SUSTAV ZA NADZOR VLAŽNOSTI I NAVODNJAVANJE TLA

Mudrovčić, Dino

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

SUSTAV ZA NADZOR VLAŽNOSTI I NAVODNJAVANJE TLA

Dino Mudrovčić

Zagreb, studeni 2019.

„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, 07.11.2019.

Predgovor

Zahvaljujem mentoru mag. Aleksanderu Radovanu, na pomoći tijekom izrade završnog rada.

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

Sažetak

Ovaj rad prikazuje cjelovito rješenje sustava za nadzor vlažnosti i navodnjavanje tla. Korisnik se prijavljuje u Android aplikaciju i odabirom lokacije pregledava stanje dobivenih vrijednosti sa senzora, vremensku prognozu te kontrolira stanje ventila za vodu. Podatci sa senzora prikupljaju se pomoću Arduino kompatibilnih pločica, kao i slanje tih podataka te kontrola ventila za vodu. Za pohranu podataka sa senzora, statusa ventila za vodu i lokacije senzora, koristi se Firebase baza podataka u realnom vremenu.

Ključne riječi: Android, Arduino, Firebase, senzor, ventil.

Summary

This project shows complete solution of system for moisture monitoring and soir irrigation. The user logs into the Android application and by selecting the location, he inspects the status of the sensors, the weather forecast and controls the status of water valves. Sensor dana is collected using Arduino-compliant tiles, as well as sending that data and water valve control. Real-time Firebase database is used to store sensor data, water valve status and sensor locations.

Keywords: Android, Arduino, Firebase, sensor, valve.

Sadržaj

1. Uvod

Završni rad obrađuje temu sustava za nadzor vlažnosti i navodnjavanja tla. Sustav je osmišljen kako bi olakšao poljoprivrednicima, ali i običnim građanima, kontrolu navodnjavanja željenih biljaka ili polja. Trenutno se to, većinom, obavlja ručno uz poneki automatizirani proizvod, dok ovaj sustav nudi potpunu automatizaciju procesa te kontrolu na daljinu. Ovaj sustav sastoji se od tri dijela. Prvi dio predstavlja Firebase baza podataka, drugi dio predstavlja rješenje sustava senzora i ventila za vodu, pomoću Arduino platforme i treći dio koji predstavlja Android aplikacija.

Firebase baza podataka služi za pohranu podataka sa senzora i pohranu statusa ventila, ali i ostale korisne podatke, poput informacija o korisniku i informacija o lokaciji. Firebase baza podataka također sadrži i slike, koje se koriste u Android aplikaciji.

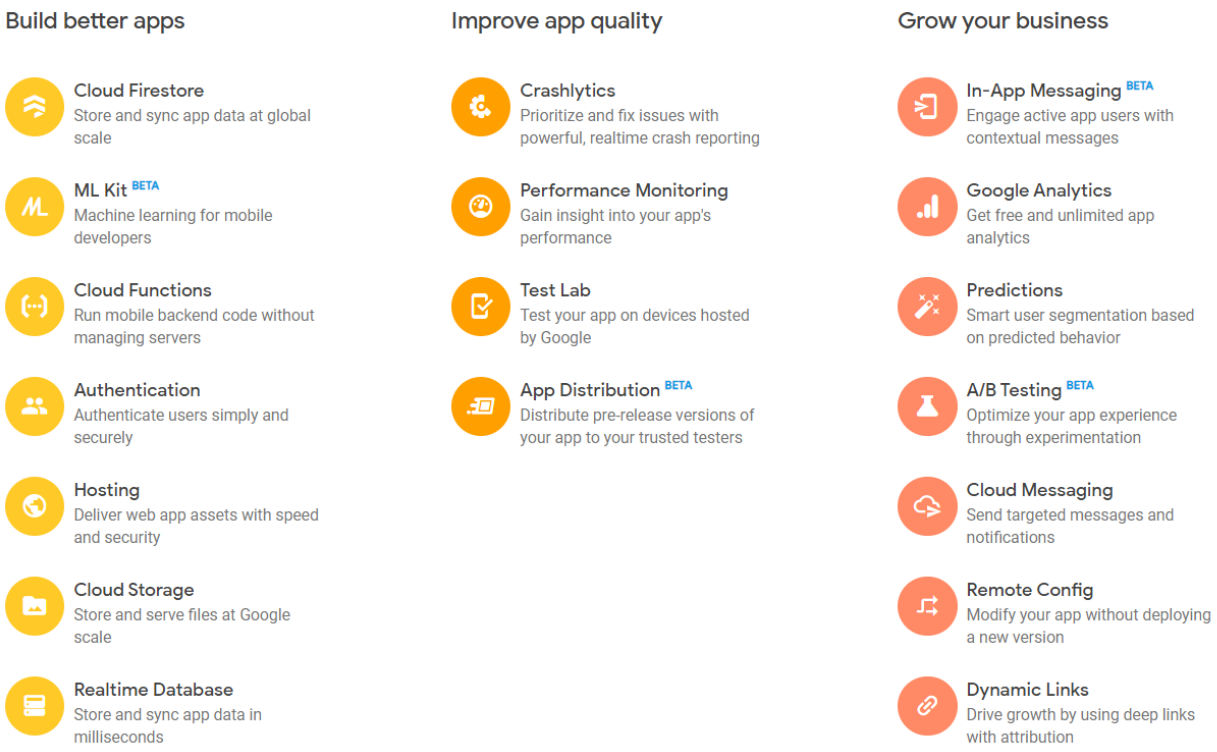
Rješenje sustava senzora i ventila pomoću Arduino platforme, koristi kompatibilne pločice koje se proizvode u Republici Hrvatskoj. Pomoću tih pločica prikupljaju se podatci sa senzora te se upravlja ventilima za vodu. Koristi se Wi-Fi tehnologija za komunikaciju s internetom.

Android aplikacija razvijena je kako bi prikazala podatke o lokaciji senzora prijavljenog korisnika, podatke sa senzora koji su na određenoj lokaciji, podatke o vremenskoj prognozi za određenu lokaciju i kontrolu stanja ventila za vodu.

Cilj i svrha ovog sustava je automatizacija postojećih procesa u poljoprivredi, te mogućnost da se to sve, pomoću aplikacije, prati i kontrolira na daljinu.

2. Platforma Firebase

Firebase [2] je web platforma koja služi za razvoj mobilnih i web aplikacija. Pokrenuta je 2011. godine u tvrtki Firebase Inc., a od 2014. godine tvrtku preuzima Google i počinje s dodatnim proširenjima. Osim što služi za razvoj tih aplikacija, Firebase sadrži jako puno korisnih servisa za korisnike, koji u svom razvoju odluče koristiti Firebase platformu. Trenutno je dostupno 18 servisa na platformi (Slika 1.)



Slika 1. Popis servisa Firebase platforme[2]

Bitni servisi [2] za ovaj rad su:

- Autentifikacija (engl. *Authentication*) – služi za olakšavanje prijave korisnika u aplikaciju. Servis podržava autentifikaciju putem lozinke (email adresa), mobilnih telefona, Google računa, Facebook računa, Twitter računa i sl.
- Prostor u oblaku (engl. *Cloud Storage*) – služi programerima za aplikacije koje trebaju pohranu i posluživanje sadržajem korisnika, kao što su slike ili videozapisi.
- Baza u realnom vremenu (engl. *Realtime Database*) – služi za pohranu i sinkronizaciju podataka između više različitih platformi, kao što su mobilna ili web aplikacija i Firebase NoSQL baze podataka u oblaku.

3. Uvod u Arduino

3.1. Opće informacije

Arduino [19] je elektronička platforma otvorenog kôda (engl. *open source*), koja se temelji na jednostavnom korištenju *hardware*-a i *software*-a. Arduino pločice sposobne su čitati dolazeće podatke, s bilo kojeg senzora, s kojim su kompatibilne. Osim čitanja, pločice su sposobne i slati izlazne podatke, odnosno impulse, kako bi se ispisali podatci na ekran ili recimo, kako bi se upalila LED žarulja.

Upravljanje pločicom je moguće putem grupe instrukcija, koje se šalju na mikrokontroler, koji se nalazi na pločici. Pisanje instrukcija se obavlja pomoću Arduino IDE-a i Arduino programskog jezika, koji je ustvari C++.

Arduino je stvoren u Ivrea Interaction Design Institute-u, u Italiji. Stvoren je kao jednostavan alat za kreiranje prototipa, prvenstveno namijenjen studentima bez podloge o elektronici i programiranju. Međutim, Arduino je postao vrlo popularan u svijetu sa svojim 8-bitnim pločicama, koje su omogućile razvoj IoT proizvoda, 3D print proizvoda te *embedded* okoline. S obzirom na otvorenost kôda, Arduino je ohrabrio korisnike da kreiraju različite proizvode i da ih prilagode svojim potrebama [19].

3.2. Prednosti Arduina

Neke od konkurentnih platformi Arduinu su: Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard i sl. Međutim, ono što mu daje prednost nad tim platformama je, sigurno, jednostavnost korištenja te iskustvo korisnika. Ovu pločicu mogu koristiti svi, koji žele nešto naučiti o elektronici te razvoju *hardware*-a i *software*-a, ali i oni koji su iskusniji korisnici i žele napredovati u svom polju i razviti jeftiniji, ali i kvalitetan prototip svojeg proizvoda. Također, Arduino se može pokrenuti na Windows-u, Macintosh OS X-u te Linux operacijskom sustavu [19].

Kada se sve zbroji, prednosti se mogu prikazati kroz sljedećih pet točaka:

- Niska cijena
- Radi na više platformi
- Jednostavno programsko okruženje
- Otvorenost kôda i proširivi *software*
- Otvorenost kôda i proširivi *hardware*

3.3. Arduino *hardware*

Najpoznatija pločica [21] je Arduino Uno (Slika 2.), koju se smatra najboljom za početak bavljenja elektronikom i pisanjem kôda. Dokumentacija za ovu pločicu je najbolja i dostupna je na internetu.

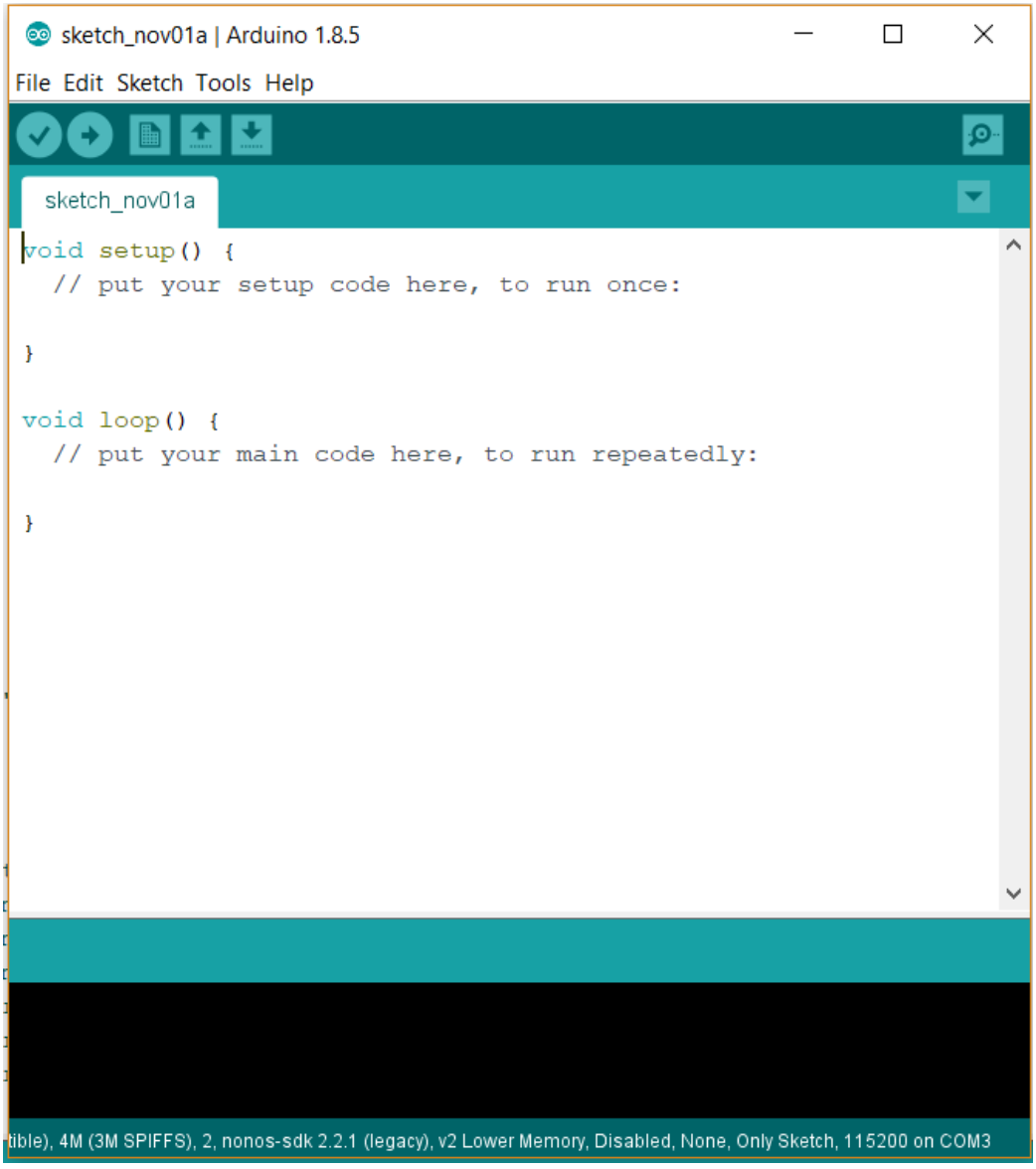
Pločica je mikrokontroler baziran na ATmega328P mikrokontroleru. Pločica ima 14 digitalnih ulazno-izlaznih pinova, 6 analognih ulaznih pinova, USB, napajanje i gumb za *reset*. Najjednostavnije ga je spojiti USB kabelom na računalo.

Kako bi ga računalo prepoznalo, potrebno je instalirati drivere za taj konkretan Arduino. Svaka pločica ima posebne drivere pa je potrebno pripaziti na to koji driveri su instalirani na računalo.

Slika 2. Arduino Uno pločica[11]

3.4. Arduino *software*

Za pisanje naredbi za mikrokontroler, koristi se Arduino IDE[22]. Najnovija verzija je Arduino 1.8.10. Arduino IDE je otvorenog kôda, jednostavan za pisanje kôda i *upload* kôda na pločicu. Moguće ga je instalirati na Windows, Mac OS X te Linux. U novije vrijeme, razvijen je i Arduino Web Editor, koji dodatno olakšava početak programiranja pločice. Bez potrebe za instalacijom, pristupa se pisanju kôda putem internet preglednika. Jedini korak koji treba napraviti je registracija email adresom i lozinkom, na Arduino internet lokaciju.



Slika 3. Arduino IDE

Korisničko sučelje Arduino IDE (Slika 3.) sadrži gumbe za *compile*, *upload*, *new*, *save* i *load*. Vidljive su dvije metode, `setup` i `loop`. Metoda `setup` je metoda koja se poziva samo jedan put i uglavnom se koristi za inicijalizaciju varijabli i pinova. Metoda `loop` je metoda koja se izvršava periodično i u nju se upisuje kôd koji je potrebno stalno izvršavati. Moguće je dodati više dodatnih metoda koje se smatraju korisnim i potrebnim.

```
// the setup function runs once when you press reset or power the board
void setup() {
      // initialize digital pin LED_BUILTIN as an output.
      pinMode(LED_BUILTIN, OUTPUT);
}
// the loop function runs over and over again forever
void loop() {
      digitalWrite(LED_BUILTIN, HIGH);
      delay(1000);
      digitalWrite(LED_BUILTIN, LOW);
      delay(1000);
}
```

Kôd 1. Jednostavan Arduino program

U kôdu (Kôd 1.) je prikazana jednostavna Arduino aplikacija. U `setup` metodi inicijaliziran je pin i definiran je kao izlazni pin, parametrom 'OUTPUT'. U `loop` metodi se nalaze četiri linije kôda. Prvo se pali LED lampica, nakon toga se čeka jedna sekunda, pa se LED lampica gasi i zadnji korak je čekanje u trajanju od jedne sekunde. Nakon što se izvrši taj krug naredbi, Arduino kreće u `loop` metodu ispočetka.

4. Uvod u Android

4.1. Opće informacije

Android je operacijski sustav otvorenog kôda (engl. *open source*), koji se koristi za razvoj mobilnih aplikacija. Android operacijski sustav temeljen je na Linux 2.6 jezgri[1]. Razlog korištenja Linux-a je otvorenost kôda, njegova portabilnost te sigurnost. U ovoj komponenti sustava nalaze se još i:

- Upravljanje procesima
- Upravljanje memorijom
- Upravljanje energijom
- Upravljanje mrežom

Sljedeća važna komponenta Androida je skup nativnih biblioteka (engl. *native libraries*). Nativne biblioteke pisane su u C / C++ kôdu, a odlikuje ih otvorenost kôda. Primjeri nativnih biblioteka su:

- SQLite
- OpenGL

SQLite[1] je jednostavna ugrađena baza podataka, dok je OpenGL biblioteka za korištenje 3D grafike na mobilnom uređaju.

Osim nativnih, u Androidu postoje i Android biblioteke koje predstavljaju rad sa `string`-ovima, brinu o datotečnom sustavu i o pristupu mreži.

Nakon nativnih i Android biblioteka dolazi Dalvik, koji je vlastita virtualna mašina kojom se rješavaju problemi poput:

- Trajanje baterije
- Procesorska moć
- Licenciranje

Dalvik virtualna mašina optimizirana je za simultano izvršavanje više instanci. Dalvik i Android biblioteke predstavljaju cjelinu koja se naziva Android Runtime.

Zadnja komponenta Android operacijskog sustava je aplikacijski okvir. Unutar aplikacijskog okvira nalaze se komponente, koje se koriste za razvoj aplikacija. Koncepti tih komponenti su:

- Ponovna iskoristivost kôda (engl. *reusability*)
- Zamjenjivost komponenti (engl. *interchangeable*)
- Publiciranje mogućnosti i podataka

Tablica 1. Prikaz prednosti i mana razvoja aplikacija u okviru Android operacijskog sustava

Prednosti	Mane
• Raširenost • Jednostavna distribucija • Nema certificiranja • Nema postupka odobravanja • Komunikacija s ugrađenim aplikacijama • Izjednačavanje s ugrađenim aplikacijama	• Procesorska snaga • Radna memorija • Trajna memorija • Trajanje baterije • Troškovi prijenosa podataka • Latencija • Brzina prijenosa • Raznolikost uređaja

4.2. Razvojno okruženje

Programski jezici koji se koriste za pisanje Android aplikacija su Java i Kotlin. Oba izvorna kôda se prevode pomoću prevoditelja (engl. *compiler*) u bajt kôd (engl. *bytecode*). Dalje se putem Dex prevoditelja prevodi u Dalvik bajt kôd, koji se nakon toga izvršava na virtualnom stroju Dalvik[1].

Ostali softverski zahtjevi su (minimalni):

- Operacijski sustav
    - Microsoft Windows XP
    - Mac OS X 10.5.8
    - Linux
- Razvojni alati
    - Java Development Kit (JDK) 6
    - Android Software Development Kit (SDK)

Android SDK je skup alata, komponenti i dokumentacije potrebnih za razvoj aplikacija. Sadrži sljedeće komponente:

- Android API
- Razvojni alat
- AVD manager
- Emulator
- Dokumentacija

4.3. Verzije Androida

Verzije Androida su važna stavka u procesu razvoja mobilnih aplikacija. Prva verzija Androida je izašla rujnu, 2008. godine[1]. Nije imala kodno ime, broj verzije je bio 1.0 i API *level* 1.

Trenutna verzija je 10.0, ima kodno ime 'Android 10' i API *level* 29 te je dostupna od rujna, 2019. godine. Zadnja, starija, verzija koja ima Google podršku je verzija 7.0, kodnog imena 'Nougat' i API *level*-a 24.

4.4. Vrste aplikacija

U Androidu postoje četiri grupe aplikacija koje se mogu razvijati[1]:

- *Foreground*
- *Background*
- *Intermittent*
- *Widgets* i *Live Wallpapers*

Foreground aplikacije su vidljive aplikacije. To su najčešće igre, galerija, internet preglednik i sl. *Background* aplikacije su aplikacije koje se „vrte“ u pozadini, odnosno to su skrivene aplikacije. Primjeri takvih aplikacija su alarm, telefon i SMS. *Intermittent* aplikacije su većina aplikacija kao što su *media player*, e-mail, vijesti i sl. *Widgets* i *Live Wallpapers* su aplikacije koje se prikazuju na glavnom ekranu poput sata, prognoze, podešavanja zvuka i kalendara.

4.5. Životni ciklus aplikacije

Svaki životni ciklus aplikacije počinje s pokretanjem same aplikacije. Aplikacija se pokreće kao zaseban Linux proces. Za upravljanje životnim ciklusom aplikacije, zadužen je upravitelj aktivnosti (engl. *activity manager*). Upravitelj aktivnosti je ugrađeni mehanizam unutar same Android aplikacije[1].

Načini završavanja životnog ciklusa aplikacije su tradicionalni i Android-ov način. Tradicionalni načini su:

- Korisnik eksplicitno zatvara aplikaciju
- Neki zadatak se ne odradi
- Dogodi se nepredviđena iznimka

Android-ov način je ili tradicionalni način ili nasilno prekidanje aplikacije.

Android poprilično agresivno upravlja resursima i radi sve potrebno kako bi pružio potrebne resurse procesu višeg prioriteta[1]. U tom slučaju, oslobađa resurse manje prioritetnih procesa. U slučaju da dvije aplikacije imaju isti prioritet, Android će „ubiti“ proces koji je duže u tom prioritetu. Poredak procesa prema prioritetima:

1. Aktivni procesi
2. Vidljivi procesi
3. Procesi s pokrenutim servisom
4. Pozadinski procesi
5. Prazni procesi

Glavna komponenta svake Android aplikacije je aktivnost[1]. Aktivnost predstavlja prezentacijski sloj aplikacije, odnosno prozor koji je prikazan korisnicima. Glavne stavke aktivnosti su prezentacija i interakcija. Svaka aktivnost mora biti definirana u manifestu Android aplikacije (Kôd 2.).

```
<activity android:name=".activities.MainActivity">
      <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
</activity>
```

Kôd 2. Prikaz definirane aktivnosti u Android manifestu

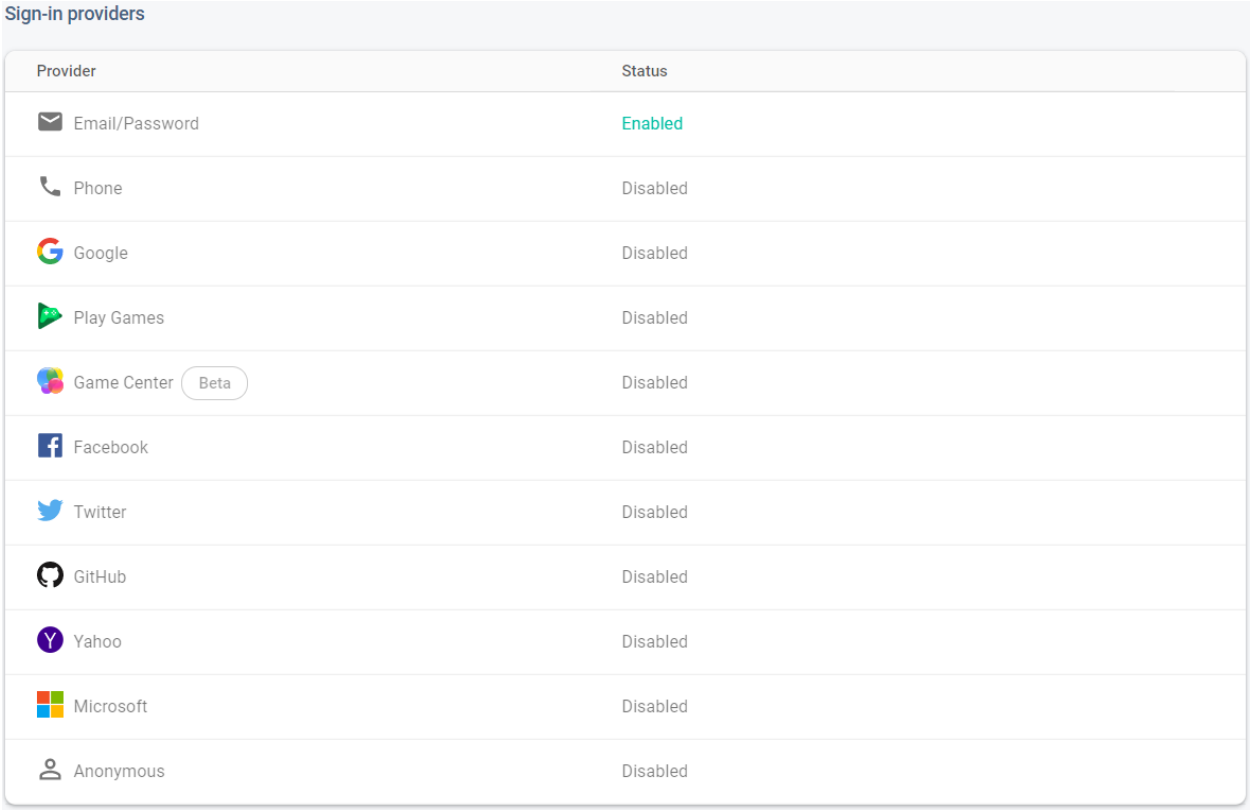
Svaka aktivnost ima četiri stanja aktivnosti:

- Aktivna
- Pauzirana
- Zaustavljena
- Neaktivna

5. Funkcionalnost Firebase platforme

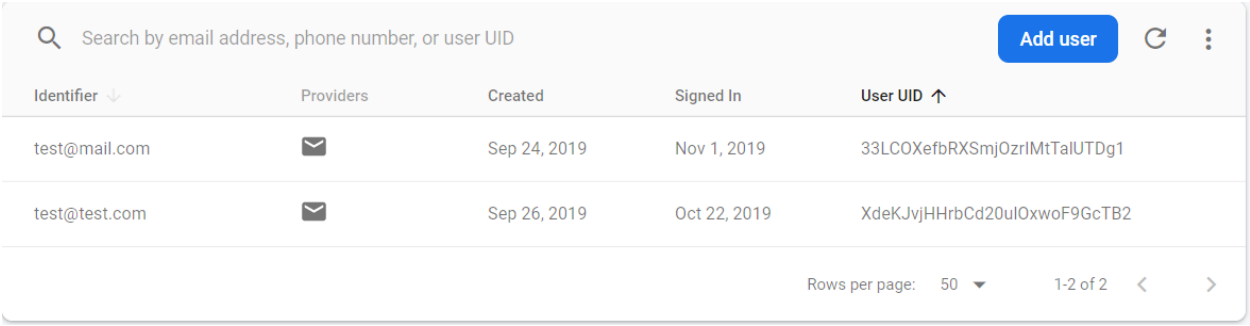
5.1. Autentifikacija

Uz pomoć Firebase autentifikacije[5], prijava u aplikaciju je vrlo jednostavna. U Firebase konzoli je, pod karticom Authentication, dostupno nekoliko mogućnosti za prijavu (Slika 4.).



Slika 4. Prikaz mogućnosti prijave korisnika u aplikaciju

Za osnovnu prijavu dovoljno je uključiti opciju Email/Password. Nakon što je opcija uključena, potrebno je u kartici *Users*, dodati korisnike aplikacija za koje će biti omogućena prijava. U tablici dodanih korisnika (Slika 5.), prikazani su stupci za identifikator korisnika, način na koji se prijavljuje, datum kada je korisnik kreiran, datum kada je zadnji puta ulogiran te UID koji predstavlja ID korisnika.



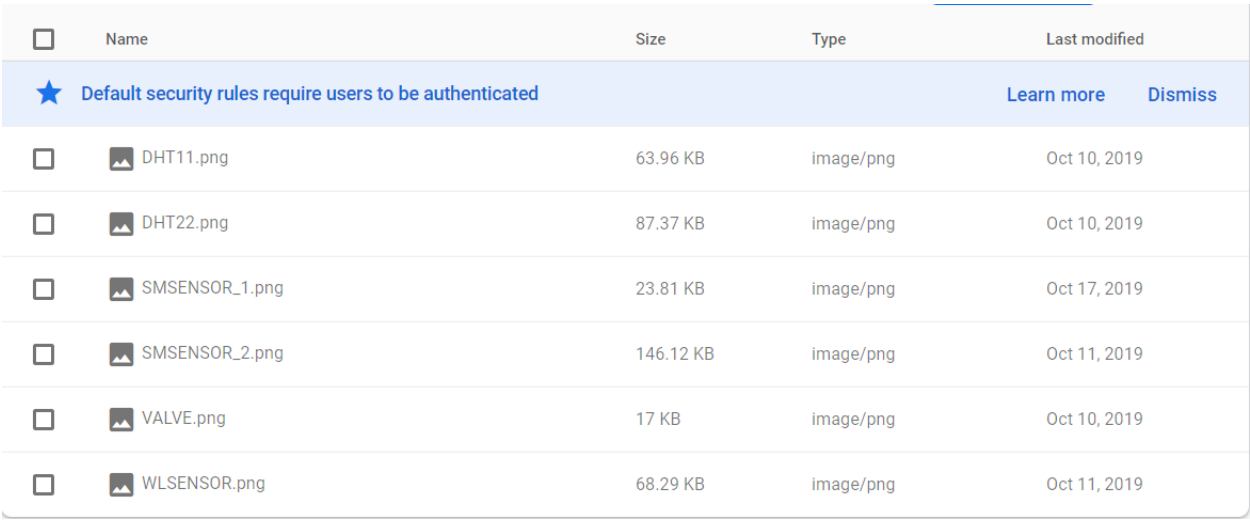
Slika 5. Korisnici za koje je omogućena prijava u aplikaciju

5.2. Prostor u oblaku

Prostor u oblaku[3] služi za pohranu slika, videozapisa ili drugih tipova dokumenata. To je servis koji olakšava dohvat tih dokumenata, ali i administraciju istih. Tri su bitne kartice u ovom servisu:

- Dokumenti
- Pravila
- Korištenje

Kartica 'Dokumenti' prikazuje koji su to dokumenti trenutno pohranjeni na prostor u oblaku. To je tablica (Slika 6.) koja prikazuje podatke kao što su ime dokumenta, veličina dokumenta, tip dokumenta i zadnja promjena dokumenta.



Slika 6. Popis pohranjenih dokumenata na prostor u oblaku

Kartica 'Pravila' sadrži popis pravila o korištenju resursa iz prostora u oblaku. Definira tko smije čitati i/ili mijenjati dokumente te na koji način će ti dokumenti biti prikazani. Definiranje tih pravila se izvršava kroz nekoliko linija kôda (Kôd 3.). Ovo je vrlo važna stavka sigurnosti, jer ukoliko se dozvoli pristup prostoru u oblaku bez autentifikacije, može doći do zlouporabe sustava.

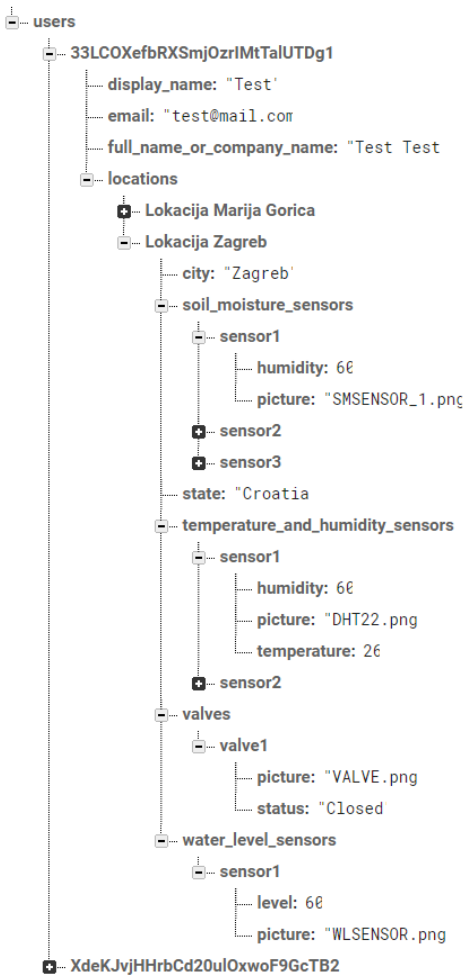
```
rules_version = '2';
service firebase.storage {
  match /b/{bucket}/o {
    match /{allPaths=**} {
      allow read, write: if request.auth != null;
    }
  }
}
```

Kôd 3. Definiranje pravila pristupa prostoru u oblaku

Kartica 'Korištenje' predstavlja prikaz podataka o internet prometu i o pozivima prema resursima, koji su pohranjeni na prostor u oblaku. To je važan podatak za praćenje prometa, ali i zahtjeva za resurse iz oblaka.

5.3. Baza u realnom vremenu

Firebase baza u realnom vremenu[3] je idealna za brzi razvoj aplikacija. Ova baza je NoSQL baza podataka. Specifičnost ove baze podataka je ta, što svi klijenti dijele istu instancu baze te u realnom vremenu dobivaju nove informacije iz baze podataka. Ukoliko uređaj nije spojen na mrežu[10], podatci se spremaju lokalno, a naknadno prijavom na mrežu ti podatci se sinkroniziraju s Firebase bazom podataka, koja rješava eventualne konflikte između podataka. Podatci u bazi su strukturirani u obliku JSON stabla. Glavni čvor (Slika 7.) predstavlja čvor `users`. Djeca tog čvora su zapravo UID podatci korisnika, s obzirom da se dalje baza grana prema svakom korisniku. Baza je osmišljena tako da se uvijek mogu dodati novi korisnici unutar grupe korisnika, nove lokacije unutar grupe lokacija i novi senzori unutar grupe senzora, ali osim dodavanja, isti se mogu i brisati.



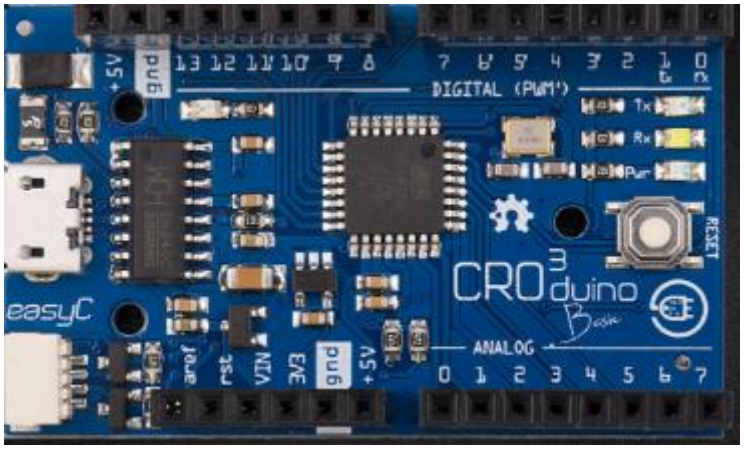
Slika 7. Prikaz strukture baze podataka

6. Funkcionalnost Arduino mikrokontrolera

Za prototip rješenja koristi se CROduino Basic 2 pločica te CROduino Nova pločica. Te dvije pločice osmišljene su i proizvedene u Republici Hrvatskoj, a proizvodi i prodaje ih internet trgovina e-radionica.com. Obje pločice su kompatibilne s Arduinom. Za prototip se koristi šest senzora, jedan elektromagnetni ventil, dvije eksperimentalne pločice, jedan relej, punjač od 12V te nekoliko desetaka komada žica za spajanje.

6.1. CROduino Basic 2

CROduino Basic 2 (Slika 8.) je standardna hrvatska Arduino kompatibilna pločica[12]. Idealna je za rad s elektronikom ili Arduinom. Pločica je kompatibilna s Arduino Nano pločicom. Mikrokontroler koji koristi je Atmelov ATmega328 koji pruža 22 ulazno-izlazna pina, od kojih je 14 digitalnih i 8 analognih pinova. Za komunikaciju putem USB kabela pločica koristi Silabs CP2102 USB to UART bridge sučelje, s kojim se vrlo lako povezuje s računalom, radi lakšeg programiranja pločice. Važno je napomenuti da pločica podržava I2C, SPI i serijsku komunikaciju.



Slika 8. CROduino Basic 2 pločica[12]

Za programiranje pločice koristi se Arduino IDE program. Prije *upload*-a kôda na pločicu, potrebno je u Arduino IDE programu definirati o kojoj pločici se radi. U izborniku *Tools*, pod opcijom *Board*, treba odabrati Arduino Nano, a pod opcijom *Processor* treba odabrati ATmega328 procesor. Nakon toga je dovoljno kliknuti na gumb *Upload* i pričekati da Arduino IDE obavi posao postavljanja kôda na pločicu.

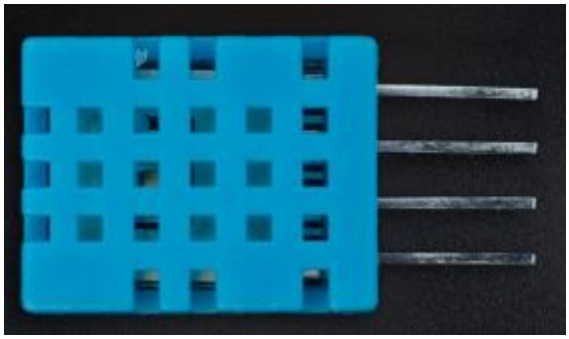
6.2. CROduino Nova

Slika 9. CROduino Nova pločica[13]

CROduino Nova (Slika 9.) je mikrokontrolerska pločica s mogućnošću konekcije na Wi-Fi mrežu[13]. Baziran je na ESP8266 modulu. Modul dolazi s TCP/IP stogom, koji podržava DNS. Pločica sadrži devet 'GPIO' pinova te jedan analogni ulaz. Pločica podržava I2C, SPI i serijsku komunikaciju. Za komunikaciju s računalom koristi micro USB, preko sučelja Silabs CP2102 USB to UART bridge.

Za programiranje pločice koristi se Arduino IDE program. Prije *Upload*-a kôda na pločicu potrebno je definirati pločicu unutar Arduino IDE programa. U izborniku *Tools*, pod opcijom *Board*, treba odabrati 'Generic ESP8266 module'. Preporuča se i za opciju *Reset mode* postaviti odabir 'nodemcu' zbog lakšeg *upload*-a kôda i pokretanja programa na pločici. Nakon definiranja pločice u programu, klikom na gumb *Upload*, kôd se postavlja na pločicu i pločica je spremna za korištenje.

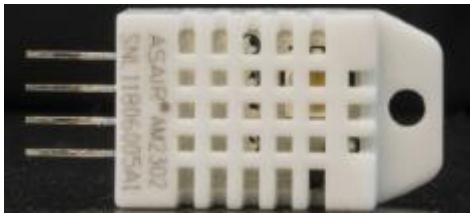
6.3. Senzor DHT11



Slika 10. Senzor DHT11[14]

Senzor DHT11 (Slika 10.) je najjednostavniji senzor za mjerenje temperature i vlage zraka[14]. Spaja se na pločicu jednostavnim spajanjem žica. Senzor na svom izlazu ima četiri nožice. Prva nožica, s lijeve strane, spaja se na '5V' pin pločice, druga nožica, s lijeve strane, spaja se na neki od digitalnih pinova na pločici. Prva nožica, s desne strane, spaja se sa 'gnd' pinom na pločici koja predstavlja uzemljenje. Senzor ima mogućnost mjerenja temperature u rasponu od 0-50 u jedinici stupanj celzijus, s preciznošću ±2°C. Vlažnost mjeri u postotcima u rasponu od 20-90, s preciznošću ±5%.

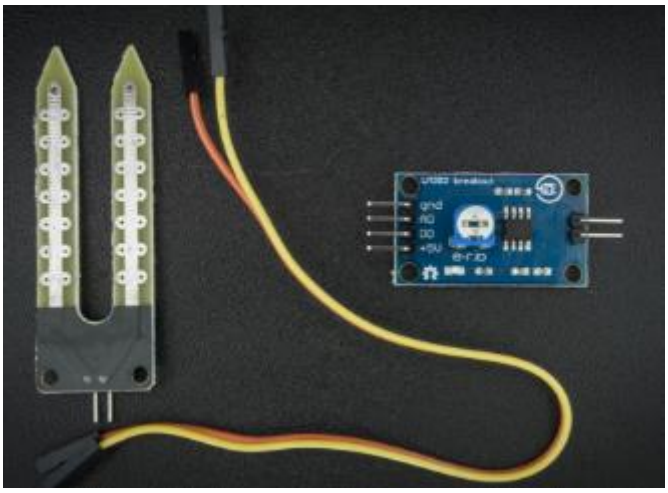
6.4. Senzor DHT22



Slika 11. Senzor DHT22[15]

Senzor DHT22 (Slika 11.) je senzor za mjerenje temperature i vlage u zraku[15]. Spaja se na pločicu isto kao i senzor DHT11. Prva nožica, s lijeve strane, spaja se na '5V' pin na pločici. Druga nožica, s lijeve strane, spaja se s odabranim digitalnim pinom na pločici. Prva nožica, s desne strane, spaja se na 'gnd' pin na pločici. Senzor mjeri temperaturu od -40°C do 80°C, s preciznošću od ±0.5°C. Vlažnost mjeri u rasponu od 0-100% s preciznošću od ±2%. To je nešto bolji senzor od senzora DHT11. Važno je još napomenuti da ovom senzoru treba neko vrijeme da dosegne radnu temperaturu, nakon koje se mogu očekivati očitanja temperature i vlažnosti.

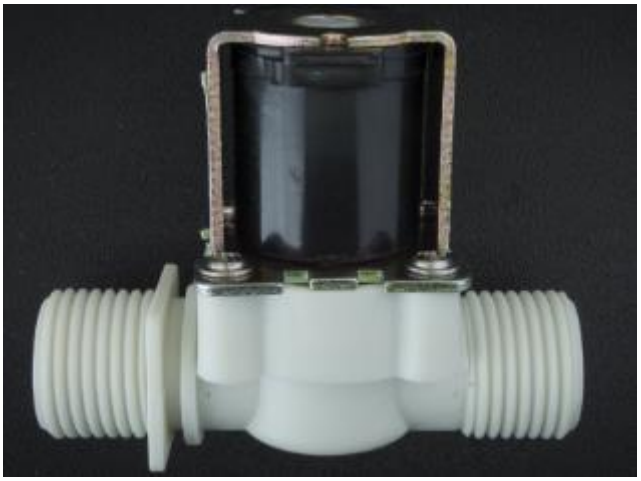
6.5. Senzor vlažnosti tla



Slika 12. Senzor vlažnosti tla[16]

Senzor za vlažnost tla (Slika 12.) je senzor koji mjeri količinu vlage u tlu[16]. Senzor, na sebi ili komparatoru na kojeg je spojen, ima označene nožice s oznakama '5V', 'gnd' i 'AO'. To olakšava spajanje na pločicu. Senzor na svojim izlazima, ovisno o količini vlage, šalje analogni signal (0V-5V), koji se prikazuje kao brojčana vrijednost u rasponu od 0-1023.

6.6. Elektromagnetni ventil

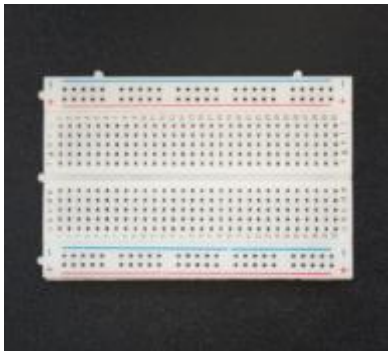


Slika 13. Elektromagnetni ventil[17]

Za potrebe automatizacije procesa rada s biljkama, koristi se elektromagnetni ventil (Slika 13.). Ovaj ventil[17] radi na jednostavnom principu, kada se ventilu propusti struja, ventil će se aktivirati i propustiti će tekućinu kroz cijev, a ako se dovod struje prekine, ventil će zatvoriti cijev i tekućina neće teći.

Još jedna stavka potrebna za aktiviranje ventila je i pritisak tekućine. Ventil prihvaća cijevi promjera ½, a na ulaznom dijelu nalazi se i filter za neželjene čestice. S obzirom da je napon na izlazu Arduino pločice 5V, potrebno je osigurati dodatan izvor napona, a to je moguće spajanjem kućnog punjača, od 9-12V, preko releja.

6.7. Eksperimentalna pločica



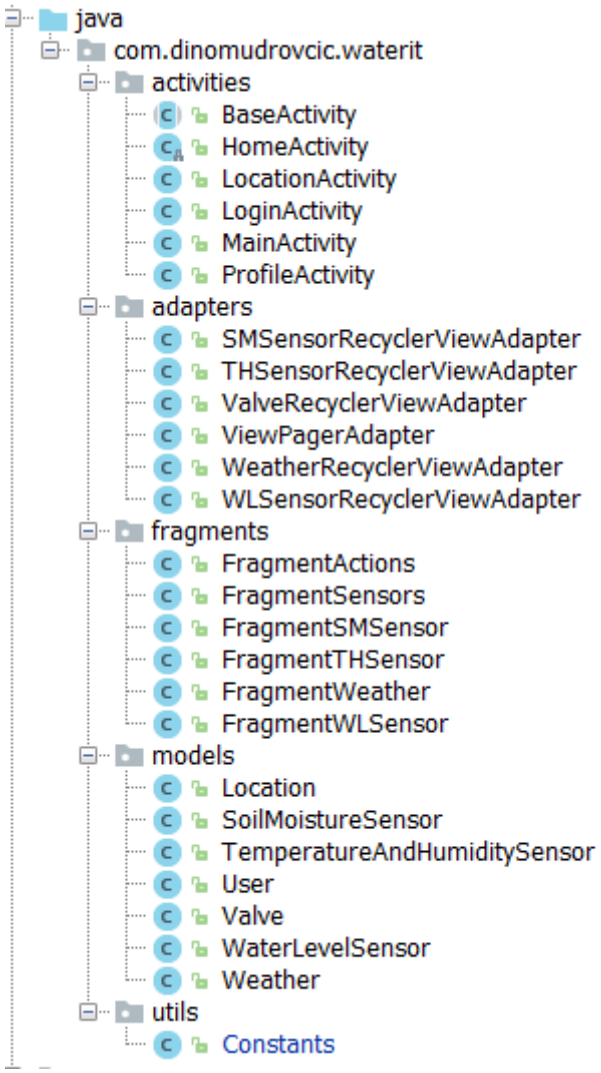
Slika 14. Eksperimentalna pločica[18]

Eksperimentalna pločica[18] (Slika 14.) omogućava jednostavno spajanje komponenti i modula, bez potrebe lemljenja te na taj način osigurava jednostavno testiranje, kreiranje prototipa i učenje.

7. Funkcionalnost Android aplikacija – WaterIT

Android aplikacija, na mobilnom uređaju s Android operacijskim sustavom, nosi naziv „WaterIT“. Aplikacija je pisana u programskom jeziku Java. Ova aplikacija je zamišljena i razvijena kao aplikacija za prikaz podataka sa senzora, prikaz vremenske prognoze i kontroliranje ventila. Korisničko sučelje je jednostavno za korištenje. Svaki korisnik se prijavljuje sa svojim podatcima, nakon kojih ima ponuđen odabir svojih lokacija. Nakon odabira lokacije, aplikacija korisniku nudi prikaz senzora, vremenske prognoze i kontrolu ventila. Također, svaki korisnik ima uvid u svoj profil.

7.1. Struktura Android aplikacije



Slika 15. Prikaz strukture Android aplikacije

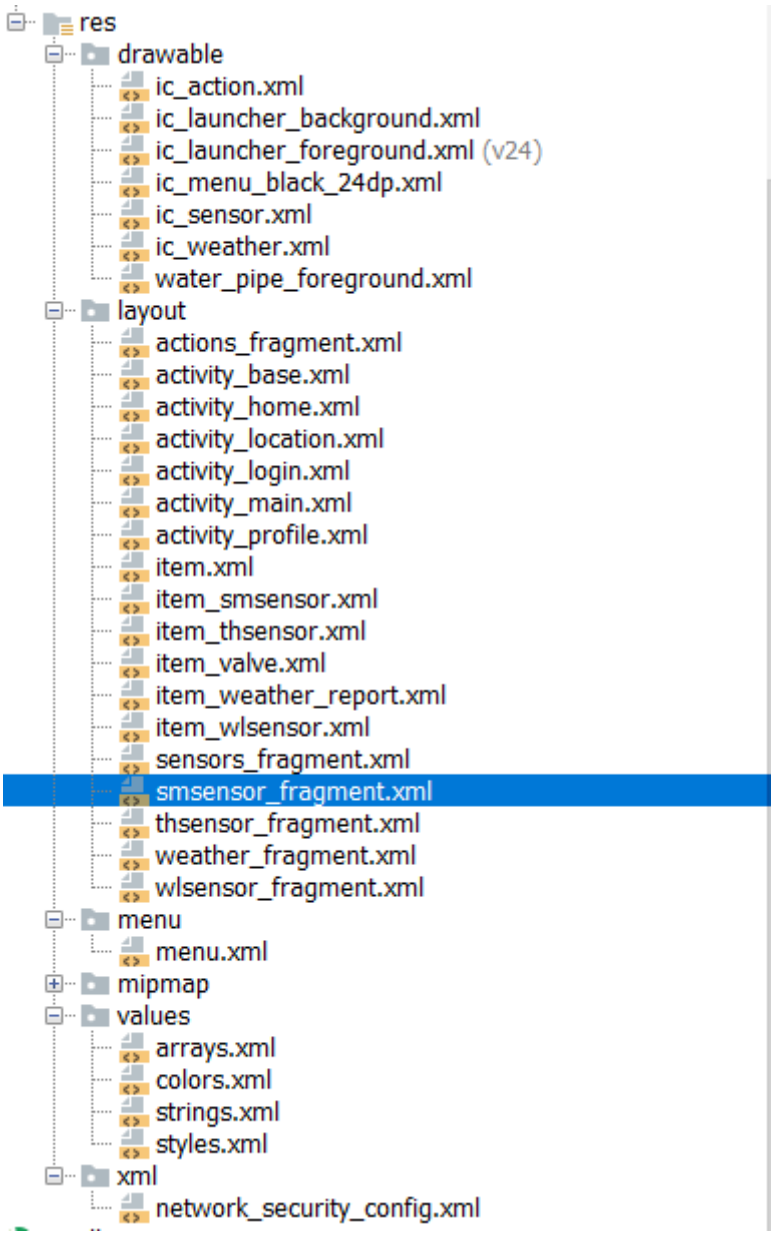
Struktura (Slika 15.) prikazuje način na koji su posloženi paketi i klase. Paketi su podijeljeni na pakete aktivnosti, adaptera, fragmenata, modela i pomoćni *utils* paket. Paket aktivnosti sadrži šest aktivnosti u aplikaciji koje predstavljaju pet ekrana. Početni ekran definiran je u `HomeActivity.java` klasi. Paket adaptera predstavlja klase, koje definiraju način na koji će određeni podatci biti prikazani, u korisničkom sučelju pojedine aktivnosti ili fragmenta. Paket fragmenata predstavlja šest fragmenata koji se koriste u aplikaciji. Fragmenti su predviđeni za višekratnu upotrebu prikaza podataka o senzorima, vremenskoj prognozi i ventilu. Paket modela je skup klasa koje definiraju izgled i upravljanje podatcima iz baze podataka.

```
import com.google.firebase.database.IgnoreExtraProperties;
import java.util.HashMap;
@IgnoreExtraProperties
public class User {
    public String display_name;
    public String email;
    public String full_name_or_company_name;
    public HashMap<String, Location> locations;
    public User(){}
    public User(String displayName, String email, String
                fullNameOrCompanyName, HashMap<String,
                Location> locations){
        this.display_name = displayName;
        this.email = email;
        this.full_name_or_company_name = fullNameOrCompanyName;
        this.locations = locations;
    }
}
```

Kôd 4. Prikaz klase User

Prikazani kôd (Kôd 4.) je primjer modela `User`. Model se sastoji od četiri atributa i dva konstruktora. Nazivi atributa su jednaki nazivima atributa iz baze podataka iz razloga što Firebase baza podataka funkcionira na taj način, odnosno potrebno je imati iste nazive atributa radi povezivanja podataka. Također, prazan konstruktor je potreban kako bi se kreirala instanca klase prilikom dohvata podataka iz baze podataka. Drugi konstruktor je tu zbog lakše inicijalizacije objekta klase. Pomoćni, *utils*, paket sadrži klasu koja sadrži sve konstante koje se koriste u aplikaciji.

7.2. Resursi Android aplikacije



Slika 16. Prikaz strukture resursa Android aplikacije

Struktura resursa Android aplikacije (Slika 16.) prikazuje resurse koji se koriste u Android aplikaciji. Resursi su zapravo `XML` datoteke, u kojima je opisan dizajn pojedinih komponenti. Paket *drawable* sadrži ikonice, koje su korištene u aplikaciji. Paket *layout* je skup `XML` datoteka koje predstavljaju dizajn aplikacije, odnosno njenih aktivnosti i fragmenata. Tu su također i datoteke koje predstavljaju dizajn prikaza podataka u listi.

7.3. *Build* sustav - Gradle

Gradle je prilagođen alat za *build* Android aplikacije. Potrebno je navesti sve zavisnosti (engl. *dependencies*) te neke konfiguracije kako bi Gradle napravio *build*. Neke od konfiguracija koje treba definirati su SDK verzija, koja zapravo predstavlja API *level* (Slika 4.).

```
compileSdkVersion 28
    defaultConfig {
        applicationId "com.dinomudrovcic.waterit"
        minSdkVersion 22
        targetSdkVersion 28
        versionCode 1
        versionName "1.0.0"
        testInstrumentationRunner
      "android.support.test.runner.AndroidJUnitRunner"
    }
```

Kôd 5. Konfiguracija Gradle-a

Definiranje *dependency*-a u Gradle *build*-u je važno kako bi se koristile vanjske biblioteke za razvoj programskog kôda u aplikaciji ili kako bi se aplikacija spojila na bazu podataka. U aplikaciji su korišteni *dependency*-i za korištenje biblioteka potrebnih za spajanje na Firebase bazu podataka.

```
implementation 'com.google.firebase:firebase-database:11.0.4'
implementation 'com.google.firebase:firebase-storage:11.0.4'
implementation 'com.google.firebase:firebase-auth:11.0.4'
implementation 'com.firebaseui:firebase-ui-storage:2.0.1'
```

Kôd 6. Definiranje *dependency*-a

7.4. Klase i komponente

Neke od važnijih klasa i komponenti korištenih u radu, koje su važne za spomenuti i objasniti:

Klase:

- `FirebaseAuth`
- `DatabaseReference`
- `ButterKnife`
- `Intent`

Elementi:

- `RecyclerView`
- `LinearLayout`
- `TabLayout`
- `ViewPager`
- `Spinner`
- `TextView`
- `Button`
- `ImageView`
- `EditText`
- `ProgressBar`

`FirebaseAuth` je klasa pomoću koje se obavlja ovjera autentičnosti aplikacije prema instanci Firebase baze podataka.

`DatabaseReference` je klasa pomoću koje se dohvaća određena lokacija iz Firebase baze podataka za koju je moguće čitanje ili pisanje podataka.

`ButterKnife` je *injection* biblioteka koja omogućava povezivanje polja i metoda za prikaze u Androidu.

`Intent` je klasa koja predstavlja apstraktni opis operacije koja će biti izvršena.

`TextView` je komponenta dizajna aplikacije, koja se koristi za prikaz teksta. Važni atributi ove komponente su: ID, širina, visina, pozicija teksta te veličina teksta.

`EditText` je komponenta dizajna aplikacije, koja se koristi za prikaz i unos teksta. Važni atributi ove komponente su: ID, širina, visina, veličina teksta te tip unosa.

`ImageView` je komponenta dizajna aplikacije, koja se koristi za prikaz slika. Važni atributi ove komponente su: ID, širina te visina.

`Button` je komponenta dizajna aplikacije, koja se koristi za prikaz gumba. Važni atributi ove komponente su: ID, visina, širina, pozicija, stil te tekst na gumbu.

`LinearLayout` je komponenta dizajna aplikacije, koja se koristi za grupu prikaza drugih komponenti kao što su: `TextView`, `EditText`, `ImageView` ili `Button`. Važni atributi ove komponente su: ID, visina, širina, orijentacija, stil te pozadina.

`ProgressBar` je komponenta dizajna aplikacije, koja se koristi za prikaz učitavanja podataka. Kada korisnik mora pričekati na učitavanje, ovo je idealno grafičko rješenje koje korisniku prikazuje da se nešto odrađuje u pozadini. Važni atributi ove komponente su: ID, visina, širina te vidljivost.

`Spinner` je komponenta dizajna aplikacije, koja se koristi za prikaz liste pomoću padajućeg izbornika. Važni atributi ove komponente su: ID, visina i širina.

`TabLayout` je komponenta dizajna aplikacije, koja se koristi za prikaz dijelova aplikacije unutar grupe kartica. Važni atributi ove komponente su: ID, visina, širina, pozadina te stil.

`ViewPager` je komponenta dizajna aplikacije, koja se koristi za prebacivanje s kartice na karticu, pomicanjem lijevo-desno. Važni atributi ove komponente su: ID, visina i širina.

`RecyclerView` je komponenta dizajna aplikacije, koja se koristi za prikaz lista podataka u aplikaciji. Važni atributi ove komponente su: ID, visina i širina.

8. Implementacija Arduino mikrokontrolera

8.1. Spajanje CROduino Basic 2 mikrokontrolera i senzora

Slika 17. CROduino Basic 2 sa spojenim senzorima

Za spajanje CROduino Basic 2 pločice i potrebnih senzora (Slika 17.), koristi se eksperimentalna pločica te nekoliko žica. DHT11 i DHT22 senzori spojeni su preko digitalnih pinova, dok su tri senzora za vlažnost tla te jedan senzor za visinu tekućine, spojeni preko analognih pinova. Za kontrolu struje koja putuje iz pločice prema senzoru koristi se otpornik 10kΩ, čisto iz sigurnosnih razloga, da se ne ošteti senzor.

Prvi korak je spajanje pina '5V' s CROduino Basic 2 pločice s rupicom pozitivnog stupca na eksperimentalnoj pločici. Zatim je potrebno spojiti 'gnd' pin s pločice s rupicom negativnog stupca na eksperimentalnoj pločici. Na ovaj način je osigurano da svaki senzor ima konekciju sa '5V' naponom i 'gnd' uzemljenjem. CROduino Basic 2 ima samo dva pina od '5V' i dva pina 'gnd' uzemljenja. Eksperimentalna pločica u ovom slučaju dobro dođe jer bi se u suprotnom mogla spojiti samo dva senzora.

DHT11 i DHT22 senzori su sa svojih 'DO' nožica spojeni na digitalne pinove '7' i '8' na CROduino Basic 2 pločici. 'Vcc' i 'gnd' nožice sa senzora spojene su u pozitivan i negativan stupac na eksperimentalnoj pločici.

Četiri senzora koja idu u zemlju i šalju analogni signal, spojeni su na analogne pinove CROduino Basic 2 pločice na 'A1', 'A3', 'A5' i 'A7' pin. 'Vcc' i 'gnd' nožice s tih senzora su također spojene u pozitivan i negativan stupac na eksperimentalnoj pločici.

Sljedeći korak je programiranje pločice. Za DHT11 i DHT22 senzore potrebno je preuzeti i uključiti vanjske biblioteke u projekt. Radi se o bibliotekama `DHT.h` i `DHT_U.h`. To su C++ biblioteke, napisane za lakše korištenje senzora te sadrže metode za očitavanje vrijednosti sa tih senzora. Potrebno je u kôdu definirati pinove za te senzore na sljedeći način:

```
#define dataPin22 7
#define dataPin11 8
DHT dht11 (dataPin11, DHT11);
DHT dht22 (dataPin22, DHT22);
```

Prvo se, pomoću `#define`, definiraju brojevi digitalnih pinova na pločici. Nakon toga se, kroz konstruktor `DHT` klase, definira na kojem pinu je koji tip senzora. U `setup` metodi potrebno je još napisati sljedeće linije kôda:

```
dht11.begin();
dht22.begin();
```

Senzori koji koriste analogne pinove, te iste pinove definiraju u `loop` metodi.

U `loop` (Kôd 7.) metodi nalaze se instrukcije za čitanje podataka sa senzora. Podatci sa senzora DHT11 i DHT22 čitaju se pomoću metoda `readTemperature` i `readHumidity`. Rezultati koje vraćaju te metode su tipa `float`, pa se zato spremaju u varijable tog tipa.

Kod analognog čitanja, podatci sa senzora spremaju se u varijablu tipa `double`. Za čitanje podataka koristi se ugrađena metoda `analogRead`, koja kao parametar prima analogni pin, na koji je senzor spojen.

Između svakog očitanja podataka sa senzora, nalazi se `delay(100)` poziv, zbog toga što nekad, podatci sa senzora mogu kasniti neko kratko vrijeme, pa je ovo dobar način zaobilaženja tog problema.

```
double sensorIntData = 0;
//first sensor
float t1 = dht22.readTemperature();
delay(100);
float h1 = dht22.readHumidity();
delay(100);
//second sensor
float t2 = dht11.readTemperature();
delay(100);
float h2 = dht11.readHumidity();
delay(100);
//third sensor
sensorIntData = analogRead(A1);
delay(100);
double sensorValue1 = calculateNonER(sensorIntData);
//forth sensor
sensorIntData = analogRead(A3);
delay(100);
double sensorValue2 = calculateNonER(sensorIntData);
//fifth sensor
sensorIntData = analogRead(A5);
delay(100);
double sensorValue3 = calculateER(sensorIntData);
//sixth sensor
sensorIntData = analogRead(A7);
delay(100);
double sensorValue4 = calculateER(sensorIntData);
```

Kôd 7. Čitanje podataka sa senzora

S ovim je kompletirano spajanje senzora sa CROduino Basic 2 pločicom te očitavanje podataka sa senzora putem instrukcija u `loop` metodi.

8.2. Spajanje CROduino Basic 2 mikrokontrolera i CROduino Nova mikrokontrolera putem serijske komunikacije

CROduino Basic 2 koristi se samo za očitanje podataka sa senzora. Kako bi ti podatci završili u bazi podataka, potrebno je osigurati povezivanje na internet. CROduino Nova mikrokontroler idealan je za spajanje na internet i slanje i primanje podataka sa interneta, odnosno iz baze podataka. Međutim, tu se javlja problem kako prebaciti podatke s CROduino Basic 2 mikrokontrolera na CROduino Nova mikrokontroler. S obzirom da oba mikrokontrolera podržavaju serijsku komunikaciju, te se na njima nalaze 'RX' i 'TX' pinovi, spajanje i komunikacija između te dvije pločice postaje vrlo jednostavna. Dovoljno je na obje pločice definirati serijski port preko kojega će se obavljati komunikacija. Dalje se jednostavno šalju podatci s jedne pločice na drugu.

Na CROduino Basic 2 mikrokontroleru potrebno je preuzeti i uključiti biblioteku `SoftwareSerial.h`, koja omogućuje otvaranja serijskog porta na toj pločici. Zatim je potrebno definirati naziv porta i pinove koje će koristiti.

```
SoftwareSerial Serial1(0,1);
```

U `setup` metodi potrebno je napraviti inicijalizaciju serijskog porta.

```
Serial1.begin(9600);
```

Zatim se u `loop` metodi postavlja pomoćna varijabla `String sensorData;` na koju će se dodavati podatci sa senzora. Konstrukcija teksta koji će se slati je po redu kako se očitavaju senzori u `loop` metodi. Svaki očitani podatak se razdvaja zarezom. Na kraju je još potrebna linija teksta za slanje preko serijske komunikacije. Za to se koristi metoda `write`.

```
char *str = (char*)
malloc(sizeof(char)*sensorData.length()+1);
sensorData.toCharArray(str, sensorData.length()+1);
Serial1.write(str);
free(str);
```

Na CROduino Nova mikrokontroleru potrebno je otvoriti serijski port za komunikaciju koji će primiti liniju teksta s CROduino Basic 2 mikrokontrolera. U `setup` metodi se inicijalizira serijski port na koji će se primati podatci putem serijske komunikacije.

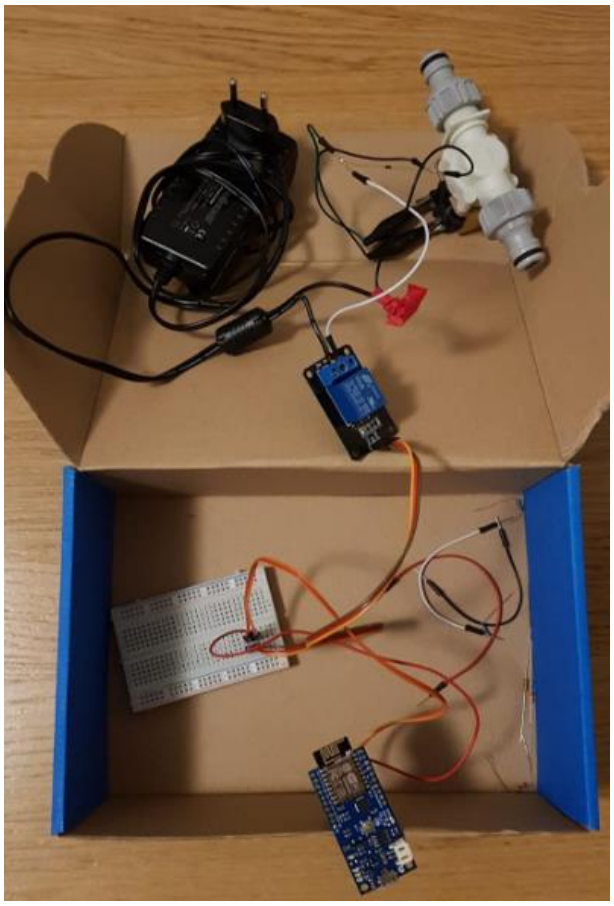
```
Serial.begin(9600);
```

U `loop` metodi prvo se provjerava da li postoji komunikacija na serijskom portu. Nakon toga, čita se podatak sa serijskog porta. S obzirom da se na tom portu očekuje linija teksta s podatcima odvojenim zarezima, koristi se `StringSplitter` klasa, koja razdvaja ulaznu liniju teksta na osam dijelova odvojenih zarezom.

```
if(Serial.available()){
    delay(100);
    String readSerial = Serial.readString();
    StringSplitter *splitter = new StringSplitter(readSerial,
',', 8);
}
```

Kako bi se ove instrukcije izvršavale na pločicama, potrebno je prvo *upload*-ati kôd na svaku pločicu te obavezno poslije *upload*-a spojiti, prvo 'TX' pin na CROduino Basic 2 pločici s 'RX' pinom na CROduino Nova pločici. Zatim se spaja 'TX' pin na CROduino Nova pločici s 'RX' pinom na CROduino Basic 2 pločici. I zadnji korak je međusobno spajanje 'gnd' pinova na obje pločice.

8.3. Spajanje CROduino Nova mikrokontrolera s elektromagnetnim ventilom



Slika 18. CROduino Nova sa spojenim elektromagnetnim ventilom

Za spajanje CROduino Nova mikrokontrolera i elektromagnetnog ventila (Slika 18.), koristi se eksperimentalna pločica, jedan relej, izvor struje, dva krokodil konektora i nekoliko žica. Pločica preko releja kontrolira otvaranje i zatvaranje ventila. Budući da je napon premali za ovaj ventil, potrebno je dodati punjač koji može dati napon od 12V. Punjač se pozitivnom žicom spaja u relej, a uzemljenjem se spaja na uzemljenje koje ide iz ventila. Relej se preko eksperimentalne pločice spaja na '5V' i 'gnd' pinove na CROduino Nova pločici te na 'GPIO 15' pin za kontrolu ventila. Između releja i ventila postavljena je i dioda. Razlog postavljanja diode je izbjegavanje uništavanja releja, budući da u situaciji kada se relej isključi, može doći do pojave velikog skoka napona koji je negativan, što može trajno oštetiti relej.

Kod programiranja instrukcija za upravljanje ventilom putem pločice, potrebno je definirati pin preko kojega će se instrukcija izvršavati.

```
#define VALVE 15
```

Za otvaranje i zatvaranje ventila preko pina '15', koristi se metoda `digitalWrite`. Ova metoda prima parametre pina koji se koristi i stanja. Stanja kod digitalnih pinova su 'HIGH' i 'LOW'. Kada ventil treba otvoriti, koristi se 'LOW' stanje, a kada je ventil potrebno zatvoriti, koristi se 'HIGH' stanje. Preporuča se pozivanje `delay(1000);` poziva kako bi se instrukcija stigla izvršiti.

```
digitalWrite(VALVE, LOW);
delay(1000);
digitalWrite(VALVE, HIGH);
delay(1000);
```

8.4. Spajanje CROduino Nova mikrokontrolera s Firebase bazom podataka

Spajanje CROduino Nova mikrokontrolera na internet preko Wi-Fi modula je vrlo jednostavno. Sve što treba je preuzeti i uključiti biblioteku `ESP8266.WiFi.h`. Zatim je potrebno definirati mrežu na koju će se mikrokontroler spojiti te lozinku za spajanje na tu mrežu.

```
const char* WIFI_SSID = "AndroidAP";
const char* WIFI_PASSWORD = "12345678";
```

Nakon toga u `setup` metodi se poziva:

```
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
```

U ovom trenutku, mikrokontroler je spojen na mrežu 'AndroidAP' i spreman je za komunikaciju na internetu.

Za spajanje Firebase baze podataka sa CROduino Nova pločicom potrebno je preuzeti i uključiti sljedeće biblioteke:

- `FirebaseArduino.h`
- `ArduinoJson.h`

Nakon uključivanja tih biblioteka, potrebno je definirati Firebase varijablu, Firebase *host* i Firebase autentifikaciju.

```
FirebaseArduino firebaseStream;
const char* FIREBASE_HOST = "android-arduino-
final.firebaseio.com";
const char* FIREBASE_AUTH =
"9NW5rYj4bU9A01Fknq4bNGrO0GNkwmQn9yu6g6XR";
```

Potrebno je definirati putanje do čvora na kojemu će, Firebase baza podataka, spremiti podatke koje dobije s mikrokontrolera. S obzirom na konstrukciju baze podataka, potrebno je definirati više čvorova (Kôd 8.).

```
String FIREBASE_USER = "/users/33LCOXefbRXSmjOzrIMtTalUTDg1";
String LOCATION = "/locations/Lokacija Marija Gorica";
String PATH_TO_VALVE = FIREBASE_USER + LOCATION +
                       "/valves/valve1/status"; //valve
String PATH_TO_SMSENSOR_SENSOR_1 = FIREBASE_USER + LOCATION +
                       "/soil_moisture_sensors/sensor1"; //A7
String PATH_TO_SMSENSOR_SENSOR_2 = FIREBASE_USER + LOCATION +
                       "/soil_moisture_sensors/sensor2"; //A5
String PATH_TO_SMSENSOR_SENSOR_3 = FIREBASE_USER + LOCATION +
                       "/soil_moisture_sensors/sensor3"; //A3
String PATH_TO_THSENSOR_SENSOR_1 = FIREBASE_USER + LOCATION +
                       "/temperature_and_humidity_sensors/sensor1";
                       //DHT11
String PATH_TO_THSENSOR_SENSOR_2 = FIREBASE_USER + LOCATION +
                       "/temperature_and_humidity_sensors/sensor2";
                       //DHT22
String PATH_TO_WATER_LEVEL_SENSOR = FIREBASE_USER + LOCATION +
                       "/water_level_sensors/sensor1"; //A1
```

Kôd 8. Definiranje čvorova za Firebase bazu podataka

U metodi `setup`, potrebno je pokrenuti komunikaciju s Firebase bazom podataka.

```
Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
```

Slanje podataka s mikrokontrolera u bazu podataka obavlja se pomoću metode `setFloat`, koja za parametre ima putanju do atributa u bazi podataka i podatak, koji će na tu lokaciju spremiti. S obzirom da je podatak na mikrokontroleru tipa `String`, potrebno je pretvoriti vrijednost u `float`.

```
Firebase.setFloat(PATH_TO_THSENSOR_SENSOR_2 + "/temperature",
splitter->getItemAtIndex(0).toFloat());
```

8.5. Kontrola elektromagnetnog ventila

Mikrokontroler CROduino Nova kontrolira stanje ventila pomoću podatka iz Firebase baze podataka. Ako je podatak *Open*, mikrokontroler otvara ventil, a ako je podatak *Closed*, mikrokontroler će zatvoriti ventil. Međutim, tu se javlja jedan problem. Mikrokontroler periodično u `loop` petlji provjerava stanje u Firebase bazi podataka te promjenu na podatku o statusu ventila. To može potrošiti dosta internet prometa, pa je idealno rješenje za ovaj problem korištenje `stream` metode. Ona se ponaša kao *Listener* koji sluša promjene na Firebase bazi podataka i onda kada dobije informaciju da se podatak promijenio, pokreće upit prema bazi podataka. Prvo je potrebno pokrenuti slušanje na mikrokontroleru, a to se dešava u `setup` metodi.

```
firebaseStream.begin(FIREBASE_HOST, FIREBASE_AUTH);
firebaseStream.stream(PATH_TO_VALVE);
```

Zatim je, u `loop` metodi, potrebno zadati instrukciju koja sluša da li postoji promjena na podatku kojeg se traži. Ako promjena postoji, pokrenuti će se upit prema bazi podataka.

```
if(firebaseStream.available()){
      readValveStatus();
}
```

`ReadValveStatus` je privatna metoda koja čita status iz baze podataka i na temelju njega otvara ili zatvara ventil.

```
void readValveStatus(){
      String valveStatus = Firebase.getString(PATH_TO_VALVE);
      delay(2000);
      if(valveStatus == "Open"){
            digitalWrite(VALVE, LOW);
      }else{
            digitalWrite(VALVE, HIGH);
      }
      delay(1000);
}
```

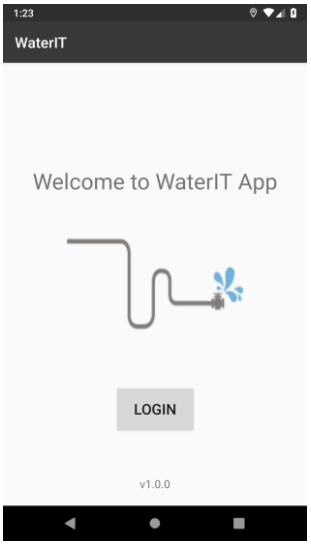
Kôd 9. Kontrola elektromagnetnog ventila

9. Implementacija Android aplikacije – WaterIT

9.1. Pokretanje aplikacije

Slika 19. WaterIT ikona

Aplikacija se pokreće klikom na ikonu (Slika 19.) koja se nalazi u prikazu aplikacija na Android uređaju. Početni ekran koji se prikazuje se sastoji od poruke dobrodošlice, naziva aplikacije, slike cijevi s ventilom, login gumbom i prikazom verzije aplikacije (Slika 20.).



Slika 20. Prikaz početnog ekrana

```
public class MainActivity extends AppCompatActivity {
    @BindView(R.id.btnSignIn)
    Button btnSignIn;
    @BindView(R.id.tvVersionNo)
    TextView tvVersionNo;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ButterKnife.bind(this);
        tvVersionNo.setText("v" + BuildConfig.VERSION_NAME);
        btnSignIn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                  startActivity(newIntent(MainActivity.this,
                  LoginActivity.class));
            }
        });
    }
}
```

Kôd 10. Početni ekran

Početni ekran sastoji se od `LinearLayout` komponente (Kôd 11.), unutar koje se nalaze tri komponente, dva `TextView`-a i jedan `Button`. Važni atributi komponenti su `android:id`, `android:layout_width`, `android:layout_height`. `Android:id` je potreban kako bi se toj komponenti moglo pristupiti u kôdu, dok su preostale dvije zapravo jedinice veličine komponente.

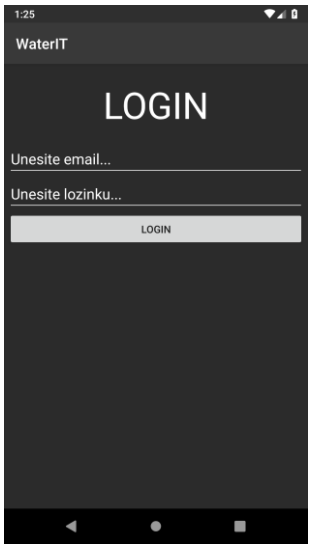
Za povezivanje komponenti u kôdu s pripadajućim ID-evima, koristi se biblioteka `ButterKnife`. Za definiranje varijable u kôdu (Kôd 10.), koristi se anotacija `@BindView`, dok se za povezivanje koristi `ButterKnife.bind(this);`.

Klikom na gumb *Login*, pokreće se nova aktivnost. Sam klik na gumb je riješen postavljanjem `setOnClickListener` metode, koja sluša aktivnost klika gumba. Pokretanje nove aktivnosti radi se pomoću metode `startActivity`, koja ima parametar `Intent`. Tu se kreira instanca klase `Intent`, s parametrima aktivnosti u kojoj je aplikacija trenutno te aktivnosti koja će se otvoriti. S obzirom da je aplikacija u *`MainActivity`* aktivnosti i želi prijeći u aktivnost prijave korisnika, parametri su `MainActivity.this` i `LoginActivity.class`.

```
<Button
    android:id="@+id/btnSignIn"
    android:layout_width="111dp"
    android:layout_height="68dp"
    android:text="LOGIN"
    android:textSize="20dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintHorizontal_bias="0.501"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.817" />
<TextView
    android:id="@+id/tvVersionNo"
    android:layout_width="match_parent"
    android:layout_height="40dp"
    android:layout_marginEnd="8dp"
    android:layout_marginStart="8dp"
    android:textAlignment="center"
    android:textSize="15dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent" />
```

Kôd 11. Komponente na početnom zaslonu

9.2. Prijava korisnika



Slika 21. Prikaz ekrana za prijavu

```
firebaseAuth.signInWithEmailAndPassword(etEmail.getText().toString(),
            etPass.getText().toString())
            .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
      @Override
      public void onComplete(@NonNull Task<AuthResult> task) {
            pbLogin.setVisibility(View.GONE);
            if(task.isSuccessful()){
            startActivity(new Intent(LoginActivity.this,
                                    HomeActivity.class));
           }else{
                  etPass.setText("");
                  Toast.makeText(LoginActivity.this,
                  task.getException().getMessage(),
                  Toast.LENGTH_LONG).show();
           }
      }
});
```

Kôd 12. Prijava korisnika

Ekran za prijavu (Slika 21.) sastoji se od naslova, polja za unos email adrese, polja za unos lozinke te gumba za potvrdu prijave. Polje za unos email adrese sadrži validaciju email-a, prema tome bilo koji pogrešan unos email adrese i pokušaj prijave rezultirati će pogreškom.

Pogrešna prijava unosom pogrešne lozinke, također će rezultirati greškom da lozinka nije ispravna. Ukoliko su polja email i lozinka prazni, prilikom pokušaja prijave doći će do greške da su polja prazna. Spomenute greške su greške koje vraća Firebase Auth API, na pokušaje prijave.

Unosom ispravne email adrese i lozinke i pritiskom na gumb *Login*, ispod gumba će se pojaviti *loader*, koji simbolizira prijavu i vrijeme čekanja. Nakon što je prijava obavljena, aplikacija će otvoriti sljedeći ekran.

Metoda `signInWithEmailAndPassword` (Kôd 12.) je metoda koja se poziva kako bi se izvršila prijava korisnika. Ona prima parametre email adrese i lozinke. Na nju se dodaje *Listener*, koji sluša korisnikovu prijavu. Ukoliko je prijava uspješna, aplikacija pokreće novu aktivnost, `HomeActivity.class`. Ako je prijava neuspješna, aplikacija putem `Toast`-a vraća poruku greške, `task.getException().getMessage()`.

9.3. `BaseActivity` aktivnost

Aktivnost `BaseActivity` je aktivnost-roditelj preostalim aktivnostima u aplikaciji. Ova aktivnost nasljeđuje `AppCompactActivity`. Aktivnost sadrži traku pri vrhu ekrana na kojoj se nalaze naslov ekrana te izbornik. Izbornik se sastoji od opcija odabira aktivnosti `ProfileActivity` te opcije odjave korisnika. Također, aktivnost nudi implementiranu *Back* opciju, za povratak na prethodnu aktivnost kada korisnik tako odluči.

Ovu aktivnost nasljeđuju sljedeće aktivnosti:

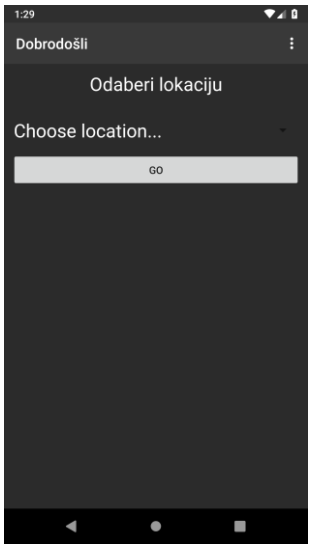
- `HomeActivity`
- `LocationActivity`
- `ProfileActivity`

```
protected void setTitleBar(String title, boolean backEnabled)
{
        ActionBar actionBar = getSupportActionBar();
        actionBar.setTitle(title);
        actionBar.setDisplayHomeAsUpEnabled(backEnabled);
        actionBar.setElevation(0);
}
```

Kôd 13. Postavljanje naslova ekrana

Svaki ekran koji nasljeđuje ovu aktivnost će imati svoje ime. Kako bi to bilo moguće, u ovoj aktivnosti nalazi se metoda `setTitleBar` (Kôd 13.), koja postavlja parametar `title`. Također sadrži parametar `backEnabled`, koji je tipa `Boolean`, i govori da li će gumb *Back*, biti prikazan na ekranu.

9.4. Odabir lokacije



Slika 22. Prikaz ekrana za odabir lokacije

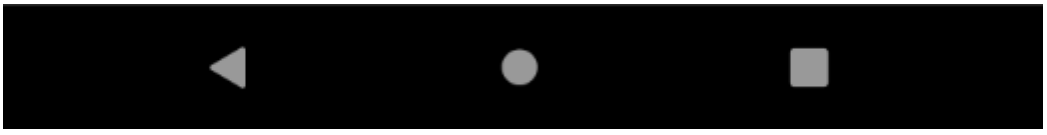
Ekran (Slika 22.) nudi odabir ponuđenih lokacija, za prijavljenog korisnika, iz baze podataka. Ekran također sadrži naslov i izbornik koji je naslijedio nasljeđivanjem aktivnosti `BaseActivity`.

S obzirom da se podaci o listi lokacija učitavaju samo prilikom kreiranja aktivnosti, koristi se metoda `addListenerForSingleValueEvent`. Ta metoda se samo jednom spaja na bazu i dohvaća podatke iz nje. Ako podatci postoje (Kôd 14.), baza podataka vraća podatke koji se spremaju u klasu `User`, te ispisuje listu u `Spinner`. Ispisivanje liste u `Spinner` se radi pomoću klase `ArrayAdapter<String>`.

```
if(dataSnapshot.exists()){
User user = dataSnapshot.getValue(User.class);
      for(Location location : user.locations.values()){
            city = location.city;
           country = location.state;
      }
      for(String location : user.locations.keySet()){
           locations.add(location);
      }
ArrayAdapter<String> adapter = new
ArrayAdapter<String>(HomeActivity.this, R.layout.item,
R.id.spinnerItem, locations);
adapter.setDropDownViewResource(R.layout.item);
                    spLocation.setAdapter(adapter);
}
```

Kôd 14. Dohvat i prikaz lokacija

Također, u ovoj aktivnosti nije implementiran *Back* gumb, u gornjem lijevom kutu, za povratak na prethodnu aktivnost. Povratak na prethodnu aktivnost bi značio ili izlazak iz aplikacije ili odjavu korisnika. U slučaju da se korisnik želi odjaviti, u gornjem desnom kutu ekrana nalazi se izbornik s opcijom odjave korisnika. U slučaju da se želi odjaviti i izaći iz aplikacije, korisnik mora pritisnuti *Back* (strelica ulijevo) gumb u donjem dijelu ekrana, gdje se nalazi navigacijska traka Android sustava (Slika 23.).



Slika 23. Navigacijska traka Android mobilnog uređaja

Odabirom lokacije u padajućem izborniku te pritiskom na gumb *Go*, pokrenuti će se iduća aktivnost, `LocationActivity`.

9.5. Prikaz podataka lokacije

Ekran za prikaz podataka lokacije je podijeljen na tri dijela:

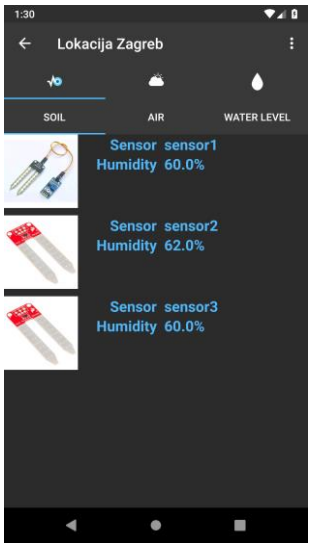
- Podatci sa senzora
- Podatci o vremenskoj prognozi
- Podatci o statusu ventila i kontrola ventila

Ti dijelovi su zapravo fragmenti, koji se pokreću kada se pokreće aktivnost `LocationActivity`. Fragmenti su prikazani zasebno unutar `TabLayout` elementa, a oni se definiraju pomoću `ViewPagerAdapter`-a. Podatci sa senzora, u fragmentu `FragmentSensors`, također su podijeljeni na tri fragmenta, kako bi prikazali podatke o senzorima iz zemlje, senzorima za zrak i senzorima za visinu tekućine. Ovi fragmenti su također prikazani unutar `TabLayout` elementa. Svaki fragment ima mogućnost prikaza liste objekata, zbog čega nam je potreban `RecyclerView`. Za svaki fragment se definira `RecyclerViewAdapter`, koji je potreban za prikazivanje podataka na korisničkom sučelju.

```
FragmentSensors sensors = new FragmentSensors();
sensors.setArguments(bundle);
FragmentWeather weather = new FragmentWeather();
weather.setArguments(bundle);
FragmentActions actions = new FragmentActions();
actions.setArguments(bundle);
adapter.AddFragment(sensors, "");
adapter.AddFragment(weather, "");
adapter.AddFragment(actions, "");
```

Kôd 15. Inicijalizacija fragmenata

Prvi fragment koji se otvara, u aktivnosti `LocationActivity`, je fragment koji prikazuje podatke sa senzora, unutar kojeg se otvara njegov *child* element, `FragmentSMSensor`, koji predstavlja podatke sa senzora koji su u zemlji i mjere vlažnost zemlje (Slika 24.). Fragment predstavlja ikonica impulsa koja simbolizira podatke sa senzora.



Slika 24. Prikaz podataka senzora iz zemlje

```
locationRef = FirebaseDatabase.getInstance().getReference(path +
                              Constants.SMSENSOR_PATH_ADDITION);
ValueEventListener sensorListener = new ValueEventListener() {
      @Override
      public void onDataChange(DataSnapshot dataSnapshot) {
            sensors.clear();
            adapter.notifyDataSetChanged();
            for(DataSnapshot ds : dataSnapshot.getChildren()){
                  String sensorName = ds.getKey().toString();
                  Double humidity =
                        ds.child("humidity").getValue(Double.class);
                  String picture =
                        ds.child("picture").getValue(String.class);
                  sensors.add(new SoilMoistureSensor(humidity, picture,
                                                      sensorName, path));
            }
            adapter.notifyDataSetChanged();
      }
}
locationRef.addListenerForSingleValueEvent(sensorListener);
locationRef.addValueEventListener(sensorListener);
```

Kôd 16. Dohvat podataka sa senzora iz zemlje

```
smSensorViewHolder.tvSMSensorName
      .setText(sensors.get(position).sensorName);
smSensorViewHolder.tvSMSensorHumidity
      .setText(sensors.get(position).humidity + Constants.HUMIDITY_MARK);
StorageReference storageReference = FirebaseStorage.getInstance()
            .getReference().child(sensors.get(position).picture);
Glide.with(context)
      .using(new FirebaseImageLoader())
      .load(storageReference)
      .into(smSensorViewHolder.ivSMSensor);
```

Kôd 17. Prikaz dohvaćenih podataka, sa senzora iz zemlje, na korisničkom sučelju

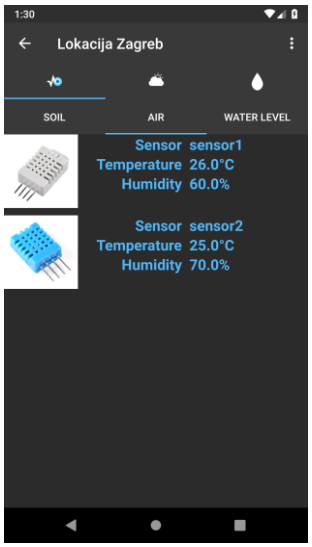
Kako bi se dohvatili podatci iz baze podataka (Kôd 16.), potrebno je dohvatiti `FirebaseReference` objekt te kreirati `ValueEventListener`. Prvo je potrebno očistiti listu senzora metodom `clear` te nakon toga, na adapter pozvati metodu `notifyDataSetChanged()`, kako bi se pripremila lista objekata `SoilMoistureSensors` klase, za prikaz novih podataka. Metoda na adapteru se zove kako bi se adapter obavijestio da se dogodila promjena na podatcima.

Sljedeći korak je očitavanje dohvaćenog rezultata iz baze podataka te popunjavanje liste objekata senzora. Zadnji korak je `notifyDataSetChanged` metoda, kojom se obavještava adapter, da su stigli novi podatci za prikaz.

S ovim koracima kreiran je `ValueEventListener`, te ga je potrebno dodati instanci `FirebaseReference`-a, kroz metode `addListenerForSingleValueEvent` i `addValueEventListener`. Te dvije metode osiguravaju da će se podatci učitati prilikom kreiranja fragmenta te prilikom svake promjene iz baze podataka, kada baza podataka javi da je došlo do promjene traženih podataka.

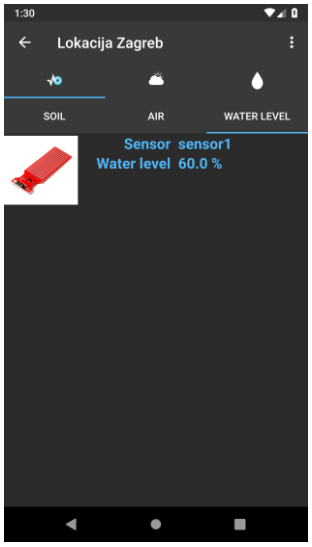
U adapteru, `SMSensorRecyclerViewAdapter`, definirana je XML datoteka u kojoj se nalazi dizajn retka u `RecyclerView` kontroli. U lijevom dijelu retka nalazi se slika senzora, desno od slike nalazi se tablica u dva stupca. Tablica sadrži redove podataka koji su dohvaćeni iz baze podataka. Prikazani podatci su naziv senzora te vlažnost zemlje u postotcima.

Također, u adapteru je definiran način prikaza proslijeđenih podataka (Kôd 17.) iz fragmenta u metodi `onBindViewHolder`. Korištena je i `StorageReference` klasa, koja služi za dohvaćanje slika koje su pohranjene u Firebase Storage-u.



Slika 25. Prikaz senzora za mjerenje temperature i vlažnosti zraka

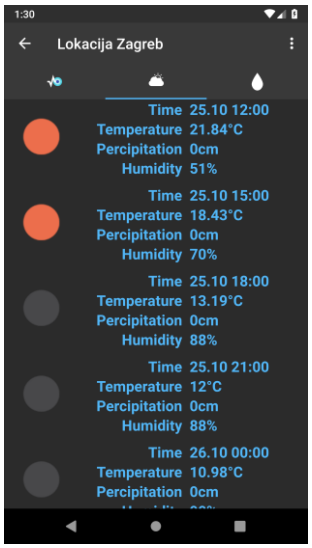
Druga kartica (Slika 25.) predstavlja podatke sa senzora koji mjere temperaturu i vlažnost zraka. Dohvat podataka i prikaz se radi na isti način kao i za prvu karticu, samo u zasebnim klasama, `THSensorRecyclerViewAdapter` i `FragmentTHSensor`. Adapter sadrži zasebnu XML datoteku za dizajn retka unutar `RecyclerView` komponente. Također, s lijeve strane sadrži sliku, te desno od slike, tablicu s podatcima. Prikazani podatci su naziv senzora, temperatura zraka u jedinici stupanj celzijus, te vlažnost zraka u postotcima.



Slika 26. Prikaz senzora za mjerenje količine tekućine

Treća kartica (Slika 26.) prikazuje podatke sa senzora o količini tekućine u zemlji. Dohvat podataka i prikaz za ovu vrstu senzora je identičan kao i za prve dvije, samo u zasebnim klasama, `WLSensorRecyclerViewAdapter` i `FragmentWLSensor`.

Adapter učitava XML datoteku koja je predviđena za prikaz dizajna retka `RecyclerView` komponente. Slika senzora se nalazi s lijeve strane, dok je s desne strane tablica s podatcima sa senzora. Prikazani su naziv senzora te visina tekućine u postotcima.



Slika 27. Prikaz vremenske prognoze

```
String countryReg = getCountryReg(country);
URL url;
HttpURLConnection connection = null;
String urlString = Constants.API_WEATHER_MAIN_URL + "?appid=" +
Constants.API_KEY + "&q=" + city + ","
            + countryReg + "&units=" + Constants.TEMPERATURE_UNITS_VALUE;
try {
      url = new URL(urlString);
      connection = (HttpURLConnection) url.openConnection();
      InputStream in = new
            BufferedInputStream(connection.getInputStream());
      JSONParser jsonParser = new JSONParser();
      JSONObject weatherObject = (JSONObject) jsonParser.parse
                              (new InputStreamReader(in, "UTF-8"));
      JSONArray weatherList = (JSONArray)
                  weatherObject.get(Constants.WEATHER_LISTS);
      for(int i = 0; i < 10; i++){
            //get weather object
            JSONObject weatherObjectItem =
                        (JSONObject) weatherList.get(i);


            //get date-time
```

```
            String dt = convertDateTime(weatherObjectItem
                  .get(Constants.WEATHER_LISTS_DT_TXT).toString());
            //get main
            JSONObject main = (JSONObject) weatherObjectItem
                              .get(Constants.WEATHER_LISTS_MAIN);
            String temp = main
                  .get(Constants.WEATHER_LISTS_MAIN_TEMP).toString();
            String hum = main
                  .get(Constants.WEATHER_LISTS_MAIN_HUM).toString();
            //get weather
            JSONArray weat = (JSONArray) weatherObjectItem
                  .get(Constants.WEATHER_LISTS_WEATHER);
            JSONObject weatherObj = (JSONObject) weat.get(0);
            String icon = Constants.API_IMG_MAIN_URL +
                  weatherObj.get(Constants.WEATHER_LISTS_WEATHER_ICON)
                        .toString() + Constants.API_IMG_EXTENSION;
            //get percipitation
            String rainStr = "0";

      if(weatherObjectItem.containsKey(Constants.WEATHER_LISTS_RAIN)){
                  JSONObject rain = (JSONObject) weatherObjectItem
                              .get(Constants.WEATHER_LISTS_RAIN);
                  rainStr = calculateToCm(rain
                  .get(Constants.WEATHER_LISTS_RAIN_3H).toString());
            }
            data.add(new Weather(dt, temp, rainStr,
                                    hum, icon, city, country));
      }
}
```

Kôd 18. Dohvat podataka o vremenskoj prognozi

Podatci o vremenskoj prognozi su prikazani u drugoj kartici `LocationActivity` aktivnosti (Slika 27.). Karticu predstavlja ikonica oblaka kao vremenske prognoze. Prikazuje se trenutna vremenska prognoza za odabranu lokaciju te vremenska prognoza u iduća 24 sata, u periodima po 3 sata. Za dohvat podataka vremenske prognoze, za odabranu lokaciju, koristi se OpenWeatherMap API.

Prvo je potrebno dohvatiti podatke lokacije iz baze o gradu i državi. Ti podatci su potrebni za API poziv. Nakon toga je potrebno dohvatiti s OpenWeatherMap-a, kôd države prema ISO 3166 standardu popisa kôdova imena država.

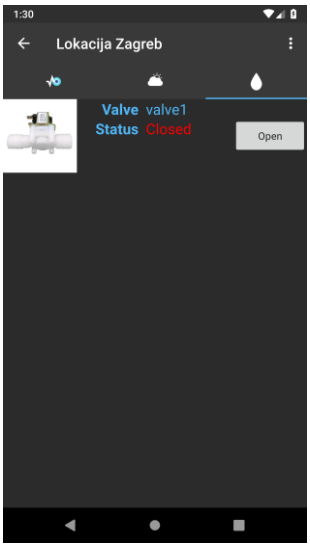
Nakon dohvaćenog kôda države, sastavlja se URL za API poziv. Potrebno je otvoriti konekciju te učitati `InputStream`. S obzirom da je rezultat API poziva JSON, potrebno je napraviti sintaktičku analizu i deserijalizaciju pomoću `JSONParser` klase. Pomoću `JSONArray` i `JSONObject` klasa, dohvaća se temperatura, vlažnost zraka, količina padalina, ikona te vrijeme i datum prognoze. Nakon dohvata svih potrebnih varijabli, iste se koriste za kreiranje instance `Weather` klase, koja se dodaje u listu vremenske prognoze, koja se dalje šalje u adapter.

```
weatherViewHolder.tvWeatherTime.setText(weatherData.get(position).time);
weatherViewHolder.tvWeatherTemperature.setText(weatherData.get(position).
                                temperature + Constants.TEMPERATURE_MARK);
weatherViewHolder.tvWeatherPercipitation.setText(weatherData
            .get(position).percipitation + Constants.PERCIPITATION_MARK);
weatherViewHolder.tvWeatherHumidity.setText(weatherData.get(position)
                                      .humidity + Constants.HUMIDITY_MARK);
Glide.with(context)
            .load(weatherData.get(position).picture)
            .into(weatherViewHolder.ivWeather);
```

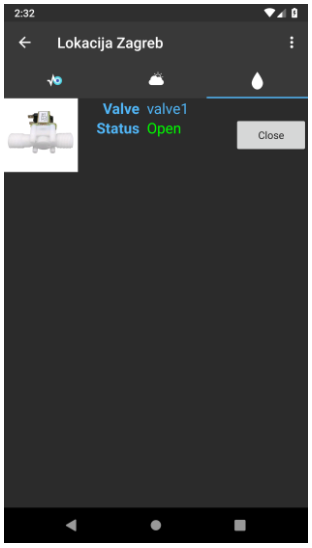
Kôd 19. Adapter za vremensku prognozu

U adapteru (Kôd 19.) `WeatherRecyclerAdapter`, definiran je način prikaza retka pomoću XML datoteke. Podatci koji se prikazuju su slika prognoze na lijevoj strani, te u tablici s desne strane slike podatci o datumu i vremenu prognoze, temperaturi, količini padalina te vlažnosti zraka.

Treća kartica, u aktivnosti `LocationActivity`, prikazuje podatke o statusu ventila te kontrolu za otvaranje i zatvaranje istog. Karticu predstavlja ikonica kapljice vode, koja predstavlja vodu iz ventila.



Slika 28. Prikaz statusa ventila kada je zatvoren



Slika 29. Prikaz statusa ventila kada je otvoren

```
locationRef = FirebaseDatabase.getInstance().getReference(path +
Constants.VALVE_PATH_ADDITION);
ValueEventListener eventListener = new ValueEventListener() {
      @Override
      public void onDataChange(DataSnapshot dataSnapshot) {
            valves.clear();
            valveRecyclerViewAdapter.notifyDataSetChanged();
            for(DataSnapshot ds : dataSnapshot.getChildren()){
                  String valveName = ds.getKey().toString();
                  String status = ds.child("status")
                              .getValue(String.class);
                  String picture = ds.child("picture")
                              .getValue(String.class);
```

```
                  valves.add(new Valve(valveName, status, picture, path +
                                    Constants.VALVE_PATH_ADDITION));
            }
            valveRecyclerViewAdapter.notifyDataSetChanged();
      }
};
locationRef.addListenerForSingleValueEvent(eventListener);
locationRef.addValueEventListener(eventListener);
```

Kôd 20. Dohvat statusa ventila

```
valveViewHolder.tvValveName.setText(valves.get(position).valveName);
valveViewHolder.tvValveStatus.setText(valves.get(position).status);
StorageReference storageReference = FirebaseStorage.getInstance()
            .getReference().child(valves.get(position).picture);
Glide.with(context)
            .using(new FirebaseImageLoader())
            .load(storageReference)
            .into(valveViewHolder.ivValve);
colorText(valveViewHolder);
final String path = valves.get(position).path;
valveViewHolder.btnChangeValveStatus.setOnClickListener(new
                  View.OnClickListener() {
      @Override
      public void onClick(View v) {
            DatabaseReference ref = FirebaseDatabase.getInstance()
                              .getReference(path);
            if(valveViewHolder.btnChangeValveStatus.getText()
                  .toString().equals("Close")){
                  ref.child(valveViewHolder.tvValveName.getText()
                  .toString()).child("status").setValue("Closed");
                  valveViewHolder.btnChangeValveStatus.setText("Open");
            } else if(valveViewHolder.btnChangeValveStatus.getText()
                  .toString().equals("Open")){
                  ref.child(valveViewHolder.tvValveName.getText()
                  .toString()).child("status").setValue("Open");
                  valveViewHolder.btnChangeValveStatus.setText("Close");
            }
      }
});
```

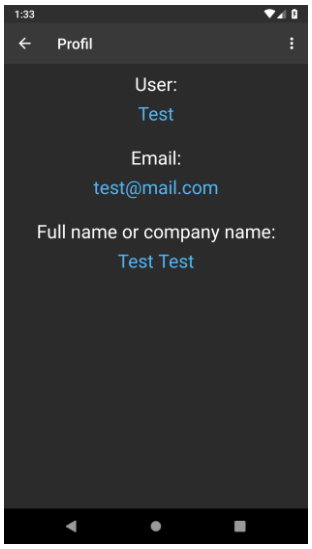
```
private void colorText(ValveViewHolder valveViewHolder) {
      if(valveViewHolder.tvValveStatus.getText()
            .toString().equals("Closed")) {
            valveViewHolder.tvValveStatus.setTextColor(Color.RED);
            valveViewHolder.btnChangeValveStatus.setText("Open");
      }
      else if(valveViewHolder.tvValveStatus.getText()
            .toString().equals("Open")) {
            valveViewHolder.tvValveStatus.setTextColor(Color.GREEN);
            valveViewHolder.btnChangeValveStatus.setText("Close");
      }
}
```

Kôd 21. Adapter za prikaz podataka o ventilu

Kod pokretanja fragmenta, učitavaju se podatci o ventilima (Kôd 20.). Slično kao i kod učitavanja senzora, prvo se čisti lista ventila te se obavještava adapter o promjenama. Iz rezultata se izvlače podatci o statusu i putanji do slike. Učitani podatci o ventilu se spremaju u klasu `Valve` te se dodaju u listu ventila. Zadnji korak je obavještavanje adaptera o promjeni putem `notifyDataSetChanged` metode.

Adapter (Kôd 21.) povlači dizajn retka `RecyclerView` komponente iz XML datoteke. S lijeve strane nalazi se slika ventila, dok se desno od ventila nalazi tablica s podatcima o nazivu ventila i njegovom statusu, te na krajnjoj desnoj strani nalazi se gumb za kontrolu otvaranja i zatvaranja tog ventila. Nakon definiranja prikaza podataka, definira se i prikaz dva stanja ventila, otvoren i zatvoren. Kada je ventil zatvoren (Slika 28.), status će biti prikazan crvenom bojom, dok će tekst na gumbu pokazivati na akciju otvaranja ventila. Drugo stanje je kada je ventil otvoren (Slika 29.), gdje je status prikazan zelenom bojom, a tekst na gumbu pokazuje na akciju zatvaranja ventila.

9.6. Profil



Slika 30. Prikaz profila korisnika

`ProfileActivity` aktivnost (Slika 30.) je aktivnost koja se može prikazati iz bilo koje aktivnosti koja nasljeđuje `BaseActivity`. Sadrži podatke o korisniku koji je ulogiran, a koji su za istog navedeni u bazi podataka. Podatci koji se prikazuju su nadimak korisnika, email korisnika te puno ime korisnika ili kompanije.

10. Poboljšanja

Moguća poboljšanja sustava za nadzor vlažnosti i navodnjavanje tla i njegovih dijelova su:

- Administratorska web aplikacija za CRUD operacije prema Firebase bazi podataka
- Obavijest na Android aplikaciji kada je neki senzor u kritičnoj vrijednosti
- Automatsko zatvaranje ventila za vodu nakon određenog ili zadanog vremena
- Preporuka za otvaranje ventila
- Podrška za korisnike putem Android aplikacije
- Mikrokontroler za svaki senzor

Administratorska web aplikacija, za CRUD operacije prema Firebase bazi podataka, bila bi jako korisna za osobu koja bi se bavila nadzorom sustava i implementacijom novih senzora ili ventila. Olakšavala bi rad s bazom podataka u smislu dodavanja, brisanja i izmjene postojećih senzora ili ventila.

Obavijest na Android aplikaciji, kada je neki senzor u kritičnoj vrijednosti, bio bi koristan za samog korisnika kako bi ga upozorila na, eventualnu prenisku vrijednost pojedinog senzora.

Automatsko zatvaranje ventila za vodu, nakon određenog ili zadanog vremena, omogućavalo bi korisniku napredniju kontrolu samog ventila. Korisnik bi mogao lako zadati vrijeme koje želi i nakon kojega bi se ventil zatvorio ili bi sustav uzeo neku općenitu vrijednost i na taj način automatski zatvorio ventil nakon tog vremena.

Preporuka za otvaranje ventila bi bila nadogradnja postojećeg prikaza kontrole ventila. Na odabir ventila iz liste, korisniku bi se prikazala preporuka da li je potrebno otvoriti ventil ili ne.

Podrška za korisnike putem Android aplikacije poboljšala bi samu podršku korisniku, koja zasad ide samo putem email kanala.

Mikrokontroler za svaki senzor olakšao bi slanje podataka sa senzora prema Firebase bazi podataka, ali i smanjio broj žica i ovisnost o položaju samog senzora.

Zaključak

Sustav za nadzor vlažnosti i navodnjavanje tla uspješno je izrađen i spreman je za probna korištenja. Sustav je testiran nekoliko puta u pravom okruženju i rezultati testiranja su bili zadovoljavajući. S obzirom na to, da bi testiranje trajalo duže da su senzori bili u zemlji, korištene su čaše s vodom kako bi testiranje bilo brže i efikasnije odrađeno.

Kod izrade ovog sustava, pojavilo se dosta problema s Arduino platformom, budući da je za neke situacije pokrivenost dokumentacijom i rješenjima na internetu bila jako loša. Također, debugiranje nije kao kod klasičnih aplikacija i sustava, nego je puno zahtjevnije otkriti problem i da li je rješenje koje se testira za taj problem, zadovoljavajuće.

Sam sustav ima mjesta za poboljšanje i to su poboljšanja koja su navedena u prethodnom poglavlju. Poboljšanja se tiču tehničkih dijelova i podrške.

Sustav je u ovoj fazi spreman za dodatna testiranja, prezentaciju i implementaciju.

Popis kratica

AO	Analog output	Analogni izlaz
API	Application programming interface	Sučelje za programiranje aplikacija
AVD	Android virtual device	Android virtualni uređaj
CRUD	Create/Read/Update/Delete	Kreiraj/Čitaj/Izmijeni/Izbriši
DNS	Domain name system	Sustav domena
DO	Digital output	Digitalni izlaz
GPIO	General purpose input/output	Ulaz/izlaz opće namjene
I2C	Inter integrated circuit	Među integrirani krug
IDE	Integrated development enviroment	Integrirano razvojno okruženje
IoT	Internet of things	Internet stvari
JDK	Java development kit	Java razvojni komplet
JSON	JavaScript object notation	JavaScript notacija objekta
LED	Light-emitting diode	Dioda koja emitira svijetlo
NoSQL	No Standard Query Language	Nije strukturirano upitni jezik
RX	Recieve	Primi
SDK	Software development kit	Komplet za razvoj softvera
SMS	Short message service	Usluge kratkih poruka
SPI	Serial peripheral inteface	Serijsko periferno sučelje
TCP/IP	Transmission control protocol /	Protokol kontrole prijenosa / Internetski

	Internet protocol	protokol
TX	Transmit	Prenesi
UID	User identification	Identifikacija korisnika
USB	Universal serial bus	Univerzalni serijski autobus
XML	Extensible markup language	Proširivi označni jezik

Popis slika

Popis tablica

Popis kôdova

Literatura

[1] Grupa autora, Algebra, Izrada aplikacija za mobilne uređaje, Zagreb 2013

[2] “A comprehensive app development platform”, https://firebase.google.com/, listopad 2019.

[3] “Build better apps”, https://firebase.google.com/products, listopad 2019.

[4] “Add Firebase to your Android project”, https://firebase.google.com/docs/android/setup, listopad 2019.

[5] “Authenticate with Firebase using email link in Android”, https://firebase.google.com/docs/auth/android/email-link-auth, listopad 2019.

[6] “Get started with Firebase Authentication on Android”, https://firebase.google.com/docs/database/android/start, listopad 2019.

[7] “Structure your database”, https://firebase.google.com/docs/database/android/structure-data, listopad 2019.

[8] “Read and write data on Android”, https://firebase.google.com/docs/database/android/read-and-write, listopad 2019.

[9] “Work with lists of data on Android”, https://firebase.google.com/docs/database/android/lists-of-data, listopad 2019.

[10] “Enable offline capabilities on Android”, https://firebase.google.com/docs/database/android/offline-capabilities, listopad 2019.

[11] “Arduino Uno”, https://store.arduino.cc/arduino-uno-rev3, listopad 2019.

[12] “CROduino Basic 2”, https://e-radionica.com/hr/croduino-basic2.html, listopad 2019.

[13] “CROduino Nova”, https://e-radionica.com/hr/croduino-nova.html, listopad 2019.

[14] “Senzor za temperaturu i vlažnost zraka DHT11”, https://e-radionica.com/hr/dht11-senzor-temperature-i-vlage.html, listopad 2019.

[15] “Senzor za temperature I vlažnost zraka DHT22”, https://e-radionica.com/hr/dht22-senzor-temperature-i-vlage.html, listopad 2019.

[16] “Senzor vlažnosti tla”, https://e-radionica.com/hr/senzor-vlaznosti-tla.html, listopad 2019.

[17] “Ventil za vodu”, https://e-radionica.com/hr/elektronski-ventil-za-vodu.html, listopad 2019.

[18] “Eksperimentalna pločica”, https://e-radionica.com/hr/eksperimentalna-plocica-200.html, listopad 2019.

[19] “What is Arduino”, https://www.arduino.cc/en/Guide/Introduction, listopad 2019.

[20] “Arduino products”, https://www.arduino.cc/en/Main/Products, listopad 2019.

[21] “Arduino IDE”, https://www.arduino.cc/en/Main/Software, listopad 2019.

ALGEBRA
VISOKO
UČILIŠTE

NASLOV ZAVRŠNOG RADA

Pristupnik: Dino Mudrovčić, 0082047310

Mentor: mag. Aleksander Radovan