Kućna meteorološka stanica s mobilnom aplikacijom

Petričević, Mario

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

KUĆNA METEOROLOŠKA STANICA S MOBILNOM APLIKACIJOM

Završni rad

Mario Petričević

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 28.08.2020.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime studenta:	Mario Petričević
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4115, 25.09.2019.
OIB studenta:	78259349457
Mentor:	Doc.dr.sc. Mirko Köhler
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Kućna meteorološka stanica s mobilnom aplikacijom
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 2 razina
Datum prijedloga ocjene mentora:	28.08.2020.
Datum potvrde ocjene Odbora:	09.09.2020.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 09.09.2020.

Ime i prezime studenta:	Mario Petričević
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4115, 25.09.2019.
Turnitin podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Kućna meteorološka stanica s mobilnom aplikacijom**

izrađen pod vodstvom mentora Doc.dr.sc. Mirko Köhler

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA

o odobrenju za pohranu i objavu ocjenskog rada

kojom ja __Mario Petričević_____, OIB: __78259349457_____, student/ica Fakulteta elektrotehnike,
računarstva i informacijskih tehnologija Osijek na studiju _Preddiplomski sveučilišni studij Računarstvo _, kao autor/ica ocjenskog rada pod naslovom: _Kućna meteorološka stanica s mobilnom aplikacijom**,**
dajem odobrenje da se, bez naknade, trajno pohrani moj ocjenski rad u javno dostupnom digitalnom repozitoriju ustanove Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek i Sveučilišta te u javnoj internetskoj bazi radova Nacionalne i sveučilišne knjižnice u Zagrebu, sukladno obvezi iz odredbe članka 83. stavka 11. *Zakona o znanstvenoj djelatnosti i visokom obrazovanju* (NN 123/03, 198/03, 105/04, 174/04, 02/07, 46/07, 45/09, 63/11, 94/13, 139/13, 101/14, 60/15).
Potvrđujem da je za pohranu dostavljena završna verzija obranjenog i dovršenog ocjenskog rada. Ovom izjavom, kao autor/ica ocjenskog rada dajem odobrenje i da se moj ocjenski rad, bez naknade, trajno javno objavi i besplatno učini dostupnim:

a) široj javnosti

b) studentima/icama i djelatnicima/ama ustanove

c) široj javnosti, ali nakon proteka 6 / 12 / 24 mjeseci (zaokružite odgovarajući broj mjeseci).

**U slučaju potrebe dodatnog ograničavanja pristupa Vašem ocjenskom radu, podnosi se obrazloženi zahtjev nadležnom tijelu Ustanove.*

__Osijek, 09.09.2020._____ _____

(mjesto i datum) (vlastoručni potpis studenta/ice)

SADRŽAJ

1. UVOD

Meteorološka stanica je uređaj za mjerenje i predviđanje vremenskih prilika na određenoj lokaciji. Razvoj tehnologije u posljednja dva desetljeća pridonio je razvoju sve kvalitetnijih i boljih meteoroloških stanica. Tako danas postoje složene meteorološke stanice koje prate vremenske uvjete na širokom području. To su najčešće cijela postrojenja koja iznimno točno i precizno prate i predviđaju vremenske prilike. S druge strane, postoje i jednostavne kućne meteorološke stanice koje se mogu kupiti u slobodnoj prodaji te se najčešće koriste u krugu kućanstva. Najčešće se koriste za mjerenje temperature, vlage, tlaka, jačine puhanja vjetra; pružaju informacije o vremenskim uvjetima i sl.. Zadatak ovoga završnog rada bio je upravo izrada takve meteorološke stanice.

Ideja je bila razviti kućnu meteorološku stanicu te aplikaciju uz pomoć koje će se pratiti vremenski uvjeti na toj stanici. Zbog same veličine kućnih meteoroloških stanica kupljenih u maloprodaji te njihove nekompaktnosti i nemogućnosti da ih stalno nosimo sa sobom, ideja se pokazala korisnom budući da stanje na njoj možemo jednostavno pratiti odlaskom na aplikaciju na vlastitom mobilnom uređaju.

Kao sustav za mjerenje vremenskih prilika korišten je Raspberry Pi s Raspbian operacijskim sustavom, inačicom Linuxa.

Kako bi se uspješno koristio Raspberry Pi potrebno je poznavanje Linux komandi te poznavanje rasporeda pinova na koje se postavljaju senzori. Senzor korišten za mjerenje temperature i vlage je DHT11. Senzor za mjerenje tlaka zraka je BMP085. Programski jezici korišteni za očitavanje podataka sa senzora te rad s bazom podataka u koju se spremaju podaci su: Python, PHP, SQL. Za bazu podataka korištena je PHPMyAdmin baza na lokalnom računalu. Za izradu aplikacije korišten je Android Studio te programski jezik Java za pisanje koda. Struktura rada je sljedeća: u trećem poglavlju opisan je rad na Raspberry Pi-u te tehnologije korištene za postavljanje senzora i rad s njima; četvrtim poglavljem dano je programsko rješenje meteorološke stanice, a peto poglavlje obuhvaća izradu Android aplikacije. Na kraju je dan zaključak rada.

1.1. Zadatak završnog rada

Zadatak ovog završnog rada je uz pomoć navedenih tehnologija napraviti samostalni sustav za praćenje vremenskih prilika. Uz to, potrebno je razviti mobilnu aplikaciju kojoj će se slati podatci o vremenu u redovitim vremenskim intervalima te na korisnikov zahtjev. Također, potrebno je pružiti informacije o vremenskim prilikama s drugih izvora na internetu te prikazati dobivene podatke grafički u vremenu; unutar aplikacija dati mogućnost dva različita prikaza podataka.

2. USPOREDBA S POSTOJEĆIM RJEŠENJIMA

Kao samostalni projekti, meteorološke stanice uvijek su bile zanimljive i široko obrađene. Tako na internetu pronalazimo mnoga rješenja na istu temu. Najsličnije rješenje [1] također uključuje izradu meteorološke stanice uz Raspberry Pi. Rješenje, osim klasičnih elektroničkih senzora za mjerenje tlaka, temperature, vlage; uključuje i mehaničke senzore. Mehanički senzori služe za mjerenje količine padalina, smjer kretanja vjetra i njegovu jačinu. Ti senzori fizički međudjeluju s okolinom.

Osim na Raspberry Pi-u, prisutna su i mnoga rješenja koja koriste Arduino. Tako npr. [2] prikazuje informacije o vremenskim prilikama na OLED (*engl. Organic light-emitting diode*) ekranu. Projekt [3] koristi *Oregon Scientific* kućnu meteorološku stanicu koja ima mogućnost povezivanja s Raspberry Pi-jem te Android uređajem preko *Bluetootha. CARROT Weather* Android je aplikacija slična aplikaciji napravljenoj u ovom radu[4]. Također, većina Android uređaja već ima ugrađene *widgete* koji pružaju informacije o vremenu[5]. *Vantage Vue* kućna meteorološka stanica gotovo je i kompletno rješenje koje proizvodi tvrtka *Davis Instruments* te se može kupiti na stranici [6].

Za razliku od navedenih rješenja, rješenje u ovom projektu pruža grafički prikaz izmjerenih podataka sa senzora. Također većina rješenja prikazuje podatke na vanjskom ekranu dok rješenje ovog završnog rada prikazuje podatke unutar aplikacije na pametnom telefonu. Ovakav pristup je puno ekonomičniji jer nema dodatnih uređaja i dodatne elektronike. Također, za pristup senzorima potrebno je biti spojen na lokalnu bežičnu mrežu putem pametnog telefona dok kod ostalih rješenja dovoljno je samo biti spojen na Internet. Daljnjim radom navedeno je moguće unaprijediti.

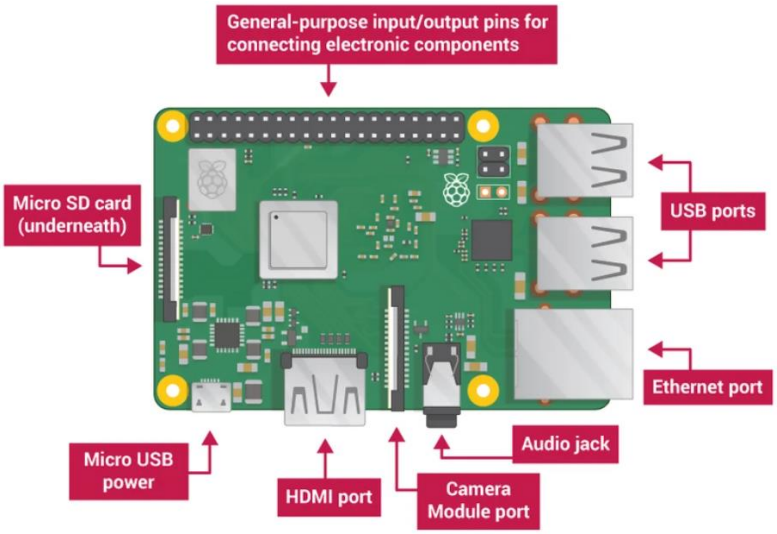
3. RASPBERRY PI

Raspberry Pi sitno je računalo, ne veće od kreditne kartice, koje se nalazi na jednoj matičnoj ploči u cijelosti. Razvila ga je kompanija Raspberry Pi Foundation. Koristi se na širokom području. Od školskih ustanova s ciljem edukacije djece o arhitekturi računala do upotrebe za različita tehnološka rješenja[7]. Za izradu završnog rada korišten je Raspberry Pi 3.

3.1. Tehničke specifikacije i struktura

Raspberry Pi 3 nalazi se na Model B matičnoj ploči. Pokreće ga procesor Broadcom BCM2837. CPU (engl. *Central Processing Unit*) četverojezgreni je ARM Cortex-A53. Radi na taktu od 1.2 GHz. Veličina radne memorije odnosno RAM-a (engl. *Random Acess Memory)* mu je 1 GB. Dolazi s grafičkim čipom VideoCore IV 400MHz. Za razliku od prijašnjih verzija, ovaj model dolazi s mogućnosti bežičnog WiFi te Bluetooth povezivanja. Ima ugrađena 4 USB(engl. *Universal Serial Bus)* porta na koja se mogu spojiti miševi, tipkovnice i sl..

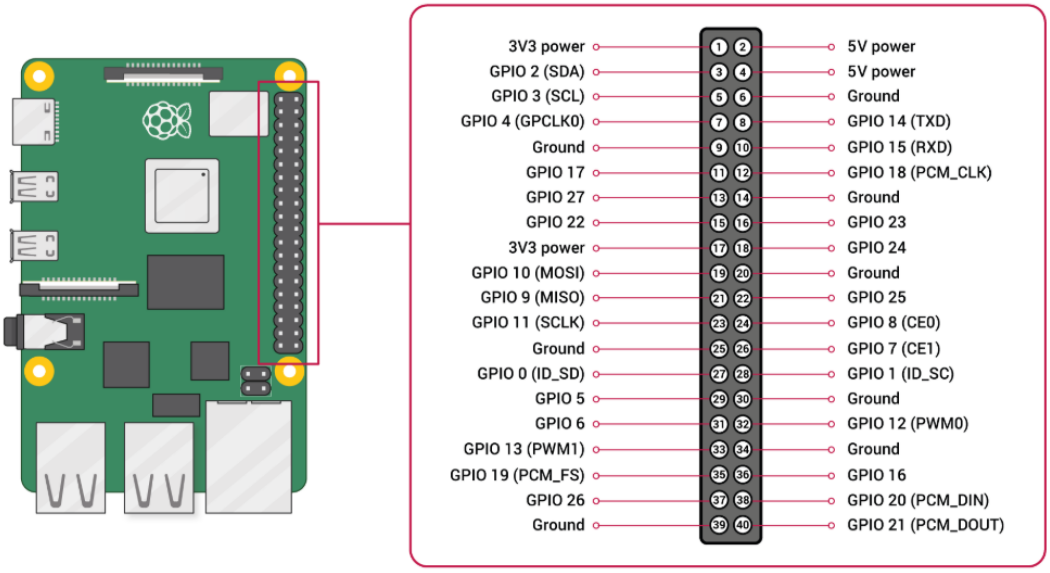
Prikaz sučelja odvija se pomoću ugrađenog HDMI (*engl. High definition multimedia interface)* ulaza (*port*) koji se povezuje pomoću kabela s ekranom. *Audio jack* ulaz za zvuk te kraj njega modul za spajanje kamere. Također, tu je i *Ethernet* ulaz za žično povezivanje s internetom. Izvor napajanja se vrši preko *MicroUSB* ulaza do 5V. Potrebno je istaknuti kako Raspberry Pi nema ugrađen tvrdi disk te je potrebno koristiti memorijsku karticu za instalaciju Raspbian operacijskog sustava[8].



Slika 3.1. *Shema Raspberry Pi-ja s njegovim dijelovima[8]*

3.1.1. Raspored GPIO pinova

Raspberry Pi dolazi s 2 ugrađena reda GPIO (engl. *General-purpose input/output)* pinova na desnom rubu pločice. Sastoji se od 40 pinova koji se koriste kao ulaz ili izlaz te služe za spajanje dodatnih elektroničkih komponenti te upravljanje njima. Njihov raspored dan je na slici 3.2. Poznavanje pinova izuzetno je bitno kako bismo znali pravilno spojiti senzore uz pomoć kojih ćemo mjeriti vremenske prilike[9].



Slika 3.2. *Raspored GPIO pinova[9]*

3.2. Spajanje senzora

Kako bismo uspješno povezali senzore te s njih očitavali podatke, moramo ih ispravno spojiti na GPIO pinove. Pri spajanju moramo paziti da odgovarajuće nožice spojimo na odgovarajuće pinove kako ne bismo oštetili Raspberry Pi.

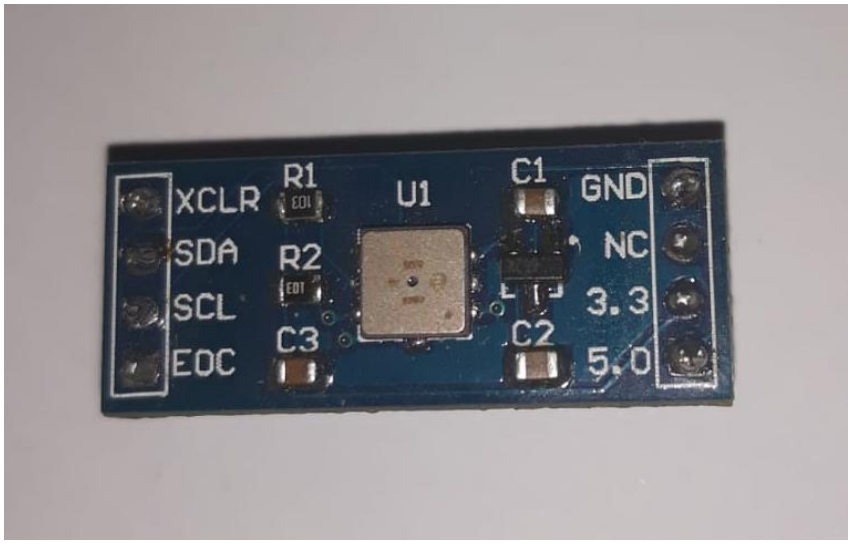
3.2.1. Spajanje DHT11 senzora

DHT11 senzor ima raspored pinova prema slici 3.3. Napajanje senzora spajamo na 3V3 pin koji se nalazi na slici 2.2. pod brojem 1. Izlaz senzora spajamo na GPIO17 pin dok uzemljenje spajamo na pin broj 9.

Slika 3.3. *DHT11 senzor[10]*

3.2.2. Spajanje BMP085 senzora

Kod BMP085 senzora imamo nešto kompliciraniji postupak. Na slici 3.4. vidimo raspored pinova BMP085 senzora. Spajanje radimo na sljedeći način. Pin 3.3 spajamo na pin 17 na Raspberry Pi-u odnosno na 3V3 napon. GND (*engl. Ground*) pin spajamo na pin 20. Idući pinovi koji su nam potrebni su SDA i SCL. SCL pin odnosi se na signal takta dok je SDA za prijenos podataka. SDA pin spajamo na GPIO 2(SDA), a SCL na GPIO 3(SCL). Nakon toga uspješno smo spojili senzore s Raspberry Pi-om.



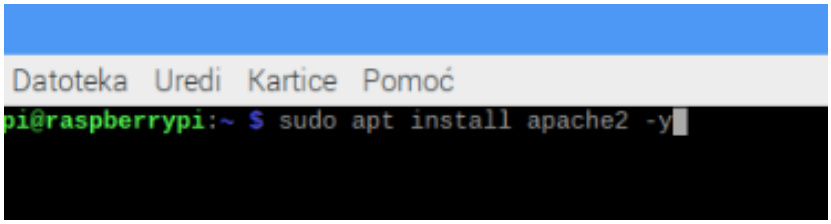
Slika 3.4. *BMP085 senzor*

4. PROGRAMSKO RJEŠENJE METEOROLOŠKE STANICE

Osnova za rad meteorološke stanice je Raspberry Pi. On je odgovoran za čitanje podataka sa senzora te slanje tih podataka aplikaciji. Za postizanje tog cilja koristimo sljedeće tehnologije: Apache web server, phpMyAdmin alat za rukovanje bazom podataka, Python programski jezik te PHP skriptni jezik. Uz to, potrebno je poznavanje Raspbian operacijskog sustava.

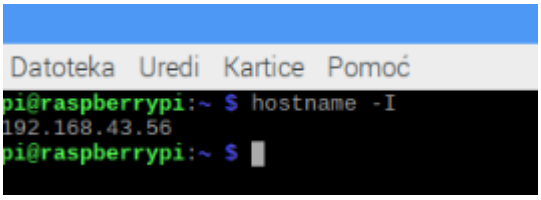
4.1. Apache *web* poslužitelj

Kako bi sve ispravno radilo, potreban nam je poslužitelj (*server*) preko kojega ćemo moći pomoću aplikacije pristupati Raspberry Pi-u. Kao rješenje, potrebno je instalirati Apache *web* poslužitelj. Najprije otvaramo terminal te u njega upisujemo naredbu prikazanu na slici 4.1.



Slika 4.1. *Naredba za instalaciju Apache poslužitelja*[11]

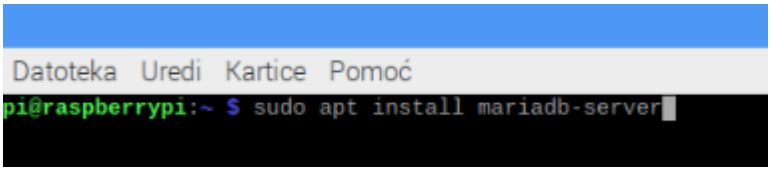
Apache je sada instaliran te je moguće prikazivati jednostavan *web* sadržaj kojemu pristupamo pomoću lokalne IP (*engl. Internet Protocol*) adrese računala koju unosimo u preglednik. Da bismo saznali IP adresu računala, upisujemo u terminal odgovarajuću naredbu, kao na slici 4.2.



Slika 4.2. *Naredba hostname*

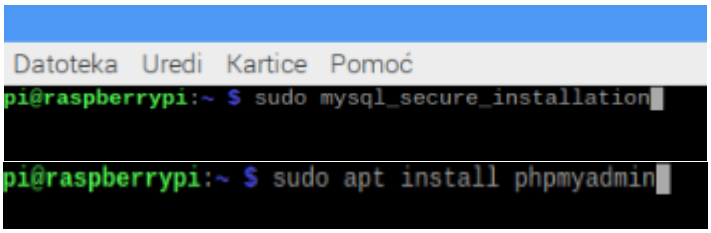
4.2. PhpMyAdmin

PhpMyAdmin alat koristit će nam za rukovanje bazom podataka. Najprije je potrebno instalirati MySQL poslužitelj na lokalnom računalu. Navedeno postižemo naredbom prema slici 4.3.



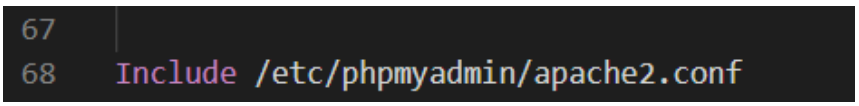
Slika 4.3. *Naredba za instalaciju MySQL poslužitelja*

Potrebno je osigurati pristup poslužitelju kako bi spriječili nedozvoljen pristup[12]. Nakon toga, potrebno je instalirati PhpMyAdmin alat (Slika 4.4.). U procesu odabiremo *apache2 web* poslužitelj, postavljamo lozinku i korisničko ime.



Slika 4.4. *Naredbe za osiguravanje pristupa i instalaciju PhpMyAdmin alata*

Prije nego što pristupimo PhpMyAdmin sučelju, unosimo promjene u Apache2.conf datoteku. Na kraj izlistane datoteke dodajemo liniju koda prikazanu na slici 4.5. Promjene spremamo kombinacijom tipki CTRL+X. Nakon toga, sve je spremno za daljnji rad.



Slika 4.5. *Linija dodana u Apache2.conf*

PhpMyAdmin sučelju pristupamo preko lokalne IP adrese[13]. Sada je potrebno kreirati bazu podataka te tablicu u koju ćemo spremati podatke.

Kreiramo dvije tablice: sensorDataTable i onClickData. Tablica sensorDataTable služi nam za prikupljanje podataka koji se mjere i šalju u bazi svakih sat vremena. Na taj način u tablici će

uvijek biti 24 podatka jer će se svi podatci stariji od jednog dana obrisati. Ova tablica uglavnom će nam služiti za iscrtavanje grafa dnevne temperature.

Tablica onClickData služi nam za dohvaćanje trenutnih vremenskih prilika na zahtjev korisnika. Klikom na gumb unutar aplikacije, mjere se trenutačni uvjeti te se šalju bazi podataka. Nakon toga, iz baze se ti isti podaci dohvaćaju i prikazuju u aplikaciji. Obje tablice imaju iste atribute, a oni su: *id* tipa INTEGER koji je također i primarni ključ, *time* tipa DATETIME, *temperature* tipa VARCHAR(20), *humidity* tipa VARCHAR(20) i *pressure* VARCHAR(20). Sada imamo sve potrebno za pisanje koda za upravljanje senzorima.

Tablica 4.1. *Naziv i tip podatka u tablicama*

Tablice sensorDataTable i onClickData	
Ime podatke	Tip podatka
id	INTEGER (primarni ključ)
time	DATETIME
temperature	VARCHAR(20)
humidity	VARCHAR(20)
pressure	VARCHAR(20)

4.3. Rješenje u pythonu

Pomoću Python programskog jezika obavljat ćemo dohvaćanje podataka sa senzora i njihovo spremanje u bazu. Za rad s prethodno navedenim senzorima, najprije je potrebno dodati potrebne biblioteke za rad u naš projekt. Biblioteke koje su potrebne su: Adafruit_DHT(za rad s DHT11 senzorom) te Adafruit_BMP (za rad s BMP085 senzorom). Također, potrebno je uključiti biblioteku *time* koja nam omogućuje „mirovanje“ programa prije ili poslije izvršavanja ostalog dijela programa. Nakon toga pišemo programski kod na slici 4.6. za dohvaćanje podataka sa senzora. Definiramo varijable *humidity* i *temperature* unutar kojih ćemo spremiti vrijednosti vlage i temperature zraka. Analogno tome, vrijednost tlaka zraka spremamo unutar varijable *pressure*.

```
#DOHVAĆANJE TEMPERATURE I VLAGE SA SENZORA
##########################################
DHT = 17#pin za DHT senzor
global humidity
global temperature
humidity,temperature = dht.read_retry(dht.DHT11, DHT)
#DOHVAĆANJE TLAKA ZRAKA SA SENZORA
#########################################
time.sleep(2)
sensor = BMP085.BMP085()
pressure=sensor.read_pressure()
```

Slika 4.6. *Isječak programskog koda za dohvaćanje podataka sa senzora*

Sada imamo sve potrebne podatke za spremanje podataka u bazu. To činimo unutar istog projekta. Dobivene vrijednost koje su pohranjene u varijable predat ćemo jednostavnoj SQL naredbi za spremanje podataka u bazu(Slika 4.7.).

```
cursor.execute("INSERT INTO sensorDataTable(time, temperature, humidity, pressure) VALUES(CURRENT_TIMESTAMP, %s, %s, %s)"
%(temperature, humidity, pressure))

```

Slika 4.7. *Isječak programskog koda za spremanje podataka u bazu*

Uz podatke sa senzora, također moramo znati kada su oni točno izmjereni. To nam omogućuje već gotova funkcija unutar SQL-a CURRENT_TIMESTAMP, koja vraća trenutno vrijeme i datum. Naši podaci sa senzora sada su uspješno pohranjeni unutar baze podataka u tablicu sensorDataTable. Iste podatke pohranjujemo i u drugu tablicu onClickData koja nam služi za dohvaćanje podataka sa senzora na klik gumba.

Razlog zbog kojega se oni spremaju u drugu tablicu je taj što će se u toj tablici uvijek nalaziti jedan podatak koji je izmjeren u tom trenutku. Svi ostali prijašnji podaci i prijašnja mjerenja se brišu. Budući da nam unutar tablice sensorDataTable trebaju podaci u redovitim vremenskim intervalima radi prikazivanja tih istih podataka na grafu, pokazalo se nezgodno imati podatke van tih intervala unutar iste tablice. Zbog toga nam koristi tablica onClickData, kako bismo u nju pohranili sve podatke dohvaćene kada korisnik klikne na gumb van intervala.

4.4. Rješenje u PHP-u

PHP skriptni jezik omogućit će nam dohvaćanje podataka iz baze te pripremanje tih istih podataka za prikazivanje u aplikaciji. Kako bi se podaci mogli pročitati unutar aplikacije, najjednostavnije je zapisati ih u JSON formatu. To će nam omogućiti PHP. Najprije je potrebno uspostaviti konekciju s bazom podataka te pomoću jednostavnih SQL naredbi izvući podatke iz baze. Podatke tada formatiramo u JSON format. Budući da naša aplikacija prikazuje podatke sa senzora izmjerene samo u jednom danu, potrebno je sve podatke starije od jednog dana obrisati. To isto postižemo SQL naredbom. Dio koda odgovoran za brisanje podataka koji su stariji od jednog dana te dohvaćanje podataka iz baze vidimo na slici 4.8.

```
25
26      $upit_obrisi=$conn->prepare("DELETE FROM sensorDataTable WHERE time < NOW() - INTERVAL 1 DAY;");
27      $upit_dohvati_podatke = $conn->prepare("SELECT time, temperature, humidity, pressure FROM sensorDataTable;");
28      $upit_obrisi->execute();
29      $upit_dohvati_podatke->execute();
30
```

Slika 4.8. *Brisanje i dohvaćanje podataka iz baze*

Svi podaci se sada nalaze unutar varijable *$upit_dohvati_podatke*. Svakoj pohranjenoj vrijednosti unutar upita bazi moramo dodijeliti varijablu u koju će ta vrijednost biti spremljena. To nam omogućuje da dohvaćeni podaci budu grupirani. Npr. u varijablu temperatura bit će spremljene samo vrijednosti za temperaturu, u varijablu tlak samo vrijednosti za tlak itd. Još je potrebno sve podatke formatirati u JSON format. To nam omogućuje naredba *echo json_encode($data). $data* su podaci iz tablice koje predajemo funkciji *json_encode*. Naredba *echo* će ispisati sve podatke na našoj indeks stranici.

Budući da Apache poslužitelj drži našu stranicu na lokalnoj IP adresi računala, pri upisivanju naše lokalne IP adrese u pretraživač vidjet ćemo podatke dobivene iz baze prikazane u JSON formatu. Sada se podaci nalaze na IP adresi te nam je ta ista IP adresa veza pomoću koje ćemo pristupiti podacima s Android aplikacije. Potrebno je napomenuti kako se podaci sa senzora na klik gumba nalaze na stranici s drugom adresom, odnosno na *portu* 5000. Njoj pristupamo preko adrese *localhost:5000*.

4.5. Planirani poslovi

Planirani posao (*engl. Cron job*) se realizira pomoću Cron servisa za vremensko raspoređivanje zadataka. Integriran je unutar sustava koji se temelje na Unix operacijskom sustavu. Svrha mu je izvršavanje zadataka u vrijeme koje prethodno definiramo. Možemo postaviti da se određeni zadatak izvršava svaku minutu, svaki sat ili npr. jednom tjedno. Cron će nam služiti kako bismo mogli izmjeriti vremenske prilike sa senzora točno svakih sat vremena. Cron će svakih sat vremena pokrenuti python skriptu odgovornu za mjerenja.

Kako bismo definirali vrijeme kada će se određeni zadatak izvršiti, moramo postaviti vrijednosti za minute, sate, dan u mjesecu, mjesec te tjedan. Znak * označava da će se zadatak izvršiti svake minute, sata, dana, mjeseca, tjedna. Tako npr. zadatak koji će se izvršavati svake minute zapisujemo sljedećom naredbom: * * * * * <*naredba*>.

Prva zvjezdica označava nam da će se naredba izvršiti svake minute, iduća svaki sat, svaki dan, svaki mjesec i posljednja svaki tjedan. Budući da želimo mjeriti vremenske prilike svakih sat vremena, koristit ćemo naredbu *0 * * * * python /var/www/html/SensorData.py*. Unesena naredba sadrži program kojim ćemo izvršiti zadatak te lokaciju gdje se on nalazi. Najprije otvaramo terminal te unutar njega unosimo naredbu *crontab -e*. Na dnu navedene datoteke unosimo prethodno definiranu naredbu. Datoteku spremamo kombinacijom tipki CTRL + O. Sada će se python skripta izvršiti svakih sat vremena te će se izmjereni podaci spremiti u bazu podataka.

5. PROGRAMSKO RJEŠENJE ANDROID APLIKACIJE

Za kompletno rješenje Android aplikacije koristit ćemo Android Studio. Potrebno je pružiti izbornik pomoću kojega ćemo navigirati po fragmentima. Potrebno je prikazati izmjerene podatke sa senzora te osvježiti podatke na pritisak gumba, odnosno ponovno izmjeriti vremenske prilike. Također, potrebno je izmjerene vrijednosti temperature prikazati grafički te pružiti informacije o vremenu za cijeli tjedan pomoću ostalih izvora na internetu[14].

5.1. Android Studio

Android Studio predstavlja službeno integrirano razvojno okruženje (IDE) za razvoj android aplikacija bazirano na IntelliJ IDEA. Pored razvojnih alata koje pruža IntelliJ, Android Studio dolazi s još više značajki pomoću kojih je značajno unaprijeđen razvoj mobilnih aplikacija. Neke od njih su sljedeće. Podrška za C++ i NDK, široke alate za testiranje, integraciju s GitHubom, već gotove predloške koda, dopušta potvrdu promjena koda za već pokrenutu aplikaciju bez samog *restarta* aplikacije, jedinstveno okruženje za razvoj aplikacija na svim Android uređajima, brz emulator za pokretanje aplikacija, fleksibilan sustav za gradnju temeljen na *Gradleu* i mnoge druge.

5.1.1. Android studio projekt

Prilikom izrade aplikacije, najprije je potrebno kreirati novi projekt. Svaki projekt u Android Studiu sadrži jedan ili više modula s izvornim kodom te izvornim datotekama. Sve naše datoteke u projektu bit će prikazane s naše lijeve strane. Pogled u datoteke organiziran je po modulima da bismo se lakše snalazili. Svaki modul sadržavat će sljedeće mape: manifest, java, res. Manifest sadrži AndroidManifest.xml datoteku koja sadrži nužne informacije o aplikaciji. Sadrži komponente aplikacije, što podrazumijeva sve aktivnosti (*activity)*, servise, sadržajne pakete. Također, mora sadržavati dozvole koje su potrebne aplikaciji za njen rad. Mapa java sadrži potpuni java izvorni kod te JUnit testove. Mapa res sadrži XML datoteke koje nam služe za dizajn aplikacije. Android Studio također pruža pisanje aplikacije u Kotlin programskom jeziku, no za izradu ovog projekta koristit ćemo Javu.

5.2. Java programski jezik

Java programski jezik započeo je s razvojem 1991. godine od tvrtke Sun Microsystems. Ideja je bila stvaranje programskog jezika koji će trošiti male količine memorije te raditi na slabijim varijantama procesora[15]. Osmišljen je tako da bi se generirani kod izvodio na virtualnom stroju koji ne postoji dok bi stvarni stroj imao prevoditelj koji bi prevodio takav kod. Jezik je oblikovan tako da bi sintaksa nalikovala C++ programskom jeziku, budući da su ga mnogi tada poznavali. Najprije se zvao *Oak* te je kasnije preimenovan u Java. Ključan događaj pri razvoju Jave bio je ugradnja *Swing* programskog okvira koji će omogućiti elemente potpuno napisane u Javi i grafičko sučelje. Java omogućuje učitavanje koda s diska ili s mreže te njegovo izvršavanje za vrijeme rada. To u prijevodu znači da je dinamičan. Za izradu android aplikacije nužno je poznavanje programskoj jezika Java premda je sav kod, sve klase, sučelja, funkcije pisane u javi.

5.3. XML

XML proširivi je jezik za označavanje. Dolazi od engleskih riječi „Extensible Markup Language“[16]. Pomoću njega zapisujemo podatke i dokumente u tekstualnom formatu. XML dokument predstavlja tekstualnu datoteku koja se sastoji od oznaka i sadržaja. Osim što je namijenjen za programsku obradu, lako ga mogu čitati i ljudi. Premda sam nije programski jezik, pripada jezicima za označavanje. Jedan od poznatih jezika koji pripada toj skupini je HTML (*engl. Hyper Text Markup Language)*.

Osmišljen je kao jezik za opisivanje podataka i dokumenata. Ako ga promatramo kao tekstualni format, možemo zaključiti da je on potpuno neovisan o platformi na kojoj se nalazi te operacijskom sustavu. Za njegovo korištenje nisu potrebne nikakve licence. XML sastoji se od Oznaka (*eng. Tag)* koje koristimo za opisivanje sadržaja. Oznake nisu unaprijed definirane te ih korisnici moraju sami definirati. U pravilu XML dokument počinje deklaracijom koja se nužno piše na samom početku dokumenta. Deklaracija je oblika:

<?xml version=“1.0“ encoding=“UTF-8“?>

Unutar deklaracije obavezno je navesti verziju XML-a koja se koristi te tip znakova koji se koriste u zapisu dokumenta. Android Studio koristi XML datoteke za opisivanje sadržaja. Unutar XML datoteka definira se izgled za aplikaciju. Pomoću njega uređujemo fontove, boje, raspored (*layout),* ikonice te tekstove.

5.4. Fragmenti

Fragment je svaka klasa koja pruža modularan dizajn aktivnosti. Predstavlja dodatni sloj smješten između korisničkog sučelja i aktivnosti. Imaju vlastiti životni ciklus koji je usko vezan uz aktivnost u kojem se nalaze. Dok imamo otvorenu aktivnost možemo manipulirati pojedinim fragmentima. Za svaki pojedini fragment moramo definirati njegovo sučelje u XML datoteci. Fragment mora biti povezan s XML datotekom koja predstavlja njegov dizajn. Stvaramo objekt vlastite klase fragmenta te koristimo *LayoutInflater* za napuhivanje sučelja iz XML-a.

Za potrebe naše aplikacije koristit ćemo 3 fragmenta. Svaki fragment predstavljat će jedan dio aplikacije. Kao što je već prethodno navedeno, prikazat ćemo podatke direktno sa senzora, podatke s interneta te grafički prikaz tih podataka. Zbog preglednosti i jednostavnosti izgleda aplikacije, pojedine dijelove razdvojili smo u fragmente te unutar svakog fragmenta prikazali pojedini dio.

5.4.1. Prikaz podataka sa senzora

Početni fragment aplikacije prikazivat će podatke izmjerene sa senzora. Potrebno je prikazati podatke o trenutnoj temperaturi, vlazi, tlaku zraka te trenutnom datumu i vremenu. Kreiramo klasu *ForecastFragment* koja proširuje klasu Fragment. Za odgovarajuću klasu potrebna je i XML datoteka koja će predstavljati sučelje navedenog fragmenta. Kreiramo i XML datoteku *fragment_forecast.xml* koju je potrebno napuhati pomoću *LayoutInflatera* kako bi se sučelje prikazalo prilikom otvaranja fragmenta. Navedeno činimo pomoću metode *onCreateView* koja je prikazana na slici 5.1. Navedenu metodu pozivamo upravo za napuhivanje rasporeda odnosno za grafičko prikazivanje sučelja. Metoda vraća pogled (*view*) za sučelje fragmenta. Nakon pozivanja metode *onCreateView,* također je potrebno pozvati metodu *onDestroyView*. Unutar metode *onCreateView,* također je potrebno dodijeliti elementima odgovarajuće elemente unutar *layout* datoteke po njihovim *idovima.* Svaki element koji će se prikazivati korisniku ima svoj *id* po kojemu ga pronalazimo. Fragment mora sadržavati *TextView*-ove za prikaz datuma, temperature, vlage, tlaka, *ImageView* za prikaz ikonice koju preuzimamo s interneta te gumb za osvježavanje podataka, odnosno ponovno mjerenje. Unutar *onCreateView* metode pozivamo i ostale potrebne funkcije koje ćemo naknadno objasniti.

```
public View onCreateView(@NonNull LayoutInflater inflater,
                         @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {
    View view=inflater.inflate(R.layout.fragment_forecast, container,  attachToRoot: false);

    textViewTemp=view.findViewById(R.id.senzor_temperatura);
    textViewHum=view.findViewById(R.id.senzor_vlaga);
    textViewPre=view.findViewById(R.id.senzor_tlak);
    textViewDate= view.findViewById(R.id.forecast_date);
    textView2=view.findViewById(R.id.izmjereno_vrijeme);
    button=view.findViewById(R.id.forecastButton);
    imageView=view.findViewById(R.id.forecast_picture);
    button.setOnClickListener((v) -> { setupAPICall(); });

    setupAPICall();
    setupAPICall2();
    return view;
}
```

Slika 5.1. *Isječak programskog koda metode onCreateView*

5.4.2. Dohvaćanje podataka sa senzora

Za dohvaćanje podataka koristit ćemo *Retrofit 2* kao REST (*engl. Representational state transfer*) poslužitelj. *Retrofit* nam omogućuje jednostavno dohvaćanje JSON podataka s web poslužitelja. Pomoću njega ćemo dohvaćati podatke koji će biti prikazani na lokalnoj IP adresi Raspberry Pi-a. Potrebno je napomenuti da Android uređaj i Raspberry Pi moraju biti na istoj *Wifi* mreži kako bismo mogli pristupiti podacima preko IP adrese. Najprije kreiramo POJO(*eng. Plain Old Java Object*) klasu koja će nam služiti za dohvaćanje podataka sa senzora. Unutar nje imat ćemo atribute te metode za njihovo dohvaćanje. Klasu nazivamo *TemperatureData*. Izgled klase dan je slikom 5.3.

Potrebno nam je još API (*engl. Application programming interface*) sučelje. Unutar njega navodimo krajnje točke HTTP zahtjeva unutar *GET* anotacije koja predstavlja odgovarajuću HTTP metodu čija će se implementacija generirati pri kompilaciji projekta. Metoda vraća *wrapper* objekt *Call* oko tipa podatka koji dohvaćamo. HTTP zahtjev izvršava se *enqueue* metodom koja vraća *Callback* s podacima ili greškom.

```
8   public interface APIInterfaceForecast {
9       @GET(" ")
10      Call<List<TemperatureData>> getData();
11  }
12
```

Slika 5.2. *Sučelje s metodom getData*

TemperatureData.java

```
1   package hr.ferit.sensorapp.Forecast;
2
3   public class TemperatureData {
4       private String time;
5       private Integer temperature;
6       private Float humidity;
7       private Integer pressure;
8
9       public Float getHumidity(){return humidity;}
10      public Integer getPressure(){return pressure;}
11      public String getTime() { return time; }
14      public Integer getTemperature() { return temperature; }
17  }
```

Slika 5.3. *Klasa TemperatureData*

Unutar *setupAPICall* metode instanciramo *Retrofit* servis. Ako instanca ne postoji moramo ju kreirati, a u suprotnom ju samo vraćamo. *Builder patternom* postavljamo bazni URL, koji zapravo označava našu IP adresu preko koje pristupamo podacima, HTTP klijent te JSON *parser*. Predajemo ranije kreirano sučelje s navedenim krajnjim točkama te stvaramo njegovu implementaciju[17]. *Call* metoda šalje zahtjev *web* poslužitelju i vraća odgovor.

```
public void setupAPICall(){
    //getting API call data from database
    Retrofit retrofit=new Retrofit.Builder()
            .baseUrl("http://192.168.43.56:5000/")
            .addConverterFactory(GsonConverterFactory.create())
            .build();
    APIInterfaceForecast apiinterface=retrofit.create(APIInterfaceForecast.class);
    Call<List<TemperatureData>> call=apiinterface.getData();

    call.enqueue(new Callback<List<TemperatureData>>() {
```

Slika 5.4. *Isječak programskog koda unutar funkcije setupAPICall*

Metoda *onResponse* upravlja odzivom servera. Ukoliko dođe do nekakve greške u komunikaciji s poslužiteljem, ova metoda će javiti obavijest o grešci. Nakon što uspješno pristupimo poslužitelju, podaci s poslužitelja biti će spremljeni unutar *response.body()*. Budući da od poslužitelja očekujemo listu objekata *TemperatureData*, jednostavno ćemo odgovor sa poslužitelja spremiti u listu istoimenih objekata. Svaki objekt unutar liste predstavlja jedno mjerenje senzora u određenom vremenu.

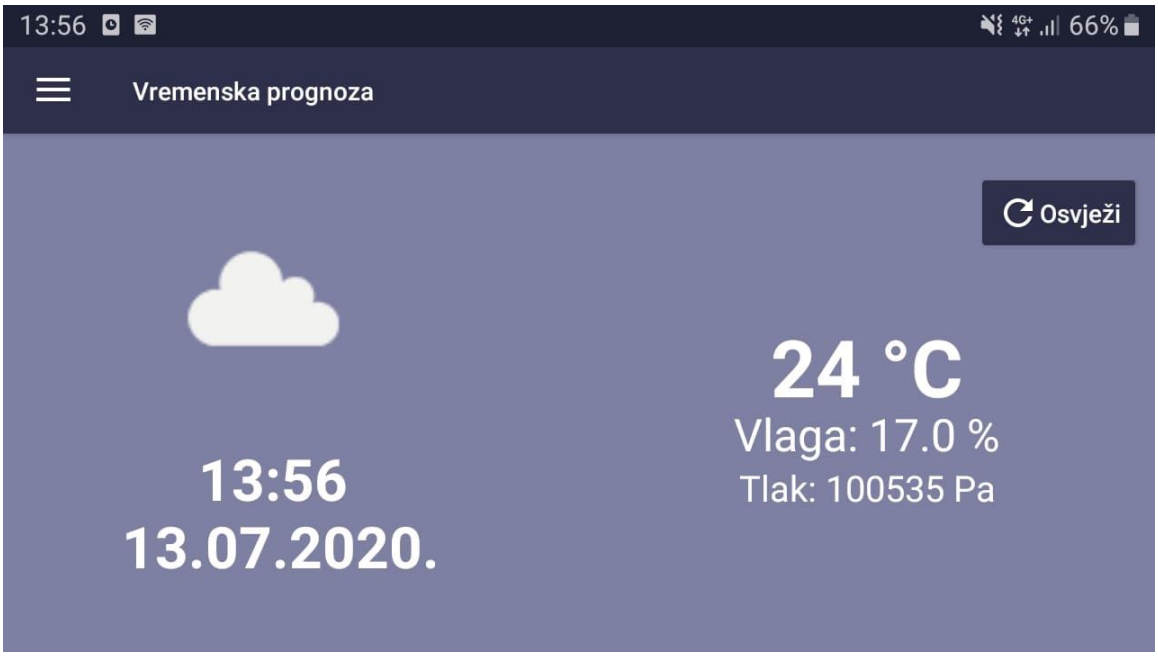
Budući da vremenske prilike mjerimo svakih sat vremena, imat ćemo svaki dan 24 podatka o mjerenju, odnosno 24 objekta unutar liste. Podaci o temperaturi, vlazi, tlaku nalaze se unutar svakog objekta te im pristupamo pomoću javnih *gettera* (metoda *get*). Dohvaćeni podaci o datumu i vremenu su tipa *string*. Kako bismo datum prikazali u odgovarajućem formatu, najprije je potrebno *string* pretvoriti u format datuma. Dobiveni rezultat tada ponovno formatiramo u nama odgovarajući prikaz. Ostale dobivene podatke prikazujemo unutar *textviewa*.

```
List<TemperatureData> mdata = response.body();
for (TemperatureData data : mdata) {
    textViewTemp.setText(""+ data.getTemperature()+ " °C");
    textViewHum.setText("Vlaga: "+ data.getHumidity()+ " %");
    textViewPre.setText("Tlak: "+ data.getPressure()+ " Pa");
```

Slika 5.5. *Isječak programskog koda za podataka u textview-u*

Izgled aktivnosti za fragment obrađen u ovom dijelu možemo vidjeti na slici 5.6. Vidimo kako je korisničko sučelje vrlo jednostavno te intuitivno. Svi dobiveni podaci sa senzora prikazat će se unutar aktivnosti. Pri pritisku na gumb *osvježi* aplikacija ponovno šalje zahtjev *web* poslužitelju. Pri uspješnom slanju zahtjeva, poslužitelj će pokrenuti skriptu u pythonu koja je odgovorna za rad sa senzorima. Nakon mjerenja, podaci se šalju u bazu. Tako pohranjeni podaci konvertiraju se u JSON te prikazuju na stranici s adresom lokalnog računala. Ti podaci tada su proslijeđeni aplikaciji kao odgovor na upit unutar *response.body()*.

U korisničkom sučelju vidljiva je i ikonica koja pobliže označava vremenske prilike. Ikonicu smo dobili s drugih izvora na internetu gdje podaci također odgovaraju trenutnim vremenskim prilikama. Korištenje podataka s drugih izvora objasnit ćemo u idućem dijelu.



Slika 5.6. *Prikaz početnog zaslona*

5.4.3. Tjedni podaci o vremenskim prilikama

Smisao svake meteorološke stanice je dati informacije i o budućim vremenskim prilikama, odnosno tjednu ili mjesečnu prognozu. Budući da pomoću senzora ne možemo predvidjeti ili izmjeriti kakva će biti temperatura narednih dana, koristit ćemo druge izvore s interneta. Sljedeći fragment sadržavat će *recyclerView* za prikaz tjednih podataka o vremenu. Postupak kreiranja fragmenta identičan je kao i u prethodnom primjeru. Jedino što se razlikuje je sučelje za koje je

potrebno napraviti novi *layout* koji će sadržavati *recyclerView*. Unutar svake ćelije *recyclerView-a* prikazat ćemo pojedinosti o vremenu za određeni datum.

5.4.4. RecyclerView

RecyclerView koristi se za prikaz velike količine podataka u obliku liste koju možemo pomicati gore ili dolje[18]. *RecyclerView* reciklira svoje elemente te na taj način povećava performanse aplikacije i brzinu rada. Njemu sličan je *ListView,* ali ga zbog jednostavnosti korištenja uvelike zamjenjuje *RecyclerView*. Unutar *layouta* za prikaz fragmenta potrebno je dodati komponentu *RecyclerView*. Bitno je odrediti *id* same komponente kako bismo ju kasnije koristili u Java kodu.

Da bi se unutar navedene komponente mogli staviti podaci o temperaturi, tlaku, vlazi potrebno ju je dohvatiti preko *id-a* u glavnoj klasi. Nakon toga potrebno je postaviti *LayoutManager*. On služi za upravljanje podacima te ih prezentira unutar liste. Za prikazivanje svakog podatka unutar liste potrebno je napraviti XML datoteku koja će opisivati izgled samo jednog podatka unutar liste. Što je XML datoteka jednostavnija, *RecyclerView* će bolje raditi. XML datoteka mora pružiti mogućnost prikaza 4 tekstualna podatka te ikonicu. Unutar jedne ćelije nalazit će se podaci o temperaturi, vlazi, tlaku, datumu te sličica koja predstavlja trenutnu vremensku priliku.

Da bi se element prezentirao unutar *RecyclerView-a* potreban je *ViewHolder*. On se koristi kod recikliranja elemenata. Da bi se jedan ekran popunio podacima biti će napravljeno onoliko instanci *ViewHoldera* koliko je potrebno za njegovo punjenje. Kreiranje *ViewHoldera* vrši se u adapteru.

Za kreiranje *ViewHoldera* te držanje podataka koji će se prikazivati u *RecyclerView-u* zaslužan je adapter. Da bi adapter radio mora sadržavati sljedeće metode. *GetItemCount* metoda vraća ukupan broj elemenata liste. Metoda *onCreateViewHolder* brine se za kreiranje i recikliranje *ViewHoldera*. *OnBindViewHolder* povezuje podatke u adapteru i *ViewHolder*.

Još je samo potrebno povezati napravljeni adapter s *RecyclerView-om* uz pomoć metode *setAdapter* te ga popuniti s podacima. Kao što smo već ranije spomenuli, koristit ćemo podatke s interneta koje ćemo dohvatiti pomoću API poziva. Meteorološke podatke dohvaćat ćemo sa stranice *OpenWeatherMap*. Potrebno je registrirati se na njihovoj stranici nakon čega dobivamo jedinstveni link preko kojega pristupamo traženim podacima.

```
recycler=view.findViewById(R.id.recycler_view);
recycler.setLayoutManager(new LinearLayoutManager(getContext()));
adapter=new RecyclerAdapter();
recycler.setAdapter(adapter);
```

Slika 5.7. *Inicijalizacija recyclerViewa i povezivanje s adapterom*

Pri posjetu navedenom linku vidimo podatke o vremenu u JSON formatu. Idući korak je kreiranje POJO objekata za svaku pojedinu klasu. Zbog jednostavnosti koristit ćemo online POJO kreator. Potrebno je samo unijeti ispravan JSON kod na osnovu kojega se automatski izrade POJO objekti. Oni sadrže metode za dohvaćanje i postavljanje atributa koji predstavljaju podatke o vremenu. Nadalje koristimo *Retrofit* čija upotreba je opisana u ranijem primjeru te ju stoga nećemo ovdje objašnjavati. Jedina razlika je što podacima više ne pristupamo preko IP adrese lokalnog računala, nego preko linka kojega smo dobili preko *OpenWeaterMap* stranice.

```
public void addData(List<Daily> events) {
    this.mdaily.clear();
    this.mdaily.addAll(events);
    notifyDataSetChanged();
}
```

Slika 5.8. *Metoda addData*

Na slici 5.8. prikazana je metoda *addData* koja se nalazi unutar adaptera. Metoda je odgovorna za popunjavanje adaptera podacima koji su dohvaćeni s interneta. Navedenu metodu pozivamo unutar klase *WeeklyTemperatureFragmen*. Nakon toga *RecyclerView* je potpuno funkcionalan te će prikazivati podatke o vremenu. Izgled *activitya* i *recyclera* prikazan je na slici 5.9.. Pomicanjem ekrana prema dolje odnosno gore pregledavamo sve elemente koji se nalaze unutar liste.

Slika 5.9. *Zaslon s recyclerom za tjednu temperaturu*

5.4.5. Grafički prikaz izmjerenih podataka

Idući korak je grafički prikazati podatke izmjerene sa senzora. Graf će prikazivati podatke o trenutno izmjerenoj temperaturi u ovisnosti o vremenu. Navedeno također činimo u novom *activity-u* te koristimo novi fragment. Podatke koje ćemo prikazivati su podaci unutar *SensorDataTable* tablice u kojoj su zapisani podaci izmjereni svakih sat vremena. Dohvaćanje podataka opisano je u prethodnim primjerima. Za grafički prikaz koristit ćemo biblioteku *MPAndroidChart*[19]. Da bismo koristili navedenu biblioteku, potrebno je biblioteku uključiti u projekt. To činimo unutar *Gradle* skripte.

```
26    dependencies {
27        implementation 'com.github.PhilJay:MPAndroidChart:v3.0.1'
28
```

Slika 5.10. *Dodavanje MPAndroidChart biblioteke u projekt*

Kreiramo odgovarajući *layout* za novi *activity* te u njega dodajemo oznaku za linijski graf. Korištenje biblioteke prilično je jednostavno i intuitivno. U klasi fragmenta inicijaliziramo graf pomoću *id-a* koji smo dodijelili grafu. Pomoću metoda koje su dane unutar biblioteke, uređujemo izgled grafa. Neke od metoda koje koristimo su sljedeće. *SetTouchEnable* koja prima *boolean* vrijednost *true* ili *false*, a omogućuje nam pomicanje grafa na dodir. *SetPinchZoom* koja

pruža uvećavanje grafa ako je potrebno. *SetDrawGridBackground* metoda određuje hoće li se pozadinska rešetka prikazivati. Pomoću metode *getLegend* prikazujemo legendu grafa. Biblioteka pruža dohvaćanje X i Y osi te je na taj način moguće prikazati samo lijevu Y os te donju X os. Također, moguće je formatirati vrijednosti koje će se prikazivati na X ili Y osi. To nam omogućava metoda *YAxisFormatter*. Ako su dobiveni podaci izraženi u celzijusima možemo ih formatirati tako da oni budu prikazani u npr. stupnjevima kelvinima.

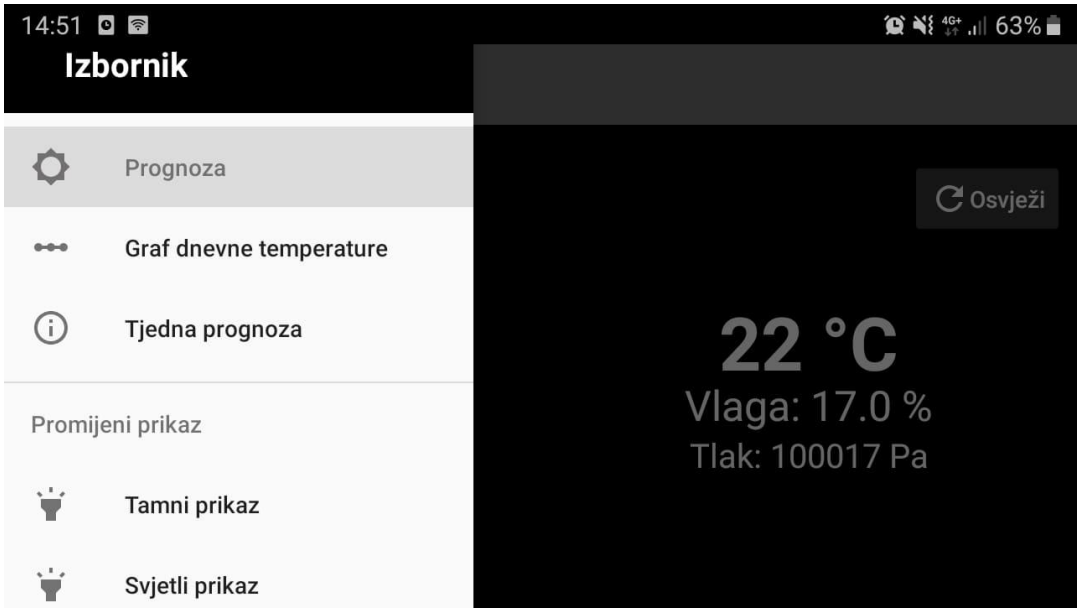
Podaci se spremaju u listu *Entrya*. Jedan *Entry* predstavlja koordinatu za X i Y os. Na taj način svaki podatak o izmjerenoj temperaturi postaje jedan *Entry*. Lista se dalje predaje *LineDataSet* objektu koji će podatke rasporediti tako da oni tvore linijski graf. Nakon toga objekt predajemo inicijaliziranom grafu te je sve spremno za njegovo pravilno prikazivanje.

Slika 5.11. *Zaslon za grafički prikaz podataka*

5.5. Izbornik

Za prebacivanje između *activitya* poslužit će nam izbornik. Izbornik će također imati mogućnost tamnog i svijetlog prikaza aplikacije. Unutar *activity_main.xml* datoteke dodajemo *DrawerLayout*, koji će izbornik prikazivati kao ladicu koja se otvara. Potrebno je kreirati novu XML datoteku koja će predstavljati sami izbornik te stavke unutar izbornika. Stavke su grupirane te svaka stavka predstavlja jedan prethodno definirani *activity*. Potrebne su 3 stavke za preostale *activity-e* te još dodatne 2 stavke koje će imati mogućnost promjene tamnog i svijetlog prikaza. Stavkama pristupamo unutar java koda preko *id-a*.

Unutar alatne trake nalazit će se ikonica s lijeve strane na čiji klik otvaramo izbornik. U glavnoj java klasi inicijaliziramo *DrawerLayout* i alatnu traku preko *id-a*. Za prebacivanje između *activitya* potrebna je metoda *onNavigationItemSelected* koja, ukoliko odabrana stavka izbornika odgovara pojedinom *id-u activitya*, prikazuje odabrani *activity*. Također, ukoliko odaberemo stavku za tamni odnosno svijetli prikaz, izgled *activity-a* će se promijeniti ovisno o tome što smo odabrali. Da bismo to postigli bilo je potrebno samo unutar XML datoteke *colors.xml* dodati novi noćni prikaz te unutar njega definirati boje koje će se prikazivati kada je on uključen. Na slici 5.12. vidimo tamni prikaz te izbornik.



Slika 5.12. *Tamni prikaz s izbornikom*

6. ZAKLJUČAK

Cilj završnog rada bio je napraviti mali samostalni sustav za praćenje vremenskih prilika s mobilnom aplikacijom. Za razliku od već postojećih kućnih meteoroloških stanica, prednost rješenja koje nudi ovaj završni rad je u tome što u svakom trenutku možemo znati podatke s kućne meteorološke stanice pogledom na Android uređaj. Jednim klikom možemo izmjeriti vremenske prilike na kućnoj meteorološkoj stanici kao da ju uvijek imamo sa sobom.

Za postizanje rješenja uvelike nam je poslužio Raspberry Pi. Pomoću njega upravljamo periferijom, odnosno sa senzorima koji su odgovorni za mjerenja. Također, na njemu pokrećemo server na kojemu se nalazi baza podataka te pomoću kojega je moguće pristupiti Raspberry Pi-u preko android aplikacije. Za rukovanje senzorima koristili smo programski jezik Python. Kao alat za rukovanje bazom podataka korišten je PHPMyAdmin. Za pripremu podataka koje možemo koristiti unutar aplikacije korišten je PHP.

Android aplikacija u potpunosti je izrađena u Android Studio razvojnom okruženju. Uz korištenje brojnih tehnologija koje pruža Android Studio uspješno smo napravili android aplikaciju. Korišteni su fragmenti, API pozivi, *RecyclerView* itd.. Uz prikaz podataka izmjerenih sa senzora, prikazani su i podaci s drugih izvora na internetu omogućavajući na taj način osim dnevne, tjednu prognozu vremena. Također, izmjereni podaci prikazani su grafički.

Prednost ovog sustava za praćenje vremenskih prilika nad ostalim kućnim meteorološkim sustavima je u tome što pruža Android aplikaciju za prikaz podataka sa senzora. Činjenica je da većina današnjeg stanovništva posjeduje Android uređaj. Nedostatak ovog sustava je u tome što za pristup podacima sa senzora moramo biti spojeni na lokalnu Internet mrežu. Sustav se može unaprijediti izradom online baze kojoj će se moći pristupiti preko interneta. Također, moguće je nadograditi sustav dodavanjem više senzora.

LITERATURA

[1] Izradi svoju meteorološku stanicu, https://projects.raspberrypi.org/en/projects/build-your-own-weather-station/4, pristupljeno 16.7.2020.

[2] Arduino projekt meteorološke stanice, https://howtomechatronics.com/tutorials/arduino/arduino-wireless-weather-station-project/, pristupljeno 16.7.2020.

[3] *Oregon* meteorološka stanica uz Raspberry Pi, https://www.instructables.com/id/Connect-Raspberry-Pi-to-Oregon-Scientific-BLE-Weat/, pristupljeno 16.7.2020.

[4] Trgovina *Play*, https://play.google.com/store/apps/details?id=com.grailr.carrotweather&hl=en_US, pristupljeno 16.7.2020.

[5] Samsung *widget*, https://www.youtube.com/watch?v=JKznFKuv-hQ, pristupljeno 16.7.2020.

[6] *Davis Instruments, Vantage Vue* meteorološka stanica, https://www.davisinstruments.com/vantage-vue/, pristupljeno 27.8.2020.

[7] Raspberry Pi, https://www.raspberrypi.org/about/, pristupljeno 3.7.2020.

[8] Raspberry Pi upute, https://thepihut.com/blogs/raspberry-pi-tutorials/the-raspberry-pi-tutorial-beginners-guide, pristupljeno 3.7.2020.

[9] GPIO, https://www.raspberrypi.org/documentation/usage/gpio/README.md, pristupljeno 3.7.2020.

[10] DHT11 senzor, https://www.thegeekpub.com/wiki/sensor-wiki-ky-015-dht11-combination-temperature-and-humidity-sensor/, pristupljeno 4.7.2020.

[11] Pokretanje Apache *web* poslužitelja, https://pimylifeup.com/raspberry-pi-apache/, pristupljeno 4.7.2020.

[12] Raspberry Pi baza podataka, https://pimylifeup.com/raspberry-pi-mysql/, pristupljeno 4.7.2020.

[13] PhpMyAdmin, https://pimylifeup.com/raspberry-pi-phpmyadmin/, pristupljeno 5.7.2020.

[14] Android Studio, https://developer.android.com/studio/intro#code_completion, pristupljeno 10.7.2020.

[15] M. Topolnik; M. Kušek, Uvod u programski jezik Java, Skripta uz kolegij Informacija, logika i jezici, Zagreb, 2008

[16] D. Kirasić, XML tehnologija i primjena u sustavima procesne informatike, Sveučilište u Zagrebu

[17] J. Balen, Osnove razvoja web i mobilnih aplikacija, LV4

[18] J. Balen, Osnove razvoja web i mobilnih aplikacija, LV2

[19] MPAndroidChart, https://github.com/PhilJay/MPAndroidChart, pristupljeno 13.7.2020.

SAŽETAK

Pomoću Raspberry Pi računala ostvaren je cilj izrade kućne meteorološke stanice. Razvojno okruženje Android Studio služio je za izradu android aplikacije. Kućna meteorološka stanica koja prati vremenske prilike te android aplikacija koja prikazuje te podatke na android uređaju, konačno je rješenje ovog završnog rada. Senzori korištenu za mjerenja vremenskih prilika su DHT11 i BMP085. Za upravljanje njima korišten je programski jezik Python.

Za razvoj android aplikacije u Android Studio okruženju korišteni su programski jezik Java i XML opisni jezik. Na pritisak gumba u android aplikaciji dohvaćaju se podaci sa senzora te se šalju aplikaciji i prikazuju na ekranu. Omogućeno je i prikazivanje tjednih podataka o vremenu. Također, podaci izmjerene temperature prikazani su grafički.

Ključne riječi: Android aplikacija, Android Studio, BMP085, DHT11, Raspberry Pi, Vremenska stanica

ABSTRACT

Home weather station with mobile application

Using Raspberry Pi computer, home weather station has been made. Android Studio IDE has been used for making Android application. Home weather station that tracks weather conditions and an Android application that display these data on the Android computer, are the final solution of this thesis. Sensors that have been used for measurement of weather conditions are DHT11 and BMP085. Programming language Python has been used for controlling them.

For development of android application in Android Studio environment we used Java programming language and XML markup language. On button press in Android application, data from sensor is sent to application and displayed on the screen. It also shows weekly temperature information. Measured temperature is shown on chart.

Key words: Android application, Android Studio, BMP085, DHT11, Raspberry Pi, Weather station

ŽIVOTOPIS

Mario Petričević rođen je 28. studenog 1998. godine u Osijeku. Nakon završene Osnovne škole Josipa Antuna Ćolnića u Đakovu, upisuje matematičku gimnaziju Antuna Gustava Matoša u istoimenom gradu. Svoj interes za programiranjem pokazuje tijekom srednjoškolskih dana gdje sudjeluje na brojnim natjecanjima. Srednju školu završava s vrlo dobrim uspjehom te 2017. godine upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, smjer računarstvo. Trenutno je student 3. godine računarstva.

Potpis: _____

PRILOZI

Prilog 1. Na optičkom disku se nalazi pismeni dio završnog rada u .pdf i .docx formatu

Prilog 2. Android aplikacija u direktoriju WeatherApp te programski kod ukupnog projekta