u jednu jedinicu koja se naziva RFID čitač. Takav sustav s RFID karticom (engl. *RFID tag)* i RFID čitačem bit će razvijen i opisan kroz ovaj završni rad.

Također, postoje dvije vrste RFID tagova - aktivni i pasivni. Aktivni imaju širi domet komunikacije i integriranu bateriju, dok pasivni zahtijevaju malu udaljenost od čitača i ne mogu sami odašiljati radio valove. U ovom sustavu korišteni su pasivni RFID tagovi.

Općenita arhitektura osnovnog RFID sustava prikazana je na slici (Slika 2.1.).

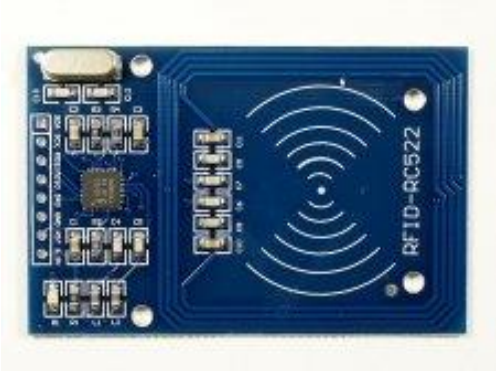
[1] Chetouane, An Overview on RFID Technology Instruction and Application



Slika 2.1. Arhitektura osnovnog RFID sustava[2]

2.2. NXP MFRC522 RFID čitač

Kao što je već spomenuto u prethodnom poglavlju, RFID čitač je komponenta RFID sustava koja u sebi ima integriranu antenu i kontroler (Slika 2.2.). U sustavu koji je predmet ovog rada koristi se MFRC522 RFID čitač/pisač koji koristi frekvenciju 13.56 MHz za bežičnu komunikaciju. MFRC522 podržava komunikaciju s ISO/IEC 14443 A/MIFARE tagovima i može postići brzinu od 848 kBd u oba smjera.[3]



Slika 2.2. Izgled MFRC522 RFID čitača

[2] Chetouane, An Overview on RFID Technology Instruction and Application
[3] NXP Semiconductors, MFRC522 Standard performance MIFARE and NTAG frontend

Za MFRC522 dostupna su sljedeća sučelja:

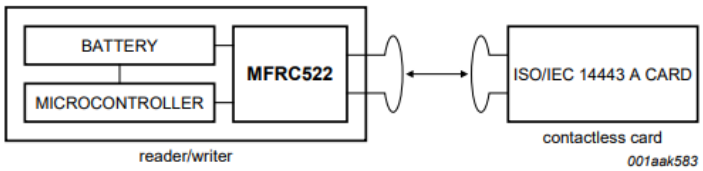
• SPI (engl. *Serial Peripheral Interface*)

• *Serial* UART

• I2C-bus *interface*

U našem sustavu korišteno je SPI sučelje.

Pojednostavljeni prikaz uloge MFRC522 čitača u sustavu prikazan je na slici (Slika 2.3.).



Slika 2.3. Pojednostavljeni prikaz uloge MFRC522 čitača u RFID sustavu[4]

2.3. Raspberry Pi

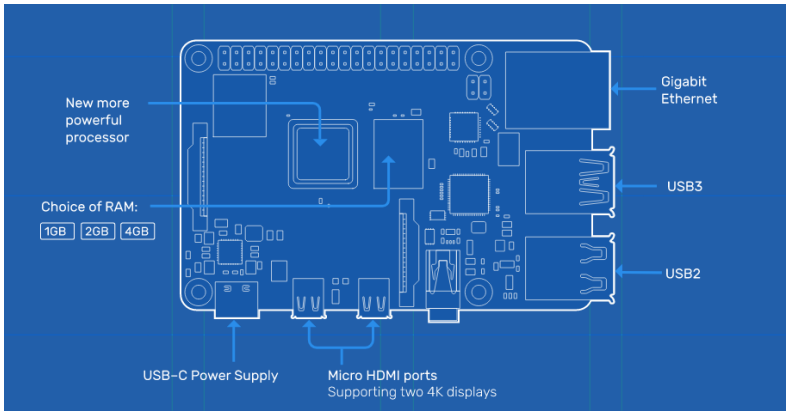
Raspberry Pi je računalo čije su glavne značajke jednostavnost, male dimenzije i niska cijena. Razvijeno je od strane organizacije *Raspberry Pi Foundation*. Motivacija za razvoj takvog, cjenovno prihvatljivog, računala bila je omogućiti učenje osnovnih znanja o programiranju i računarstvu u manje razvijenim zemljama.

Prva komercijalna verzija Raspberry Pi računala krenula je u prodaju 2012. godine, a od tada do danas izašlo je već desetak različitih modela.

U ovom sustavu koristi se model Raspberry Pi 4.

Na slici (Slika 2.4.) prikazan je shematski prikaz Raspberry Pi 4 računala:

[4] NXP Semiconductors, MFRC522 Standard performance MIFARE and NTAG frontend



Slika 2.4. Shematski model Raspberry Pi računala[5]

Za normalan rad s Raspberry Pi računalom potrebne su dodatne komponente:

1. MicroSD kartica
2. Kabel za prikaz slike - HDMI
3. Miš i tipkovnica
4. Izvor napajanja (USB Micro)

Kao mogućnost postoji spajanje *Ethernet* kabelom ili spajanje preko integriranog *wireless* modula da bi se omogućila veza s nekom lokalnom mrežom i/ili internetom.

Slika (Slika 2.5.) prikazuje izgled Raspberry Pi 4 modela.

[5] https://www.raspberrypi.org – generalne informacije o Raspberry Pi organizaciji

Slika 2.5. Izgled Raspberry Pi, model 4[6]

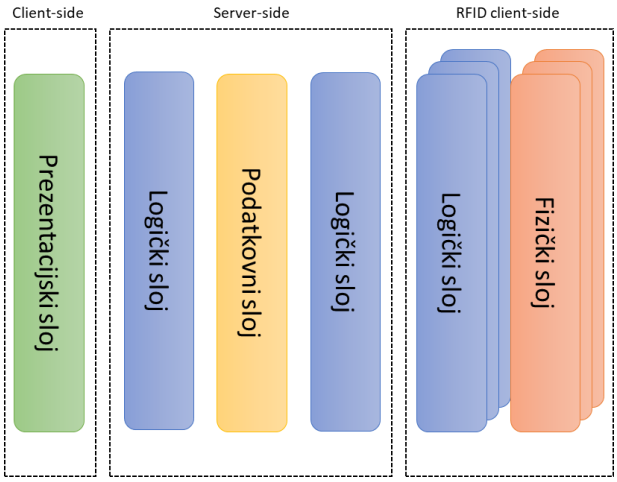
Kao operacijski sustav preporuča se Linux distribucija Raspbian, koji se koristi i u predmetnom sustavu. Dolazi s nekoliko preinstaliranih programskih jezika, među kojima je i Python koji se koristi pri razvoju ovog sustava.

[6] https://www.raspberrypi.org – generalne informacije o Raspberry Pi organizaciji

3. Arhitektura sustava

Slojevi sustava podijeljeni su u četiri vrste:

- Prezentacijski sloj
- Logički sloj
- Podatkovni sloj
- Fizički sloj



Slika 3.1. Arhitektura sustava

Pomoću ovakve podjele sustava napravljeno je razdvajanje poslova (engl. *separation of concerns*) koje osigurava da svaki sloj izvršava svoju zadaću neovisno o drugom sloju. U konačnici, opisani sustav osigurava jednostavnije održavanje i, po potrebi, bržu i lakšu nadogradnju ili promjenu sustava (na primjer, korištenje nekog drugog tipa aplikacije kao prezentacijskog sloja).

Slojevi se mogu grupirati u tri cjeline ovisno o mjestu izvršavanja i zadacima:

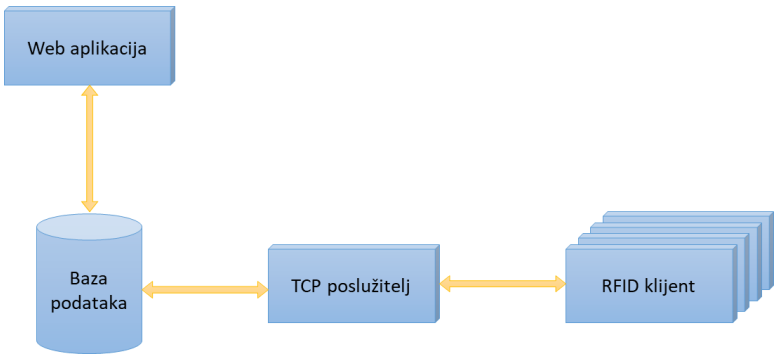
- RFID korisnička cjelina, koja se sastoji od fizičkog sloja (RFID kartica i RFID čitača) i logičkog sloja koji komunicira s poslužiteljem
- Poslužiteljska cjelina, koja se sastoji od logičkog sloja koji komunicira s RFID cjelinom i podatkovnim slojem i drugog logičkog sloja koji služi za filtriranje podataka zatraženih od prezentacijskog sloja podatkovnog sloja
- Klijentska cjelina, koja služi za *user friendly* prikaz i manipulaciju podacima

Kao što je prikazano na slici (Slika 3.1.), ovakva atomizacija sustava omogućuje da svaki sloj radi isključivo svoj „jednostavan“ zadatak i komunicira s točno određenim slojem, a cjelokupni sustav u konačnici odrađuje „složen“ posao.

4. Funkcionalnost sustava

Praktični dio rada sastoji se od četiri glavna entiteta prikazana na slici (Slika 4.1.):

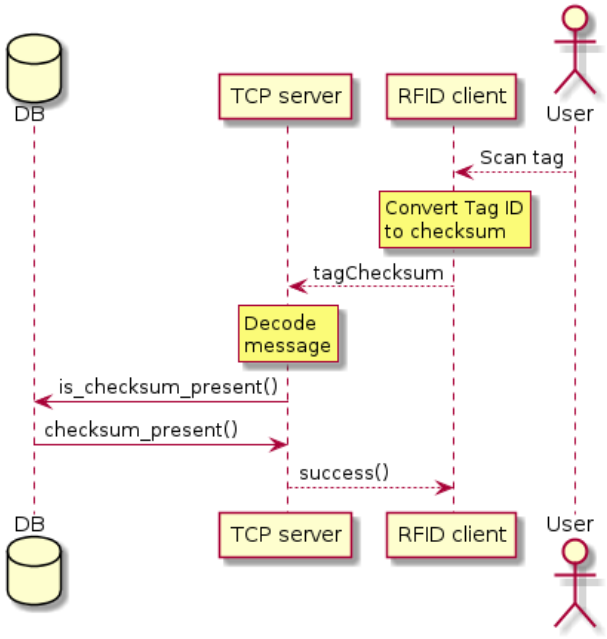
- Web aplikacija
- Baza podataka
- TCP poslužitelj
- RFID klijent(i)



Slika 4.1. Entiteti sustava

TCP poslužitelj ponaša se kao svojevrsni „servis“ koji preko TCP protokola prima podatke od jednog ili više RFID klijenata. Svaki RFID klijent preko MFRC522 RFID čitača očitava podatke s RFID kartice i preko TCP protokola šalje poziv TCP poslužitelju. TCP poslužitelj nakon svakog validnog poziva sprema podatke o RFID kartici, RFID čitaču i vremenu prolaska u bazu podataka i povratno javlja RFID klijentu da je poziv validan. RFID klijent ovisno o valjanosti poziva može imati daljnje radnje (na primjer, otvaranje vrata). Dijagram takve sekvence prikazan je na slici (Slika 4.2.).

Web aplikacija u sklopu prezentacijskog sloja ima zadaću korisničkog sučelja za pregled unesenih podataka (zapisi o ulascima, popis RFID čitača, RFID kartica i popis korisnika), a u sklopu logičkog sloja omogućene su CRUD operacije nad modelima korisnika, RFID kartica i RFID čitača.



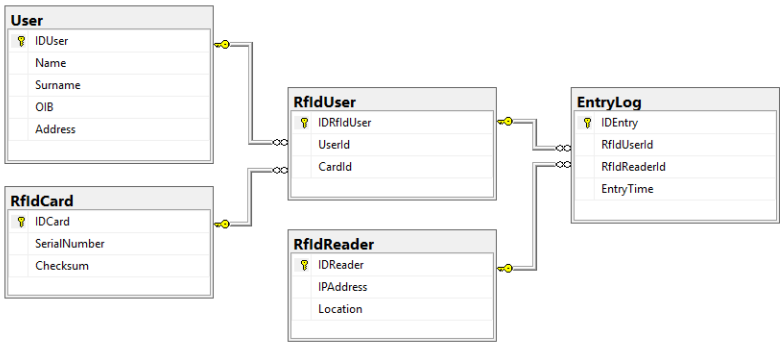
Slika 4.2. Dijagram sekvence prijave korisnika

5. Realizacija sustava

U narednom poglavlju, detaljno će biti opisan i pojašnjen način implementacije svih entiteta sustava pojedinačno.

5.1. Baza podataka

Za implementaciju baze podataka koristi se Microsoft SQL server. Na slici (Slika 5.1.) prikazan je relacijski model baze podataka.



Slika 5.1. Relacijski model baze podataka

Objašnjenje tablica i stupaca prikazano je u tablici (Tablica 5.1.):

Tablica 5.1. Opis tablica u bazi podataka

Naziv tablice	Opis
User	Tablica sadrži podatke o korisniku (Ime, prezime, OIB i adresu)
RfIdCard	Tablica sadrži podatke o svakoj RFID kartici (broj kartice i kontrolni broj)
RfIdReader	Tablica sadrži podatke o svakom RFID čitaču (lokacija i IP adresa)
EntryLog	Tablica sadrži zapise ulaska (ID korisnika, ID čitača i vrijeme ulaska)
RfIdUser	Tablica se koristi kao veza između korisnika i RFID kartice

Baza je normalizirana u 3NF. Iako se na prvi pogled čini da je tablica *RfIdUser* redundantna, njezina svrha je da se u budućnosti ostavi mogućnost implementacije na način da pojedini korisnik nema svoju RFID karticu, odnosno da jedan korisnik ima više RFID kartica.

Za komunikaciju s bazom podataka koristi se *Microsoft Entity Framework* s *database first* pristupom. Takav pristup developerima .NET aplikacija uvelike olakšava pristup bazi podataka. Tablice iz baze podataka u .NET aplikaciji prezentiraju se kao objekt klase, a stupci tablice kao parametri klase. Kod za sve klase, zajedno s ograničenjima i konekcijskim *stringom* generira se automatski.

5.2. RFID klijent

RFID klijent aplikacija nalazi se na udaljenom Raspberry Pi uređaju i njezina glavna zadaća je komunikacija s MFRC522 RFID čitačem i aplikacijom TCP poslužitelja. Na taj način omogućeno je jeftino i relativno jednostavno rješenje, budući da preko Raspberry Pi računala imamo pristup *ethernet* mreži i mogućnost prilagodbe specifičnim zahtjevima, bez kupovanja, odnosno proizvodnje personaliziranog hardvera.

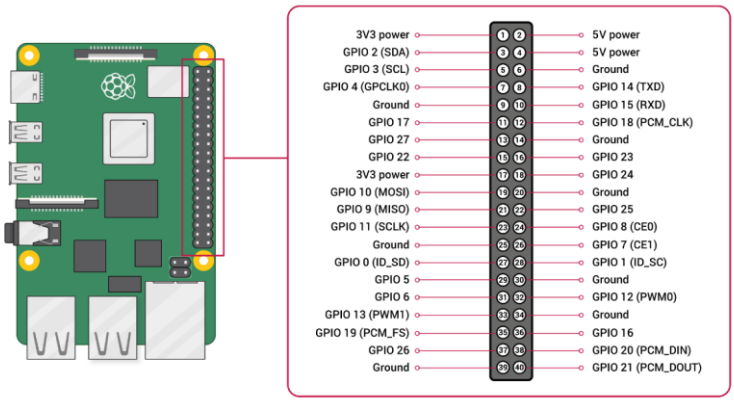
Za povezivanje aplikacije i komunikaciju s MFRC522 RFID čitačem koristi se *open-source SimpleMRFC522* programska biblioteka[7]. Programska biblioteka napisana je za programski jezik Python i daje nam sučelje za komunikaciju s MFRC522 RFID čitačem, odnosno mogućnost čitanja i pisanja na RFID kartice preko čitača.

MRRC522 RDID čitač spaja se na Raspberry Pi konektore (Slika 5.2.) na sljedeći način:

- SDA se spaja na pin 24
- SCK se spaja na pin 23
- MOSI se spaja na pin 19
- MISO se spaja na pin 21
- GND se spaja na pin 6
- RST se spaja na pin 22
- 3.3v se spaja na pin 1

[7] https://github.com/pimylifeup/MFRC522-python

Nazivi konektora nalaze se na RFID čitaču, detaljniji opis nalazi se u produktnoj dokumentaciji[8].



Slika 5.2. Popis konektora na Raspberry Pi računalu [9]

Aplikacija se sastoji od glavne beskonačne petlje, gdje je `reader.read()` blokirajuća funkcija koja čeka na prihvaćanje podataka od strane čitača (odnosno, čeka korisnika da očita RFID karticu). Nakon što se očita identifikator RFID kartice, otvara se TCP konekcija prema TCP poslužitelju i šalje se ID prethodno pretvoren u MD5 *hash*. Na taj način onemogućeno je da ukoliko netko dođe do tog podatka, nije u mogućnosti znati koja je stvarna vrijednost na RFID kartici.

Nakon što se podatak pošalje, otvara se TCP *listen socket* koji čeka potvrdu od strane aplikacije TCP poslužitelja. Ukoliko unutar tri sekunde nije dobivena tražena potvrda, petlja se prekida i vraća se u stanje čekanja na očitavanje RFID kartice. Ako potvrda stigne, izvršava se određena radnja – šalje se signal za otvaranje vrata, no moguće je napraviti bilo koji niz akcija (upaliti zeleno svjetlo u slučaju uspješnog očitavanja, odnosno crveno u slučaju neuspješnog, dodati zvučni signal, itd.). Način implementacije prikazan je u nastavku (Kod 5.1.).

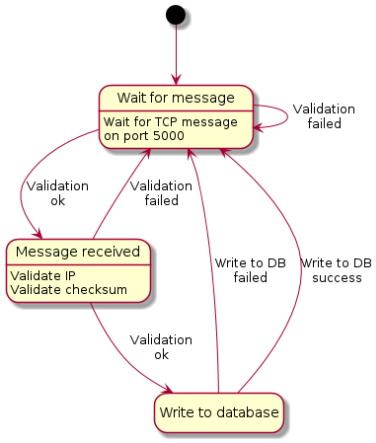
[8] https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf
[9] https://www.raspberrypi.org/documentation/usage/gpio/

```
reader = SimpleMFRC522()

while True:
  try:
    cardId, _text = reader.read()

    // send_TCP message
    tcpSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcpSocket.connect((SERVER_IP, TCP_PORT))
    tcpSocket.send(hashlib.md5(cardId))

    // wait for 'success' message
    tcpSocket.settimeout(TIMEOUT)
    tcpSocket.listen(1)
    while True:
      conn, addr = tcpSocket.accept()
      data = conn.recv(BUFFER_SIZE)
      if data == 'success':
        conn.close()
        openDoor()
        break
      conn.close()
  except:
    print("Did not receive TCP message")
  finally:
    GPIO.cleanup()
```

Kod 5.1. Implementacija RFID klijentske aplikacije

5.3. TCP poslužitelj

Zadaća ovog modula je komunikacija s udaljenim RFID klijentima, validacija dobivenih poruka i spremanje unosa u bazu podataka. Modul je napravljen kao zasebna konzolna .NET aplikacija koja je pokrenuta na poslužitelju. Na slici (Slika 5.3.) prikazan je dijagram stanja koji će biti detaljnije objašnjen u ovom poglavlju.



Slika 5.3. Dijagram stanja TCP posužitelja

Glavna (*main*) metoda aplikacije sastoji se od beskonačne petlje s *try-catch* blokom u kojem se u slučaju greške pomoću *Logger* klase zapisuju detalji o greškama i iznimkama u tekstualnu datoteku naziva *error_log_yyyymmdd*. Na taj način omogućen je robusni i neprestani pozadinski rad aplikacije (Kod 5.2.):

```
while (true) {
  string ipAddr;
  string dataReceived;

  if (!getTCPMessage(listener, out ipAddr, out dataReceived))
    continue;

  try {
    int? idReader = GetReaderId(db, ipAddr);
    if (idReader == null)
      continue;

    int? idUser = GetUserId(db, dataReceived);
    if (idUser == null)
      continue;

    db.EntryLog.Add(new EntryLog {
      RfIdReaderId = idReader,
      RfIdUserId = idUser,
      EntryTime = DateTime.Now });

    db.SaveChanges();
```

```
      sendTCPMessage(ipAddr);
    } catch (Exception ex) {
      Logger.LogException(ex); }
  };
```

Kod 5.2. Beskonačna petlja u TCP poslužitelju

Unutar petlje, nakon što je pokrenut *TCPListener* (System.Net.Sockets biblioteka), odlazi se u blokirajuću metodu `getTCPMessage` (Kod 5.3).

```
private static bool getTCPMessage(TcpListener listener,
                                  out string ipAddress, out string dataReceived) {
  ipAddress = null;
  dataReceived = null;

  try {
    TcpClient client = listener.AcceptTcpClient();

    if (!IPAddress.TryParse(client.Client.RemoteEndPoint.ToString().Split(':')[0],
                            out IPAddress ip)) {
      Logger.LogError("Could not parse IP address from: " +
                      client.Client.RemoteEndPoint.ToString());
      client.Close();
      return false;
    }

    ipAddress = ip.ToString();
    NetworkStream networkStream = client.GetStream();
    byte[] buffer = new byte[client.ReceiveBufferSize];

    int bytesRead = networkStream.Read(buffer, 0, client.ReceiveBufferSize);
    dataReceived = Encoding.ASCII.GetString(buffer, 0, bytesRead);
    client.Close();

    return true;
  } catch (Exception ex) {
    Logger.LogException(ex);
    return false;
  }
}
```

Kod 5.3. Primanje i validacija TCP poruke

Unutar te metode čeka se poruka na *Port* 5000. Kada poruka stigne, radi se validacija IPv4 adrese *End point* klijenta, te ako je vrijednost validna, ista se sprema u izlazni parametar `ipAddress`. Nadalje, otvara se *NetworkStream* i čita poruka koja se prima kao polje okteta te se potom dekodira preko ASCII kodiranja u *string* i sprema u izlazni parametar `dataReceived`. Metoda ovisno o uspjehu primanja i validacije poruke vraća *boolean* vrijednost.

S te dvije dobivene vrijednosti pristupa se bazi podataka i preko rezultata metode *GetReaderId* provjerava da li u tablici *RfIdReader* postoji RFID čitač s dobivenom IP adresom, a preko metode *GetUserId* provjerava da li u tablici *RfIdUsers* postoji korisnik čiji podaci na RFID kartici odgovaraju dobivenom podatku `dataReceived` . U slučaju da su oba uvjeta zadovoljena, u tablicu *EntryLog* dodaje se novi redak s podacima `idReader`, `idUser` i trenutnom vremenu (što otprilike odgovara vremenu kada je korisnik očitao svoju RFID karticu). Ponovno se otvara TCP *NetworkStream* i IP adresi s koje je došla poruka, šalje se odgovor o uspješnom upisu. Ovakvim provjerama ograničena je uspješna prijava isključivo korisnika s RFID karticama koje su unesene u bazu podataka, isključivo preko RFID čitača koji su također uneseni u bazu podataka.

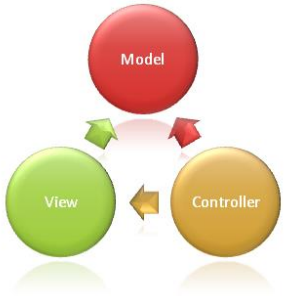
5.4. .NET Web aplikacija

Klijentska aplikacija preko *Microsoft Entity Framework*-a pristupa bazi podataka u kojoj se nalaze zapisi o ulasku, podaci o korisnicima, RFID karticama i RFID čitačima. Kao uzorak dizajna (engl. *design pattern*) korišten je MVC (*model – view – controller*) koji će biti opisan u nastavku poglavlja.

5.4.1. MVC *design pattern i* ASP.NET MVC

Model – View – Controller (MVC) arhitektura trenutno je najčešće korišten *design pattern* u izradi web aplikacija (iako se ne primjenjuje nužno samo na web aplikacije). MVC arhitekturalno dijeli aplikaciju na tri glavne grupe:

- modele (engl. *models*)
- poglede (engl. *views*)
- kontrolere (engl. *controllers*)



Slika 5.4. Odnos između MVC komponenti[10]

Takva podjela pomaže pri raspodjeli odgovornosti (engl. *separation of concerns*). Zahtjevi korisnika preusmjereni su na kontroler čija je zadaća rad s modelom na način da dohvaća rezultate upita (engl. *queries*) i izvršava korisničke zahtjeve. Kontroler odabire pogled koji se prikazuje korisniku i pruža mu sve podatke modela koji su mu potrebni. Slika (Slika 5.4.) prikazuje odnos između navedene tri komponente. Ovakva podjela odgovornosti pojednostavljuje kodiranje, testiranje i održavanje jer svaka komponenta ima samo jedan posao. Primjer je korisničko sučelje koje se mijenja češće od poslovne logike. U slučaju da se prezentacijski kod i poslovna logika kombiniraju u jednom objektu, objekt koji sadrži poslovnu logiku mora se mijenjati svaki put kad se promijeni korisničko sučelje (što zahtjeva ponovno testiranje logike nakon svake promjene u korisničkom sučelju).

Zadaća modela u MVC arhitekturi je izvršavanje operacija i poslovne logike. Također, model reprezentira podatak (odnosno klasu). Budući da se tablice iz baze podataka unutar koda reprezentiraju kao klasa, *Microsoft entity framework* savršeno se nadopunjuje s korištenjem takvog modela.

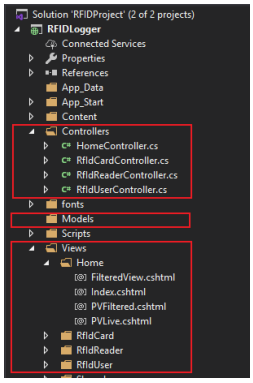
Pogled (engl. *View*) služi da korisniku na *user friendly* način prikaže podatke i omogući mu izvršavanje akcija nad njima. To je jedini dio sustava koji je korisniku stvarno vidljiv. U ASP.NET koristi se *Razor view engine* koji omogućava pisanje C# koda direktno u HTML. Iako nekad takvo rješenje izgleda primamljivo i jednostavnije, logika unutar samih pogleda trebala bi biti minimalna i fokusirana isključivo na prezentaciju sadržaja.

Kontroler je komponenta koja reagira na korisničku interakciju. Radi s modelom i odabire pogled koji će se prikazati korisniku. Kontroler je početna ulazna točka i ponaša se kao usmjerivač (engl. *router*) – uzima model i odabire pogled koji će se koristiti i prosljeđuje

[10] https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-3.1

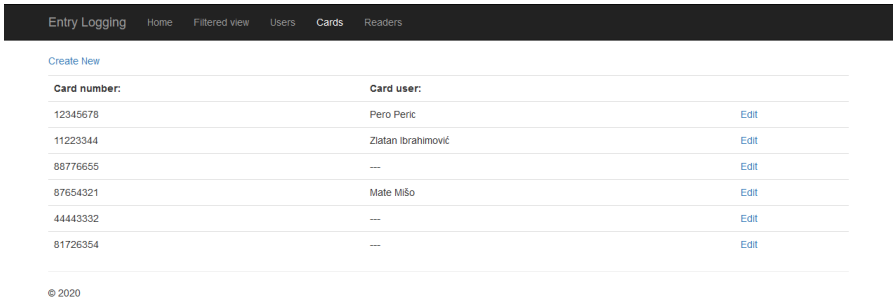
model pogledu. Slično kao i kod pogleda, logika koja bi se trebala nalaziti u kontroleru trebala bi biti vezana isključivo za prosljeđivanje modela i odabir pogleda.

Kodiranje po konvenciji (engl. *convention over configuration*) još je jedan bitan aspekt ASP.NET okvira. Uvelike olakšava i ubrzava razvoj aplikacija jer se programer (ako se drži konvencija) ne treba brinuti o nekim osnovnim aspektima aplikacije, već to programski okvir (engl. *framemork*) odrađuje sam. Na slici (Slika 5.5.) prikazana je projektna struktura aplikacije. Programski okvir pri samom kreiranju MVC .NET projekta kreira i potrebne mape. Tako se očekuje da svi kontroleri budu unutar mape *Controllers*, da njihove klase imaju naziv *NekiController.cs* i da pogledi za pojedini kontroler budu unutar mape *Views/NazivKontrolera/*. Na taj način kontroler će znati gdje potražiti neki pogled i odmah „pogoditi“ URI tipa: http://localhost:49342/RfIdUser/Edit/1.



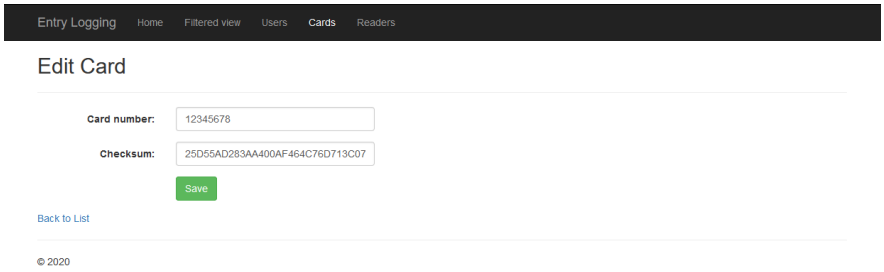
Slika 5.5. Projektna struktura u .NET MVC aplikaciji

5.4.2. RfId kartica



Slika 5.6. Pogled s listom RFID kartica

Na slici (Slika 5.6.) prikazan je pogled koji prikazuje sve RFID kartice i njihove vlasnike (u slučaju da kartica ima vlasnika). Unutar tog pogleda omogućeno je dodavanje nove kartice i uređivanje postojećih RFID kartica (Slika 5.7.).



Slika 5.7. Pogled za uređivanje podataka o RFID kartici

U *RfIdCardController* klasi koriste se metode koje pozivaju čvrsto tipizirane poglede koji omogućuju CRUD operacije nad *RfIdReader* modelom. U kodu (Kod 5.4.) prikazano je pozivanje *Index* pogleda koji prikazuje listu svih RFID kartica. U pogled je proslijeđen i popis korisnika da bi se RFID kartica unutar pogleda mogla povezati s vlasnikom (Kod 5.5.). Korištenjem ključne riječi ***using*** osigurano je da se objekt `db` (koji otvara konekciju prema bazi podataka) nakon korištenja prirodno očisti (zatvori konekciju). Kako se ta opcija mogla koristiti, klasa treba implementirati *IDisposable* sučelje (engl. *interface*). Prilikom prestanka korištenja klase automatski se poziva implementacija *Dispose* metode (slično kao kod destruktora u C++).

```
public ActionResult Index() {
  using (RFID_DBEntities db = new RFID_DBEntities()) {
    ViewBag.Users = db.RfIdUser.ToList();
    return View(db.RfIdCard);
  }
}
```

Kod 5.4. Poziv *Index* pogleda za prikaz svih RFID kartica

```
@foreach (var item in Model) {
  <tr>
    <td>@Html.DisplayFor(modelItem => item.SerialNumber)</td>
    <td>@{
      List<RfIdUser> users = ViewBag.Users;
```

```
      RfIdUser rfIdUser = users
        .Where(x =>
          x.RfIdCard.SerialNumber == item.SerialNumber)
        .FirstOrDefault();
      if (rfIdUser == null) {
        <text>---</text> }
      else {
        <text>@rfIdUser.User.DisplayedName</text> }
    }
    </td>
    <td>@Html.ActionLink("Edit", "Edit",
           new { id = item.IDCard })
    </td>
  </tr>
}
```

Kod 5.5. Povezivanje RFID kartice s korisnikom unutar pogleda

Kod *Create* i *Edit* poziva, koji služe dodavanju novih RFID kartica, kod je sličan. Sastoji se od HTTP *post* i HTTP *get* poziva koji programski okvir bira ovisno o tome dohvaća li se stranica (HTTP *get*) ili stranica šalje podatke (HTTP *post*). Smjer poziva označen je s anotacijom iznad deklaracije funkcije.

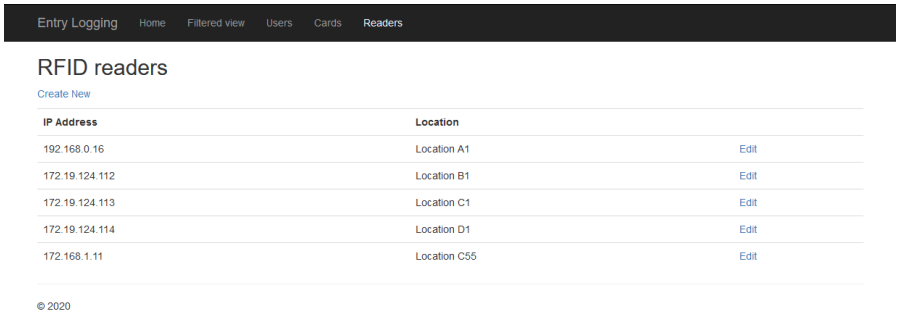
```
public ActionResult Create() {
  return View("CreateCard");
}

[HttpPost]
public ActionResult Create(RfIdCard rfIdCard) {
  try {
    using (RFID_DBEntities db = new RFID_DBEntities()) {
      db.RfIdCard.Add(new RfIdCard {
        SerialNumber = rfIdCard.SerialNumber,
        Checksum = rfIdCard.Checksum });
      db.SaveChanges();

      return RedirectToAction("Index");
    }
  } catch {
    return View("Error");
  } }
```

Kod 5.6. HTTP *get* i HTTP *post* metode za poziv pogleda *EditCard*

Iz koda (Kod 5.6.) vidljivo je kako je u HTTP *post* metodi sav kod stavljen unutar *try – catch* bloka. Time je osigurano da se u slučaju greške prikaže generički *Error* pogled koji korisnika obavještava da je došlo do greške.

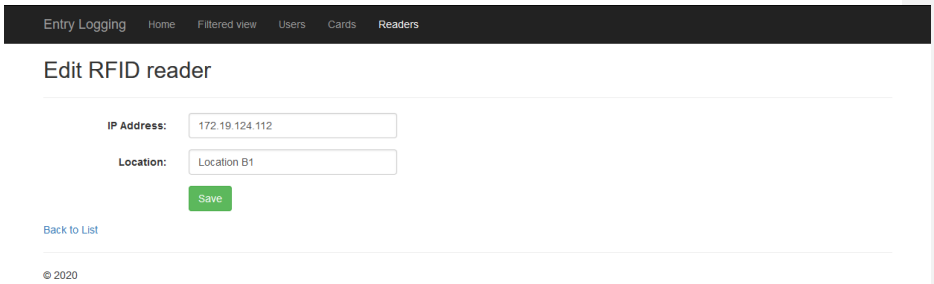
5.4.3. RfId čitač

Slično kao i kod RFID kartice, u pogledu */RfIdCard/Index.html* prikazuje se lista svih unesenih RFID čitača (Slika 5.8.).



Slika 5.8. Pogled s prikazom svih RFID čitača

U pogledu se nalaze poveznice za kreiranje novog RFID čitača i promjenu podataka o postojećim (Slika 5.9.).



Slika 5.9. Pogled za uređivanje podataka o RFID čitaču

Slično kao i kod *Create* metode, unutar *RfIdReaderController* klase nalaze se dvije *ActionResult Edit* metode (HTTP *get* i HTTP *post*). HTTP *get* metoda prima cjelobrojni

parametar *id* pomoću kojeg preko lambda izraza pretražuje tablicu *RfIdReader* i uspoređuje ga s poljem *IDReader* iz tablice (Kod 5.7.).

```
public ActionResult Edit(int id) {
  using (RFID_DBEntities db = new RFID_DBEntities())
    return View("EditReader", db.RfIdReader
      .Where(x => x.IDReader == id).FirstOrDefault());
}
```

Kod 5.7. Implementacija HTTP *get* metode za uređivanje podataka o RFID čitaču

U HTTP *post* metodi kao parametri primaju se cijeli broj *id* (koji odgovara parametru *IDReader* iz tablice *RfIdReader*) i instanca objekta *RfIdReader* s parametrima ispunjenim unutar forme pogleda (Kod 5.8.).

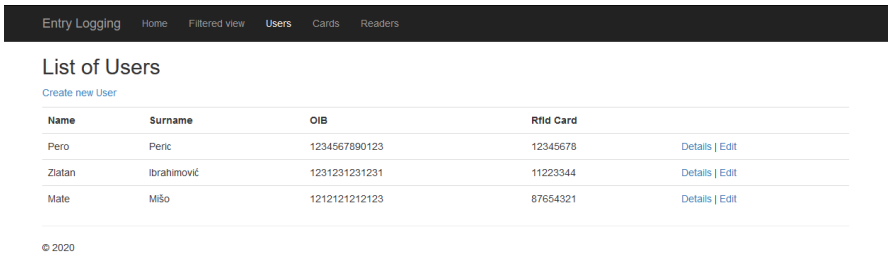
```
[HttpPost]
public ActionResult Edit(int id, RfIdReader rfIdReader) {
  try {
    using (RFID_DBEntities db = new RFID_DBEntities()) {
      RfIdReader rfIdReaderToUpdate = db.RfIdReader
       .Where(x => x.IDReader == id).FirstOrDefault();

      rfIdReaderToUpdate.IPAddress = rfIdReader.IPAddress;
      rfIdReaderToUpdate.Location = rfIdReader.Location;
      db.SaveChanges();

      return RedirectToAction("Index");
    }
  } catch {
    return View("Error");
  }
}
```

Kod 5.8. Implementacija HTTP *post* metode za uređivanje podataka o RFID čitaču

Nakon što se na isti način kao i kod HTTP *get* metode u bazi podataka pronađe odgovarajući objekt, kao vrijednosti postavljaju se dobiveni podaci i promjene se spremaju u bazu podataka. Kako je cijela implementacija stavljena unutar *try – catch* bloka, ako dođe do greške pri komunikaciji s bazom podataka (ili neke druge greške), korisnika se preusmjerava na stranicu koja ga obavještava o grešci.

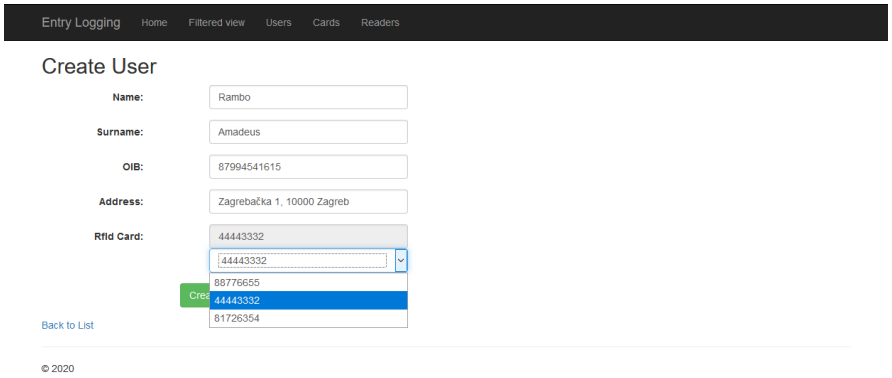
5.4.4. Rfid korisnik

Kao i kod prethodnih pogleda, u pogledu */RfIdUser/Index.html* prikazuju se svi postojeći korisnici. Model koji je vezan uz ovu tablicu je *RfIdUser* jer on sadrži poveznice na klasu *User* i na klasu *RfIdCard*. Izgled poziva može se vidjeti na slici (Slika 5.10.). Unutar pogleda nalaze se poveznice na pogled za kreiranje novog korisnika (Slika 5.11), pogled s detaljima o korisniku i pogled za uređivanje korisnika.



Slika 5.10. Pogled s prikazom svih korisnika i pripadajućih RFID kartica

Kod pogleda */RfIdUser/CreateUser.html* (Slika 5.11.) unose se podaci o novom korisniku. Za odabir RFID kartice postavljen je padajući izbornik za bolje korisnički iskustvo (ista stvar napravljena je kod pogleda za uređivanje podataka o korisniku).



Slika 5.11. Pogled za kreiranje novog korisnika

Da bi se onemogućilo da se dvama različitim korisnicima dodjeli ista RFID kartica, u HTTP *get ActionResult* metodi, lambda izrazom filtrirane su sve RFID kartice koje trenutno nisu

dodijeljene. Rezultat tog upita sprema se u *ViewBag.unusedCards* i šalje prema pogledu. Implementacija metode prikazana je u nastavku (Kod 5.9.).

```
public ActionResult Create() {
  List<RfIdCard> allCards = db.RfIdCard.ToList();
  List<RfIdUser> userList = db.RfIdUser.ToList();
  List<RfIdCard> unusedCards = allCards
    .Where(x => !userList.Any(y => y.RfIdCard.IDCard == x.IDCard))
    .ToList();
  ViewBag.UnusedCards = unusedCards;
  return View("CreateUser");
}
```

Kod 5.9. Implementacija HTTP *get* metode za kreiranje novog korisnika

Unutar samog pogleda dodan je *TextBox* za uređivanje serijskog broja RFID kartice i postavljen je atribut *@readonly* da bi se korisnika onemogućilo da napravi krivi unos. Budući da podaci o neiskorištenim RFID karticama nisu čvrsto vezani uz model koji se koristi unutar pogleda, ti podaci uzeti su iz kontejnera *ViewBag.UnusedCards* i koristeći *RazorEngine* dodani u padajući izbornik *ddlCardNumber* (Kod 5.10.).

```
<div class="form-group">
  <div class="col-lg-10">
    <label class="control-label col-md-2">RfId Card:</label>
    @Html.EditorFor(model => model.RfIdCard.SerialNumber,
       new { htmlAttributes = new { @id = "txtCardNumber",
         @class = "form-control form-inline col-md-offset-3",
         @readonly = "readonly" } })
    @Html.ValidationMessageFor(model => model.CardId, "",
      new { @class = "text-danger" })
    @{
      List<SelectListItem> selectListItems = new List<SelectListItem>();
      List<RfIdCard> unusedCards = ViewBag.UnusedCards;
      unusedCards.ForEach(x =>
        selectListItems.Add(
          new SelectListItem { Text = x.SerialNumber,
                               Value = x.IDCard.ToString() }));
      @Html.DropDownList("ddlCardNumber", selectListItems,
        new { @id = "ddlCardNumber",
              @class = "form-control form-inline col-lg-offset-3" });
    }
  </div>
```

```
</div>
```

Kod 5.10. HTML i *Razor* implementacija padajućeg izbornika

Za dinamičko prenošenje vrijednosti iz padajućeg izbornika u *text box* koristi se kratki *JavaScript* kod (Kod 5.11.).

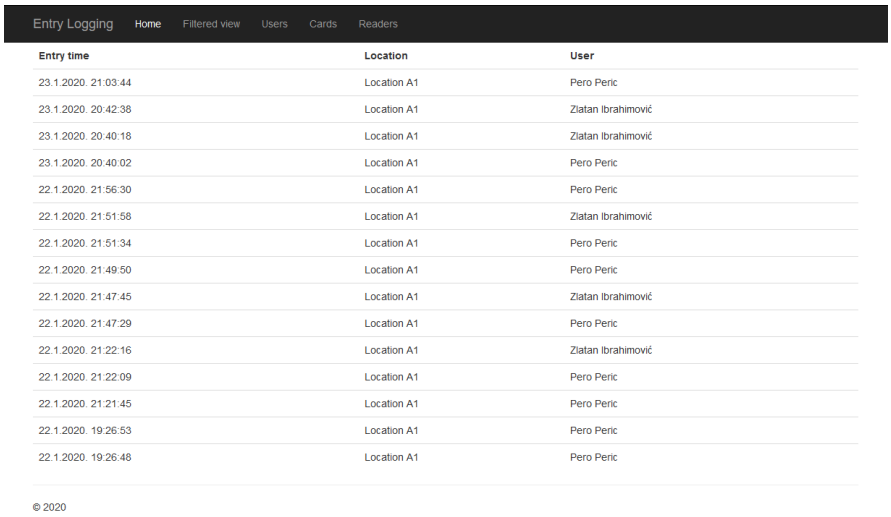
```
<script type="text/javascript">
  $('#ddlCardNumber').change(function (e) {
    var text = $('#ddlCardNumber option:selected').text();
    $('#txtCardNumber').val(text);
  });
</script>
```

Kod 5.11. JavaScript kod za dinamičko prenošenje vrijednosti iz padajućeg menija

5.4.5. Prikaz zapisa o ulasku

Za prikaz podataka o ulasku korištena su dva pogleda. Prvi pogled */Home/Index*.html prikazuje zapise o ulascima u realnom vremenu (Slika 5.12.). Prikazuje se vrijeme ulaska, lokacija RFID čitača i osoba koja se prijavila.

Commented [SF1]: o ulasku?



Slika 5.12. Prikaz podataka u ulasku u realnom vremenu

Kao osnovni pogled koristi se netipizirani pogled */Home/Index.html* koji se poziva s *ActionResult* metodom *Index():*

```
public ActionResult Index() {
```

```
    return View();
  }
```

Za prikaz podataka kreiran je tipizirani parcijalni pogled *PVLive.html* koji prima listu modela *EntryLog*. Pogledu se šalje prvih stotinu zapisa iz baze podataka, kronološki poredano prema vremenu ulaska, od novijih prema starijima (Kod 5.12.).

```
public ActionResult PVLive() {
  IOrderedQueryable<EntryLog> entryLogs =
    db.EntryLog.OrderByDescending(x => x.IDEntry);

  if (entryLogs.Count() < 100)
    return PartialView(entryLogs.ToList());
  else
    return PartialView(entryLogs.ToList().GetRange(0, 100));
}
```

Kod 5.12. *ActionResult* metoda za poziv parcijalnog pogleda */Home/PVLive.html*

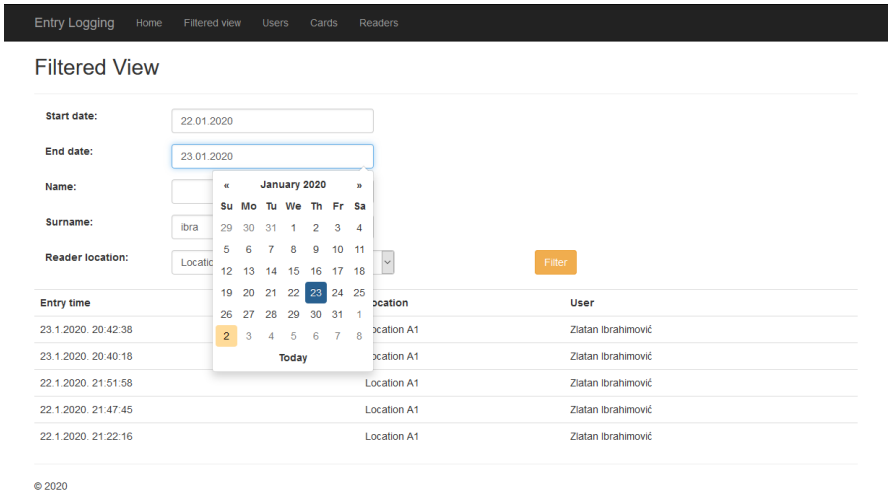
Unutar pogleda */Home/Index.html* nalazi se *JavaScript* kod koji koristi rekurzivni AJAX poziv koji svake sekunde osvježava podatke. Na taj način osigurano je da se svake sekunde ne učitava cijela stranica, nego samo parcijalni pogled koji sadrži podatke o ulascima (Kod 5.13.).

```
<div id="logContainer">
  Loading data ...
</div>

<script type="text/javascript">
  $(document).ready(function () {
    function refreshPartial() {
      setTimeout(function () {
        $.ajax({
          type: "GET",
          url: "/Home/PVLive",
          success: function (data) { $("#logContainer").html(data) }
        });

        refreshPartial();
      }, 1000);
    }

    refreshPartial();
    });
```

```
</script>
```

Kod 5.13. HTML i *JavaScript* kod s AJAX pozivom parcijalnog pogleda

Također, ostavljena je mogućnost prikaza filtriranih zapisa s pogledom */Home/FilteredView.html*. Filtriranje se može raditi prema rasponu datuma, imenu i prezimenu korisnika ili lokaciji RFID čitača (Slika 5.13.).



Slika 5.13. Prikaz filtriranih podataka u ulasku

Kao i kod prethodnog pogleda, za prikaz podataka koristi se parcijalni pogled */Home/PVFiltered.html*. Radi ugodnijeg korisničkog iskustva za prikaz datuma koristi se *Bootstrap* komponenta *Datepicker*. Pritiskom na tipku *Filter* pokreće se AJAX poziv koji predaje parametre forme (Kod 5.14.).

```
<div id="containerData">
</div>

<script>
  $('#btnFilter').click(function (e) {
    $.ajax({
      type: "GET",
      url: "/Home/PVFiltered",
      data: {
        startDate: $('#txtStartDate').val(),
        endDate: $('#txtEndDate').val(),
        name: $('#txtName').val(),
        surname: $('#txtSurname').val(),
```

```
        location: $('#txtLocation').val()
      },
      success: function (data) { $('#containerData').html(data) }
    })
  });
  </script>
```

Kod 5.14. AJAX poziv parcijalnog pogleda *PVFiltered*

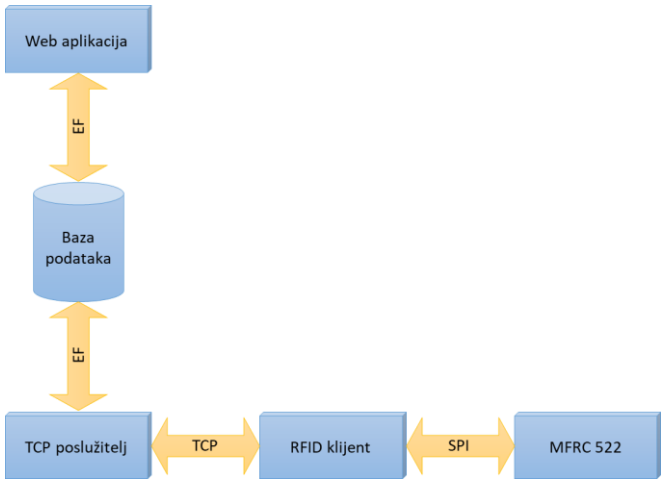
Unutar *ActionResult PVFiltered* metode primaju se parametri poslani iz AJAX poziva te nakon provjere integriteta podataka (u slučaju da se vrijednost ne primi ili je prazna, podatak se ne uzima kao kriterij filtriranja) dobiveni rezultati šalju se kao model u parcijalni pogled *PVFiltered*. (Kod 5.15.)

```
public ActionResult PVFiltered(string startDate, string endDate, string name,
                               string surname, string location) {

  DateTime dtStart = GetDateFromString(startDate, true);
  DateTime dtEnd = GetDateFromString(endDate, false);

  IQueryable<EntryLog> entryLogs = db.EntryLog
    .Where(x => x.EntryTime >= dtStart && x.EntryTime <= dtEnd);

  if (name != null && name != "")
    entryLogs = entryLogs.Where(x =>
      x.RfIdUser.User.Name.ToLower().Contains(name.Trim().ToLower()));

  if (surname != null && surname != "")
    entryLogs = entryLogs.Where(x =>
      x.RfIdUser.User.Surname.ToLower().Contains(surname.Trim().ToLower()));

  if (location != null && location != "")
    entryLogs = entryLogs.Where(x =>
      x.RfIdReader.Location.ToLower().Contains(location.Trim().ToLower()));

  return PartialView(entryLogs.OrderByDescending(x => x.IDEntry).ToList());
}
```

Kod 5.15. *ActionResult* metoda za poziv parcijalnog pogleda */Home/PVFiltered.html*

6. Interoperabilnost sustava

Kako su entiteti sustava zasebno, čak i fizički odvojene cjeline potrebno je koristiti različite protokole kako bi entiteti međusobno mogli komunicirati i time omogućiti funkcioniranje sustava kao cjeline (Slika 6.1.).



Slika 6.1. Komunikacijski protokoli između entiteta sustava

U ovom sustavu entiteti komuniciraju preko sljedećih sučelja, odnosno protokola:

- SPI
- TCP
- *Entity Framework* / TCP

SPI (engl. *Serial Peripherial Interface*) je specifikacija sinkronog serijskog komunikacijskog sučelja koje se koristi za komunikaciju na kratke udaljenosti, najčešće u ugrađenim sustavima. Razvijen je sredinom 1980-ih od strane kompanije Motorola i nakon toga je postao često korišteni standard. Najčešće se primjenjuje kod sigurnih digitalnih kartica (u ovom slučaju RFID kartica) i LCD zaslonima. SPI uređaji komuniciraju u *full duplex* modu koristeći *master – slave* arhitekturu s jednim *master* uređajem. U ovom sustavu u *master* ulozi je Raspberry Pi uređaj, koji komunicira s jednim ili više MRFC522 RFID čitača koji su u *slave* ulozi.[11]

[11] https://en.wikipedia.org/wiki/Serial_Peripheral_Interface

Za komunikaciju između aplikacije TCP poslužitelja i RFID klijenta koristi se TCP protokol. TCP protokol je jedan od osnovnih protokola unutar IP grupe protokola. Glavne značajke ovog protokola su da omogućuje pouzdanu i sortiranu isporuku toka okteta (engl. *byte stream*) s provjerom grešaka. Za razliku od UDP protokola, TCP garantira cjelovitu isporuku podataka, no zbog trostrukog rukovanja (engl. *three way handsake*), mogućeg ponovnog slanja paketa, sortiranja i provjere grešaka produžuje latenciju. Najčešće se koristi kod aplikacijskih protokola na Internetu (HTTP), protokola za razmjenu elektroničke pošte i protokola za udaljeni rad na računalu (telnet i SSH).

Komunikacija između baze podataka i web aplikacije, odnosno baze podataka i aplikacije TCP poslužitelja radi se preko Microsoft *Entity Frameworka*. Iako *Entity Framework* sam po sebi nije protokol, on modele baze podataka i aplikacije reprezentira u shemama temeljenim na XML-u. *Entity Framework* generira tri datoteke:

- CSDL – definicija konceptualnog modela (proširenje datoteke .csdl)
- SSDL – definicija modela pohrane (proširenje datoteke .ssdl)
- MSL – specifikacija mapiranja, odnosno preslikavanja između CDSL i SSDL modela (proširenje datoteke .msl)

Preko te tri datoteke aplikacijski kod (LINQ) pretvara se u SQL upit prema bazi podataka, koja potom vraća tražene podatke. Sama komunikacija između aplikacije i baze podataka radi preko TCP protokola.

7. Usporedba s postojećim industrijskim rješenjem

U ovom poglavlju usporedit će se funkcionalnost, prednosti i nedostaci s već postojećim rješenjem. Za usporedbu je korišten MENGQI-CONTROL TCP/IP *Four Door Entry Access Control Panel Kit* (Slika 7.1.).

Slika 7.1. Izgled postojećeg industrijskog rješenja

Gotovi sustav sastoji se od četiri RFID čitača, četiri elektronske brave, sto RFID kartica i upravljačke jedinice. Isto kao i naš sustav, za komunikaciju s čitačima koristi TCP protokol i zahtjeva da svaki čitač bude spojen u *ethernet* mrežu.

Postojeći sustav sličan je funkcionalnošću, no zbog korištenja prilagođenog hardvera ograničen je na 20 000 korisnika i 100 000 zapisa. Kako se u implementaciji ovog rada baza podataka nalazi na poslužitelju i ne postoji centralna kontrolna, odnosno upravljačka jedinica, količina mogućih korisnika i zapisa u praksi je mnogo veća.

Također, zbog korištenja prilagođenog hardvera, postojeći sustav je limitiran na četiri čitača, dok je predmetni sustav u potpunosti skalabilan - novi čitač se može dodati u sustav i sve što je potrebno je pristup *ethernet* mreži i izvor napajanja.

Cjenovno, postojeći sustav košta oko 500$, dok je cijena predmetnog sustava oko 30$ po čitaču (cijena sustava mogla bi se i dodatno smanjiti da se umjesto Raspberry Pi kao udaljeni klijent koristi jeftinije rješenje, na primjer Arduino). Popratni dijelovi kao što su RFID kartice i elektronske brave lako su dobavljive i cjenovno izuzetno pristupačne.

Zaključak

Završni proizvod ovog rada je funkcionalni sustav za automatsko praćenje toka ljudi kroz štićene prolaze. Sama implementacija sustava napravljena je s već gotovim, cjenovno pristupačnim hardverskim komponentama i korištenjem različitih postojećih široko primjenjivih tehnologija.

U radu je prikazano kako s dobro osmišljenom i postavljenom arhitekturom sustava, gdje svaki sloj izvršava svoju zadaću možemo smanjiti probleme kod budućih promjena te olakšati održavanje i rješavanje problema u sustavu.

Budući da se sustav sastoji od više entiteta, što neminovno izaziva primjenu interoperabilnosti unutar sustava, česti problemi i komplikacije koji se u tom slučaju pojavljuju uspješno su mitigirani.

Komunikacija s hardverom jednostavno je riješena korištenjem gotove i besplatne programske biblioteke, na čiju implementaciju bi se inače potrošilo najviše vremena. Budući da je Raspberry Pi cjenovno prihvatljiv i u čestoj upotrebi, postoji velika baza projekata i *open-source* koda, koji se može iskoristiti i lako prilagoditi specifičnim zahtjevima.

Također, pokazano je kako koristeći Microsoft ASP.NET programske okvire i alate, jednostavno i efikasno možemo povezati aplikacijske, prezentacijske i podatkovne slojeve sustava.

Na tržištu sličnih industrijskih rješenja uglavnom postoje gotovi i skupi sustavi. Završni rad pokazuje kako se ovaj sustav može povećati i prilagoditi zahtjevima korisnika, i kako se ovo rješenje, iako je cjeloviti i funkcionalni sustav, može koristiti kao polazišna točka te jednostavno i cjenovno prihvatljivo povećati.

Rješenje se uz pojedine preinake u praksi može koristiti za druge stvari osim praćenja kretanja ljudi (na primjer, praćenje predmeta) i jednostavno se proširiti s dodatnim funkcijama osim otvaranja vrata. Sustav se lako možemo prilagoditi i *rental* poduzećima koje bi mogle pratiti kada je koji proizvod napustio skladište i kada se vratio.

Popis kratica

3NF	*Third Normal Form*	treća normalna forma
AJAX	*Asynchronous JavaScript+XML*	asinkroni JavaScript i XML
ASCII	*American Standard Code for Information Interchange*	
	Američki standardni znakovnik za razmjenu informacija	
CRUD	*Create Read Update Delete*	stvori čitaj osvježi obriši
HDMI	*High-Definition Multimedia Interface*	multimedijalni sklop visoke definicije
HTML	*Hyper Text Markup Language*	prezentacijski označni jezik
HTTP	*Hyper Text Transfer Protocol*	protokol za prijenos označnog jezika
LINQ	*Language Integrated Query*	integrirani upiti u programskom jeziku
MVC	*Model View Controller*	model pogled kontroler
RFID	*Radio Frequency Identification*	radio frekvencijska identifikacija
SPI	*Serial Peripherial Interface*	serijsko periferno sučelje
SSH	*Secure Shell*	protokol za sigurnim kanalom
SQL	*Structured Query Language*	strukturirani upitni jezik
TCP	*Transmission Control Protocol*	protokol za kontrolu prijenosa
URI	*Uniformed Resource Identifier*	jednoznačni identifikator resursa
USB	*Universal Serial Bus*	univerzalna serijska sabirnica
XML	*Extensible Markup Language*	proširivi označni jezik

Popis slika

Popis tablica

Popis kôdova

Literatura

[1] CHETOUANE, *An Overview on RFID Technology Instruction and Application*; *IFAC-PapersOnLine Volume 48, Issue 3*, (2015) https://www.sciencedirect.com/science/article/pii/S240589631500350X

[2] DUROCA, TEDJINI, *RFID: A key technology for Humanity; Comptes Rendus Physique, Volume 19, Issues 1–2*, (2018) https://www.sciencedirect.com/science/article/pii/S1631070518300124

[3] NXP SEMICONDUCTORS, *MFRC522 Standard performance MIFARE and NTAG frontend*, (2016) https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf

[4] RASPBERRY PI FOUNDATION, *Raspberry Pi 4 B documentation*, (2018); https://github.com/raspberrypi/documentation