Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Anže Medved

# Implementacija mobilne naprave za GPS sledenje

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Patricio Bulić

Ljubljana 2015

*Besedilo je oblikovano z urejevalnikom besedil LaTeX.*

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Implementirajte vgrajen sistem, ki bi se lahko vgradil v potovalni kovček in nam ponujal lokacijske podatke in podatke o mikroklimi kovčka. Vgrajen sistem naj sloni na mikrokrmilniku STM32F407. Poleg vgrajenega sistema razvijte tudi zaledni strežniški del, ki shranjuje podatke prejete iz vgrajenega sistema, in mobilno aplikacijo za mobilne naprave z operacijskim sistemom Android.

# Izjava o avtorstvu diplomskega dela

Spodaj podpisani Anže Medved sem avtor diplomskega dela z naslovom:

*Implementacija mobilne naprave za GPS sledenje*

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Patricia Bulića,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 6. julija 2015 Podpis avtorja:

*Za pomoč, svetovanje in motivacijo se zahvaljujem svojem mentorju, izr. prof. dr. Patriciu Buliću, in asistentu Roku Češnovarju. Zahvaljujem se tudi svoji družini in dekletu za vso podporo in motivacijo tekom študija.*

Vsem mojim.

# Kazalo

# Seznam uporabljenih kratic

| kratica | angleško | slovensko |
|---|---|---|
| **API** | Application Programming Interface | aplikacijski programski vmesnik |
| **GPS** | Global Positioning System | globalni sistem pozicioniranja |
| **HTTP** | HyperText Transfer Protocol | spletni protokol za prenos podatkov |
| **IOT** | Internet of things | internet stvari |
| **JSON** | JavaScript Object Notation | standard za izmenjavo podatkov |
| **REST** | Representational State Transfer | arhitekturni stil programske opreme |
| **RISC** | Reduced Instruction Set Computer | procesor z omejenim naborom ukazov |
| **SSD** | Solid-state drive | naslednik trdega diska |
| **SQL** | Structured Query Language | strukturirani povpraševalni jezik |
| **TCP** | Transmission Control Protocol | protokol za nadzor prenosa |

# Povzetek

Internet stvari oz. IoT je v zadnjih nekaj letih ena izmed najbolj vročih tem s področja računalništva in elektrotehnike. Jedro vsake IoT naprave je vgrajeni sistem, ki temelji na mikrokrmilniku, senzorjih za zajem podatkov iz okolja in aktuaktorjev za izvajanje fizičnih akcij. V diplomskem delu smo se lotili implementacije vgrajenega sistema, ki bi se lahko vgradil v potovalni kovček in nam ponujal lokacijske podatke in podatke o mikroklimi kovčka. Poleg vgrajenega sistema pa smo pripravili tudi zaledni strežniški del, ki shranjuje podatke prejete iz vgrajenega sistema, in mobilno aplikacijo za mobilne naprave z operacijskim sistemom Android.

**Ključne besede:** GPS,STM32F4,vgrajeni sistem, strežnik, Android.

# Abstract

In recent years, internet of things or IoT became one of the hottest topics in field of computer science and electrical engineering. The core of each IoT device is embedded system, which is based on microcontroller, sensors and actuators. Based on data obtained by sensors, microcontroller generates response, which is executed by actuators. We developed embedded system, which could be used as a tracking device for suitcases. Apart from embedded system, we developed server-side application and mobile application for devices with Android operating system. Server application is used to store the data obtained from tracking device and acts as intermediate point between mobile application and embedded system. Mobile application offers user interface for interaction with tracking device.

**Keywords:** GPS, STM32F4, embedded system, server, Android.

# Poglavje 1

# Uvod

Internet stvari oz. IoT je v zadnjih nekaj letih ena izmed najbolj vročih tem s področja računalništva in elektrotehnike. IoT predstavlja omrežje fizičnih naprav, ki imajo vgrajeno elektroniko in programsko opremo, ki jim omogoča medsebojno komunikacijo in dostop do svetovnega spleta. Glavna ideja IoT je oddaljen dostop in manipulacija naprav, ki izboljša integracijo med fizičnim svetom in računalniškimi sistemi. Jedro vsake IoT naprave je vgrajeni sistem, ki temelji na mikrokrmilniku, senzorjih za zajem podatkov iz okolja in aktuaktorjev za izvajanje fizičnih akcij.

V diplomskem delu smo se lotili implementacije vgrajenega sistema, ki bi se lahko vgradil v potovalni kovček in nam ponujal lokacijske podatke in podatke o mikroklimi kovčka. Ena izmed možnosti uporabe so potovalni kovčki za inštrumente, ki morajo biti na točno določeni temperaturi in vlažnosti. Poleg vgrajenega sistema pa smo pripravili tudi zaledni strežniški del, ki shranjuje podatke prejete iz vgrajenega sistema, in mobilno aplikacijo za mobilne naprave z operacijskim sistemom Android.

V prvem delu diplomskega dela je opisan izbrani mikrokrmilnik STM32F4-Discovery, modul za pridobivanje GPS podatkov LEA-6S, modul za komunikacijo s strežnikom SIM800H, senzor za temperaturo in vlažnost DHT11, ter napajalna postaja.

V drugem delu je opisan zaledni strežniški sistem, ki je sestavljen iz TCP

strežnika za pridobivanje podatkov iz naprave, REST storitev za mobilno aplikacijo in nerelacijske podatkovne baze MongoDB, ki hrani podatke o meritvah.

V zadnjem delu je opisana mobilna aplikacija, ki omogoča prijavo v sistem in prikaz podatkov za izbrane naprave. Izdelana je za naprave z operacijskim sistemom Android verzije 5.0 ali višje.

# Poglavje 2

# Sledilna naprava

Glava enota razvitega sistema je sledilna naprava, ki smo jo razvili z uporabo razvojne plošče STM32F4-Discovery, GPS modula za pridobivanje lokacijskih podatkov u-blox LEA-6S, GPRS modula za uporabo mobilnega omrežja SIMCom SIM800H in senzorja za zajem podatkov o temperaturi in vlažnosti DHT11. Opis posameznih komponent in implementacija naprave je opisana v sledečih razdelkih.

## 2.1 STM32F4-Discovery

Kot osnovo smo uporabili razvojno ploščo STM32F4-Discovery, ki vsebuje mikrokrmilnik STM32F407. Krmilnik temelji na visokoperformančnem jedru ARM®Cortex®-M4F, ki deluje na frekvencah do 168 MHz in uporablja 32-bitne RISC ukaze. Jedro vsebuje enoto za operacije v plavajoči vejici (FPU), ki podpira vse ARM ukaze v enojni natančnosti in vse podatkovne tipe. Implementirani so tudi vsi ukazi za digitalno procesiranje signalov in enota za zaščito pomnilnika (MPU), ki poveča varnost aplikacij.

Pri uporabi imamo na voljo 1MB Flash pomnilnika, 192 KB SRAM pomnilnika in 4 KB pomožnega SRAM pomnilnika. Za priključitev zunanjih naprav krmilnik omogoča širok nabor vhodno-izhodnih naprav, ki jih povezuje s tremi AHB vodili, dvema APB vodiloma in AHB matriko za povezo-

vanje več vodil.

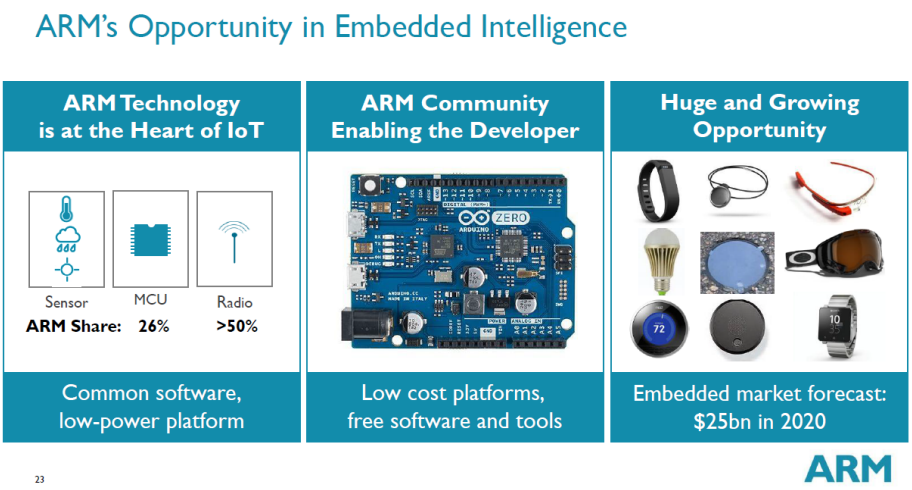
Mikrokrmilnik omogoča tri 12-bitne analogno-digitalne pretvornike (ADC), dva digitalno-analogna pretvornika (DAC), dvanajst splošno namenskih 16-bitnih časovnikov, dva splošno namenska 32-bitna časovnika in generator naključnih števil. Dva 16-bitna časovnika omogočata pulzno širinsko modulacijo (PWM) za krmiljenje motorjev.

Pri komunikaciji lahko uporabljamo standardne in napredne komunikacijske protokole. Mikrokrmilnik omogoča sledeče:

- *I²C* (angl. Inter-Integrated Circuit) - serijski half-duplex protokol, ki se uporablja za priključitev počasnejših naprav
- *SPI* (angl. Serial Peripheral Interface) - sinhroni serijski full-duplex protokol, ki se pogosto uporablja za komunikacijo s senzorji, SD karticami in LCD zasloni
- *I²S* (angl. Integrated Interchip Sound) - serijski vmesnik za povezovanje avdio naprav
- vmesnik *U(S)ART* (angl. Universal asynchronous reciever transmitter) - univerzalni asinhronski serijski vmesnik
- vmesnik *USB On-The-Go* omogoča USB napravam, da izmenjujejo vlogi gostitelja in odjemalca
- vmesnik *CAN* (angl. Controller Area Network) omogoča mikrokrmilnikom, da komunicirajo med seboj brez uporabe gostiteljskega računalnika
- vmesnik *SDIO* (angl. Secure Digital Input Output)
- *Ethernet* vmesnik

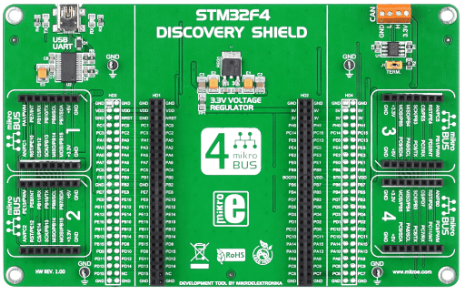
### 2.1.1 ARM

Podjetje ARM Holdings se ukvarja z razvojem računalniških procesorjev, ki temeljijo na arhitekturi RISC. Procesorji, ki temeljijo na tej arhitekturi,




Slika 2.1: Napovedi za prihodnost ARM. [2]

uporabljajo omejen nabor ukazov, zaradi česar so hitrejši in cenejši od tistih, ki temeljijo na CISC arhitekturi. Za ukaze velja, da se vsi izvedejo v enem urinem ciklu in so enako dolgi. Podjetje razvija le teoretične modele procesorjev, ki jih nato prodaja kot intelektualno lastnino, samih čipov pa ne izdeluje. Kljub temu, da ne izdelujejo čipov, je njihova arhitektura ena izmed najbolj široko uporabljenih, glede na število proizvedenih čipov. Med največje proizvajalce čipov, ki uporabljajo ARM arhitekturo, spadajo Apple, Atmel, Broadcom, Qualcomm, Samsung Electronics in drugi. ARM procesorji so postali popularni zaradi njihove nizke porabe energije, zato jih pogosto najdemo mobilnih naprav in vgrajenih sistemih. Od ustanovitve podjetja leta 1985 do danes je bilo proizvedenih prek 65 milijard čipov, od tega je bilo v prvem četrtletju 2015 proizvedenih 3.8 milijarde čipov [1]. To število hitro narašča zaradi vedno večjega povpraševanja po hitrih in energijsko učinkovitih procesorjih. V prihodnosti ARM načrtuje prehod na vgrajeno inteligenco, kar so tudi oznanili v letnem načrtu za prihodnost. Na sliki 2.1 je prikazana slika iz njihove predstavitve letnega načrta [2].




Slika 2.2: Razširitvena plošča za STM32F4-Discovery.

### 2.1.2 Priklop modulov

Za priklop modulov smo uporabili razširitveno ploščo za STM32F4-Discovery proizvajalca MikroElektronika, ki je prikazana na sliki 2.2. Plošča omogoča priklop STM32F4 v osrednji del in štirih modulov v stranske reže. Reže so povezane s STM32F4 tako da omogočajo komunikacijska protokola SPI in USART, in sicer reža 1 in 3 omogočata USART3, reži 2 in 4 pa USART6. Pri izdelavi naprave smo uporabili režo 1 za priklop GPS modula in režo 3 za GPRS modul, tako da uporabljata ločena USART kanala. Senzor za temperaturo in vlago smo vezali neposredno na STM32F4.

## 2.2 Podatki o lokaciji

Globalni sistem pozicioniranja (GPS) je satelitski sistem za pridobivanje podatkov o lokaciji in času. Sistem deluje kjerkoli na Zemlji, v vseh vremenskih pogojih, kadar imamo kontakt s štirimi ali več sateliti, od trenutno 32 aktivnih satelitov. Razvile so ga ZDA leta 1978 s prvotnim namenom uporabe v vojaške namene, danes pa je na voljo tudi v civilne in komercialne namene. GPS pa ni edini sistem za pridobivanje podatkov o lokaciji. Trenutno so v razvoju oz. so že aktivni sistemi

- *GLONASS* (rus. Globalnaya navigatsionnaya sputnikovaya sistema),

- *Galileo* je globalni sistem, ki ga razvija Evropska unija in bo predvidoma popolnoma funkcionalen leta 2019,
- *Beidou* je kitajski sistem, ki je trenutno omejen na območje Azije,
- *COMPASS* je globalni sistem, ki ga razvija Kitajska in bo predvidoma razvit do leta 2020,
- *IRNSS* (angl. Indian Regional Navigation Satellite System) je indijski navigacijski sistem, ki naj bi prišel v uporabo leta 2015,
- *QZSS* (angl. Quasi-Zenith Satellite System) je japonski navigacijski sistem, ki pokriva območje Azije in Oceanije [3].

Glavni koncept delovanja GPS sistema je čas. Vsak izmed satelitov nosi natančno in stabilno atomsko uro, ki jo sinhronizira z ostalimi sateliti in zemeljskimi urami. Vse nepravilnosti v času se popravljajo na dnevni ravni. Sateliti neprestano oddajajo signal ki vsebuje psevdonaključno vrednost, ki je znana sprejemniku, čas oddaje signala in lokacija satelita ob oddaji signala. Sprejemnik sprejme signal in na podlagi svoje ure izračuna čas potovanja signala TOF(angl. time of flight). Na podlagi štirih vrednosti TOF lahko sprejemnik nato izračuna svojo tridimenzionalno pozicijo in časovno odstopanje. Rezultat se nato prevede iz koordinatnega sistema Zemlje, ki ima izhodišče v središču Zemlje, v zemljepisno dolžino, zemljepisno širino in nadmorsko višino.

### 2.2.1 Navigacijske enačbe

Za izračun lokacije se uporabijo navigacijske enačbe, s katerimi lahko vse štiri vrednosti izračunamo hkrati. Za izračun potrebujemo $x$, $y$ in $z$ komponente lokacije satelita in čas oddaje signala, kar lahko zapišemo kot $[x_i, y_i, z_i, s_i]$. Indeks $i$ predstavlja zaporedno številko satelita in ima vrednost *1,2,...,n*, kjer mora veljati da $n \geq 4$. Čas sprejema sporočila označimo s $\tilde{t}$, pravi čas sprejema pa kot $t = \tilde{t}$ - $b$, kjer je $b$ časovno odstopanje prejemnikove

ure glede na uro satelita. Velja da je čas potovanja signala $\tilde{t}$ - $b$ - $s_i$ in prepotovana razdalja $(\tilde{t}$ - $b$ - $s_i)c$, saj predpostavljamo, da signal potuje s svetlobno hitrostjo $c$. Za n satelitov moramo rešiti enačbe 2.1

$$(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 = ([\tilde{t} - b - s_i]c)^2, i = 1, 2, ..., n \quad (2.1)$$

oziroma pri uporabi psevdorazdalje $p_i = (\tilde{t} - s_i)c$ enačbe 2.2

$$\sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} + bc = p_i, i = 1, 2, ..., n \quad (2.2)$$

Kadar imamo na voljo več kot štiri satelite lahko pri izračunu uporabimo najboljše štiri vrednosti ali pa jih uporabimo več. Kadar uporabimo več kot štiri vrednosti dobimo predefiniran sistem enačb, ki nima enolične rešitve. V tem primeru lahko sistem rešimo z uporabo metode najmanjših kvadratov ali metode uteženih najmanjših kvadratov.

$$(\hat{x}, \hat{y}, \hat{z}, \hat{b}) = \underset{(x,y,z,b)}{\operatorname{argmax}} \sum_i (\sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} + bc - p_i)^2 \quad (2.3)$$

### 2.2.2 Modul GPS

V sledilni napravi smo uporabili modul u-blox LEA-6S[4] z razširitveno ploščico, ki je prikazana na sliki 2.3. Modul uporablja globalni sistem pozicioniranja za pridobitev podatkov o lokaciji. Prve podatke pridobimo v 30 sekundah po vzpostavitvi modula oz. v nekaj sekundah če je modul pred tem že deloval. V idealnih pogojih dosegajo podatki natančnost do 2 metrov odklona od dejanske lokacije [4]. Poleg klasičnega načina delovanja pa modul podpira tudi degradiran način delovanja in t.i. *Dead Reckoning* ali slepi način delovanja.

V degradiran način delovanja preide avtomatično, kadar ima na voljo le tri vidne satelite. Navigacijski algoritem v tem primeru uporabi konstantno vrednost za nadmorsko višino, kar imenujemo dvodimenzionalno lociranje. V primeru, da je pred izgubo vidnih satelitov modul imel tridimenzionalno lociranje, se nadmorska višina ohrani na zadnji znani vrednosti.


Slika 2.3: GPS modul u-blox LEA-6S.

Če število vidnih satelitov pade na tri ali manj, torej kadar lociranje ni več mogoče, modul preide v slepi način delovanja. V tem načinu delovanja vrača lokacijske podatke na podlagi predhodnih informaciji o smeri in hitrosti premikanja, dokler ne doseže časovne omejitve oz. pridobi vidne satelite s pomočjo katerih se lahko zopet locira.

Povezavo z modulom lahko vzpostavimo preko vmesnika USB verzije 2.0, UART ali DDC. Komunikacijo z modulom izvajamo z uporabo protokola NMEA (podrobneje opisan v 2.2.3), UBX ali RTCM. Protokol UBX je lastniški protokol podjetja u-blox in se uporablja za komunikacijo med njihovimi GPS moduli in računalnikom preko asinhronega RS232 vmesnika. RTCM (angl. Radio Technical Commission for Maritime Service) je enosmerni protokol, ki GPS sprejemniku zagotavlja realno časovne diferencialne korekcijske podatke, ki izboljšajo točnost sprejemnika.

### 2.2.3 Protokol NMEA

Večina komunikacije z GPS modulom je izvedena z uporabo NMEA protokola. Format sporočil, ki jih protokol uporablja, je sestavljen iz

- *začetnega znaka*, ki ima vedno vrednost $ ,
- *naslovnega polja*, kjer prva dva znaka označujeta pošiljatelja (GP predstavlja prejemnika), preostali trije znaki pa format vsebine,
- *vrednostnega polja*, ki je spremenljive dolžine, podatki pa so ločeni z vejico,

- *kontrolne vsote*, ki se začne z znakom * in je izračunana kot vsota vseh znakov med $ in * po modulu 2,
- *zaključne sekvence*, ki vsebuje povratnico in znak za novo vrstico.

Primer veljavnega sporočila je prikazan v enačbi 2.4. Po standardu NMEA se podatki pošiljajo v formatu celega števila stopinj in minut, nato pa decimalni deli minut. Primer za zemljepisno širino iz 2.4 se prevede iz *4717.12624* v 47 stopinj in 17.12624 minut. V primeru, da GPS podatkov o lokaciji nima potem vrne le vejice v vrednostnem polju [5].

$$\$GPGLL, 4717.12624, N, 00833.91297, E, 124923.00, A, A * 6E \tag{2.4}$$

Izbrani modul podpira sledeče standardne NMEA ukaze

- *GGA* vrača podatke o trenutnem času, zemljepisni širini, zemljepisni dolžini, statusu lociranja, številu uporabljenih satelitov, nadmorski višini in starosti diferencialnih popravkov,
- *RMC* vrača enake podatke kot GGA, poleg tega pa še podatek o smeri premikanja, hitrosti premikanja in datum,
- *GSA* vrača pozicijsko, vertikalno, horizontalno natančnost in zaporedne številke uporabljenih satelitov,
- *GSV* vrača zaporedne številke vidnih satelitov.

### 2.2.4 Implementacija

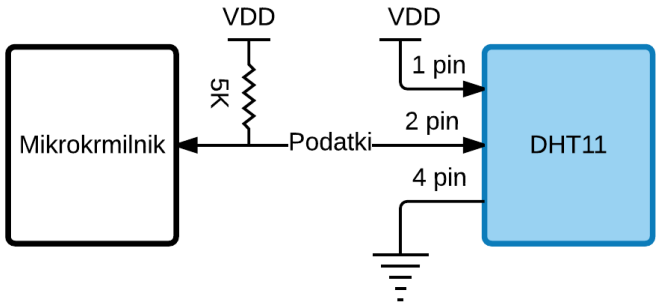
Pri implementaciji smo si pomagali s knjižnico, ki je dostopna na [6] pod licenco GNU General Public License, različice 3. Knjižnica je bila spisana za uporabo z modulom u-blox NEO-6M, ki podpira enake standarde kot LEA-6S, tako da je bilo pri inicializaciji potrebno spremeniti le nekaj vrednosti. Modul smo na začetku inicializirali tako, da smo z njim komunicirali

z uporabo USART3 s hitrostjo prenosa 9600, 8 podatkovnimi biti, 1 stop bitom, paritetnih bitov pa nismo uporabili. Osrednji del knjižnice predstavlja struktura *TM_GPS_Data_t*, ki vsebuje glavne lokacijske podatke (zemljepisna dolžina, zemljepisna širina, nadmorska višina, število vidnih satelitov, ...). Po inicializaciji je knjižnica sama poskrbela za pošiljanje GGA, RMC, GSA, GSV ukazov po protokolu NMEA.

V poslu operacijskega sistema FreeRTOS (podroben opis se nahaja v razdelku 2.5) , ki je bil namenjen GPS modulu pridobivali lokacijske podatke kot je prikazano v spodnjem izseku programske kode.

```
1  TM_GPS_Data_t GPS_Data;
2
3  // Posodobimo GPS podatke
4  result = TM_GPS_Update(&GPS_Data);
5
6  // Ce imamo neprebrane podatke
7  if (result == TM_GPS_Result_NewData) {
8
9     // Ali so podatki veljavni?
10    if (GPS_Data.Validity) {
11       // Shranimo podatke za posiljanje
12       TM_GPS_ConvertFloat(GPS_Data.Latitude, &GPS_Float, 4);
13       readDataGps[0]=GPS_Float.Integer;
14       readDataGps[1]=GPS_Float.Decimal;
15       TM_GPS_ConvertFloat(GPS_Data.Longitude, &GPS_Float, 4);
16       readDataGps[2]=GPS_Float.Integer;
17       readDataGps[3]=GPS_Float.Decimal;
18    }
19 }
```

Podatke o zemljepisni dolžini smo začasno shranili v obliki dveh celih števil, kjer prvo število predstavlja celi del, drugo število pa prva štiri decimalna mesta prvotnega števila. Enako smo storili tudi za zemljepisno širino. Vsa štiri števila smo nato prenesli v vrsto, iz katere jih je pridobil posel namenjen pošiljanju podatkov na strežnik.




Slika 2.4: Vezava senzorja DHT11.

## 2.3 Podatki o temperaturi in vlagi

Poleg lokacijskih podatkov smo iz sledilne naprave pridobivali tudi podatke o temperaturi in vlagi s pomočjo senzorja DHT11. Senzor ima merilno območje za vlažnost od 20 do 90% in za temperaturo od 0 do 50 stopinj Celzija. Natančnost meritve vlažnosti odstopa do 5% in do 2 stopinji za temperaturo. Za komunikacijo s senzorjem se uporablja t.i. *single-wire* serijski protokol. Opis uporabe in implementacije se nahaja v sledečih podpoglavjih.

### 2.3.1 Vezava senzorja

Senzor DHT11[7] ima štiri priključke, od tega so aktivni le prvi, drugi in četrti, medtem ko tretji priključek ni povezan. Na prvi priključek povežemo napajalno napetost, ki mora biti na intervalu 3-5.5V. Po priključitvi napajanja moramo počakati 1 sekundo, pred začetkom pošiljanja podatkov, sicer bi senzor lahko prešel v nestabilno stanje. Na drugi priključek povežemo podatkovno linijo mikrokrmilnika. Kadar je povezovalni kabel za podatkovno linijo krajši od dvajsetih metrov, se priporoča 5K *pull-up* upor. Na četrti priključek povežemo ozemljitev. Na sliki 2.4 je prikazana vezava senzorja.

### 2.3.2 Komunikacijski protokol

Za komunikacijo in sinhronizacijo mikrokrmilnika s senzorjem se uporablja samo en priključek - *single-wire* dvosmerna serijska komunikacija. Sporočilo je sestavljeno iz decimalnega in integralnega dela, ter je dolgo 40 bitov. Prvih 16 bitov je sestavljeno iz decimalnega in integralnega podatka o vlažnosti, drugih 16 bitov je sestavljeno iz decimalnega in integralnega podatka o temperaturi, zadnjih 8 bitov pa sestavlja kontrolna vsota. Kontrolna vsota mora biti enaka vsoti predhodnih 32 bitov (seštejemo štirikrat po 8 bitov).

Na začetku izmenjave podatkov mora mikrokrmilnik poslati senzorju začetni signal, kar povzroči, da senzor preide v stanje delovanja in čaka, da mikrokrmilnik zaključi s pošiljanjem začetnega signala. Signal je aktiven v nizkem stanju, mikrokrmilnik pa ga mora držati aktivnega vsaj 18ms, da ga DHT11 zazna. Mikrokrmilnik nato čaka 20-40us na odgovor senzorja. Senzor se odzove z aktivnim signalom, ki ga drži 80us, nato pa čaka 80us v neaktivnem stanju pred pošiljanjem podatkov. Podatki se pošljejo v obliki 40-bitnega signala, ki vsebuje podatke o relativni vlažnosti in temperaturi. Vrednost posameznega bita je odvisna od časa v katerem je signal v aktivnem stanju. Če je signal v aktivnem stanju 26-28us, potem je vrednost bita 0, v primeru, da je signal v aktivnem stanju 70us je vrednost bita 1. Senzor po vsakem poslanem bitu postavi signal v neaktivno stanje. Po poslanih podatkih DHT11 preide v stanje čakanja, v katerem porabi manj energije kot v stanju delovanja, in čaka na ponoven začetni signal mikrokrmilnika.

### 2.3.3 Implementacija

Za inicializacijo in branje podatkov smo pripravili funkcijo *readDht11*, ki implementira predhodno opisani protokol. Funkcija sprejme kazalec na seznam v katerega shrani prebrane vrednosti iz senzorja. Za zagotavljanje pravilnih časovnih intervalov smo uporabili časovnik, ki smo mu uro nastavili na 10kHz, kar pomeni, da mora za 1 sekundo preteči 10000 urinih ciklov. Z uporabo časovnika smo nato napisali funkcijo za zakasnitev za poljubno število

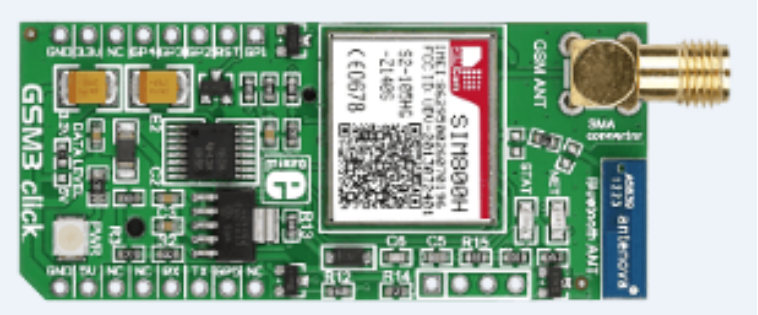
us.

```
void delayUs(int us){
    int count=0;
    TIM_Cmd(TIM3,DISABLE);
    TIM_SetCounter(TIM3, 0);
    TIM_Cmd(TIM3,ENABLE);
    TIM_ClearFlag(TIM3,TIM_FLAG_CC2);
    while(count<us){
      if(TIM_GetFlagStatus(TIM3,TIM_FLAG_CC2) == 1){
          TIM_ClearFlag(TIM3,TIM_FLAG_CC2);
          count++;
      }
    }
}
```

## 2.4 Pošiljanje podatkov na strežnik

Za pošiljanje podatkov iz naprave na strežnik smo uporabili GPRS modul SIM800H na razširitveni ploščici, ki je prikazana na sliki 2.5[8]. GPRS (angl. General packet radio service) je paketno orientirana mobilna podatkovna storitev, ki temelji na drugi generaciji (2G) globalnega sistema za mobilno komunikacijo (GSM), zato jo ponavadi označujejo kot 2.5G. Storitev spada med *best-effort* storitve, kar pomeni, da je propustnost in latenca uporabe storitve odvisna od zasedenosti omrežja. Z uporabo TDMA (angl. time division multiple access) kanalov omogoča podatkovne prenose pri hitrosti 56-114kbit/s. GPRS podpira sledeče protokole

- *Internet protocol* (IP),
- *Point-to-point protocol* (PPP), ki se ponavadi uporabi kadar mobilna naprava igra vlogo modema,
- *X.25*, ki se je uporabljal za brezžična plačila.




Slika 2.5: GPRS modul SIM800H.

V primeru uporabe TCP/IP ima lahko vsak telefon enega ali več IP naslov, kjer GPRS poskrbi za pravilno posredovanje paketov, TCP protokol pa rešuje težave izgubljenih paketov.

Za povezavo v omrežje mora uporabnik poznati ime dostopne točke (APN), uporabniško ime in geslo, ki mu ga zagotovi omrežni operater.

### 2.4.1 GPRS modul

SIM800H[10] je štiripasovni GSM/GPRS modul, ki deluje na frekvencah GSM 850Mhz, EGSM 900MHz, DSC 1800MHz in PCS 1900MHz. Podpira več načinov delovanja, in sicer stanje spanja, stanje pripravljenosti, stanje pogovora in stanje podatkovnega prenosa. Z modulom lahko komuniciramo preko UART ali USB protokola.

V našem projektu smo uporabili UART vmesnik, preko katerega smo pošiljali AT ukaze, s katerimi smo pridobivali in pošiljali podatke modulu.

### 2.4.2 Hayesov ukazni nabor

Modul za komunikacijo uporablja posebne t.i. AT ukaze, ki temeljijo na Hayesovem naboru ukazov[9]. Nabor ukazov je razvil Dennis Hayes za Hayes Smartmodem 3000 leta 1981. Ukazi so sestavljeni in krajših tekstovnih nizov, ki v kombinaciji sestavljajo popoln nabor za izvajanje klicev, prekinjanje klicev in nastavljanje parametrov povezave. Osnovni nabor vsebuje le

ukaze, ki so jih podpirali prvi 300 bit/s modemi, zato se je pojavilo veliko novih standardov. Novi standardi razširjajo nabor ukazov z novimi ukazi, ki podpirajo nove funkcionalnosti modemov.

Potreba po ukaznem naboru se je pokazala ob koncu 70ih let, ko se je močno povečalo število nizko cenovnih modemov in s tem število uporabnikov. Predhodna rešitev ročnega prestavljanja med načini delovanja modema, zaradi velikega števila uporabnikov, ni bila več ustrezna. Hayesova rešitev omogoča, da se modem samostojno preklaplja med podatkovnim in ukaznim stanjem. V podatkovnem načinu delovanja modem obravnava vse prejete pakete kot podatke, ki jih nato pošlje oddaljenemu modemu. V ukaznem načinu delovanja obravnava vse prejete pakete kot ukaze, ki jih izvrši lokalno. Za preklop v ukazni način delovanja je potrebno poslati zaporedje znakov +++ in nato počakati eno sekundo. Mnogi proizvajalci so se želeli izogniti licenciranju Hayesovega patenta tako, da po poslanih znakih za preklop v ukazni način delovanja niso počakali eno sekundo oz. niso zavrgli podatkov, ki so prišli v prvi sekundi. To je vodilo v veliko varnostno luknjo znano kot +++ATH0 s katero se je izklopilo oddaljeni modem, saj je modem sprejel ukaz ATH0 in ga po preklopu v ukazni način tudi izvedel.

### 2.4.3 AT ukazi

Za AT ukaze, ki smo jih uporabljali pri našem modulu je značilno, da se začnejo z AT in končajo vračalko (CR). Ukazi imajo lahko sledečo sintakso

- *AT+x=?*, ki se uporablja za testne ukaze,
- *AT+x?*, ki se uporablja za bralne ukaze,
- *AT+x=...*, ki se uporablja za pisalne ukaze (npr. nastavljanje parametrov),
- *AT+x*, ki se uporabljajo za eksekucijske ukaze.

Ukazi za testiranje delovanja modula so prikazani v tabeli 2.1. Prvi ukaz AT se pošilja toliko časa, dokler ne dobimo odgovora OK, saj se uporablja

| ukaz | opis delovanja |
|---|---|
| AT | prvi ukaz, ki se pošlje ob vzpostavitvi modula za preverjanje odziva in sinhronizacije |
| AT+CPIN? | preverjanje vnosa PIN kode |
| AT+CSQ | preverjanje signala |
| AT+COPS? | pridobivanje stanja povezave in mobilnega operaterja |

Tabela 2.1: Ukazi za preverjanje delovanja.

| ukaz | opis delovanja |
|---|---|
| AT+CGATT=1 | status GPRS povezave |
| AT+CSTT=‘simobil’ | vnos APN |
| AT+CIICR | vzpostavitev GPRS |
| AT+CIFSR | pridobitev lokalnega IP naslova |
| AT+CIPSTART=’TCP’,’IP’,’port’ | vzpostavitev povezave |
| AT+CIPSEND=X | pošiljanje X bajtov podatkov |
| AT+CIPCLOSE | zapiranje povezave |

Tabela 2.2: Ukazi za vzpostavljanje TCP povezave.

za sinhronizacijo hitrosti prenosa podatkov. Če hitrost prenosa nastavimo z ukazom AT+IR=(hitrost) in shranimo spremembe z ukazom AT&W, potem sinhronizacija ni potrebna. V primeru, da na ukat AT+CPIN? dobimo odgovor ERROR pomeni, da je SIM kartica zaklenjena in zahteva vnos PIN kode, ki jo vnesemo z AT+CPIN=XXXX, kjer XXXX nadomestimo z našo PIN kodo.

Pri vzpostavitvi TCP/IP povezave smo uporabili ukaze prikazane v tabeli 2.2. V primeru pravilnega delovanja dobimo pri prvih treh ukazih odgovor OK, pri ukazu AT+CIFSR pa lokalni IP naslov. Ukaz AT+CIPSTART sprejme tri parametre, in sicer vrsto povezave, IP naslov oddaljene naprave in vrata na katerih aplikacija na oddaljeni napravi pričakuje povezavo. Po

vzpostavitvi povezave pošljemo podatke z ukazom AT+CIPSEND, ki sprejme število poslanih podatkov v obliki parametra. Po izvedbi ukaza se modem prestavi v podatkovni način delovanja in se po vnosu podatkov vrne v ukazni način [9].

Ukaze smo modulu pošiljali z uporabo UART protokola.

### 2.4.4 UART protokol

UART (angl. Universal asynchronous receiver/transmitter) protokol se uporablja za serijsko komunikacijo med periferno napravo in računalnikom preko serijskega vmesnika. Oddajnik shrani posamezen bajt podatkov pred pošiljanjem v pomikalni register in nato pošlje bite v sekvenčnem redu po svojem oddajnem kanalu (TX). Sprejemnik sprejme bite na svojem vhodnem kanalu (RX) in jih shranjuje v pomikalni register, dokler ne prispejo vsi biti. Protokol oznani, da so bili podatki poslani oz. prejeti, s spremembo stanja statusnega registra TXE (angl. transmit register empty) oz. RXF (angl. recieve register full). Protokol lahko deluje v

- *simplex* načinu, kjer prva naprava pošilja podatke, druga pa jih sprejema,
- *half-duplex* načinu, kjer se napravi pri pošiljanju podatkov izmenjujeta,
- *full-duplex* načinu, kjer obe napravi hkrati pošiljata in sprejemata podatke.

Dolžina poslanih podatkov v posameznem prenosu je lahko 11 ali 12 bitov, pri čimer prvi bit označuje začetek pošiljanja (start bit), nato sledi 8 bitov podatkov od skrajno desnega LSB (angl. least significant bit) proti skrajno levemu MSB (angl. most significant bit). Po podatkih sledi bit za preverjanje paritete in eden ali dva bita, ki označujeta konec prenosa. Start bit je aktiven v nizkem stanju in se uporablja za preverjanje hitrosti pošiljanja, saj želimo signal vzorčiti na sredini intervala, kjer je najbolj stabilen.

### 2.4.5 Implementacija

Modul smo vstavili v razširitveno ploščico na mesto označeno s 3, ki uporablja USART6 s priključki PC6 in PC7. Ob inicializaciji smo USART6 nastavili na hitrost 9600 b/s in uporabili 8 podatkovnih ter 1 stop bit. Za pošiljanje AT ukazov smo napisali funkcijo, ki sprejme kazalec na vrsto znakov in jih nato pošlje v sekvenci, ki jo zaključi z vračalko in znakom za novo vrstico.

```
1  void sendATCommand(const char *s){
2    // posiljaj znake
3    while(*s){
4      USART_SendData(USART6, *s++);
5      while(!USART_GetFlagStatus(USART6, USART_FLAG_TC));
6    }
7
8    // dodaj CR
9    USART_SendData(USART6, 0x0D);
10   while(!USART_GetFlagStatus(USART6, USART_FLAG_TC));
11
12   // dodaj LF
13   USART_SendData(USART6, 0x0A);
14   while(!USART_GetFlagStatus(USART6, USART_FLAG_TC));
15 }
```

Za pridobivanje podatkov iz modula smo uporabili vektorski prekinitveni krmilnik NVIC. Prekinitev se je sprožila takrat, ko so bili na voljo podatki za sprejem na kanalu USART6. Podatke smo shranjevali v krožni pomnilnik dolžine 20, s katerim smo preverjali uspešnost izvajanja ukazov.

```
1  void USART6_IRQHandler(void){
2    if(USART_GetITStatus(USART6,USART_IT_RXNE)){
3        if(countData==20){
4          countData=0;
5        }
6        dataU=USART_ReceiveData(USART6);
7        txtGPRS[countData]=dataU;
8        countData++;
9        USART_ClearITPendingBit(USART6,USART_IT_RXNE);
```

```
10   }
11 }
```

Za aktivacijo modula je potrebno postaviti napajalni vtič za eno sekundo v nizko stanje in nato za 2 sekundi v visoko stanje, ter počakati, da modul postavi statusni vtič v nizko stanje. Po izvedeni proceduri je modul pripravljen za uporabo. Po aktivaciji smo počakali 10 sekund, da se je modul povezal v omrežje.

```
1 GPIO_ResetBits(GPIOB, GPIO_Pin_1);
2 vTaskDelay(1000 / portTICK_RATE_MS);
3 GPIO_SetBits(GPIOB, GPIO_Pin_1);
4 vTaskDelay(2000 / portTICK_RATE_MS);
5 while(GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_5)==0);
6 vTaskDelay(10000 / portTICK_RATE_MS);
```

Za pošiljanje podatkov smo uporabili ukaze prikazane v tabeli 2.2, ki smo jih poslali z uporabo funkcije *sendATCommand(ukaz)*.

## 2.5 Operacijski sistem

Sledilna naprava mora po inicializaciji komponent skrbeti za tri različna opravila, in sicer pridobivanje podatkov podatkov o lokaciji, temperaturni, vlagi in pošiljanje podatkov na strežnik. Za upravljanje z opravili smo uporabili enostaven operacijski sistem za vgrajene sisteme FreeRTOS.

### 2.5.1 FreeRTOS

FreeRTOS je realno-časovni operacijski sistem namenjen vgrajenim sistemom, ki ga podpira že 35 mikrokrmilnikov. Sistem je majhen in enostaven, saj ga sestavljajo le štiri datoteke napisane v programskem jeziku C. FreeRTOS podpira delo z več nitmi oz. opravili, mutexi, semaforji in programskimi števci. Ponuja tudi štiri načine alokacije pomnilnika

- samo alokacija,

- alociranje in sprostitev z uporabo enostavnega in hitrega algoritma,
- kompleksna alokacija in sprostitev z uporabo pomnilniškega zlivanja,
- alokacija in sprostitev z uporabo C knjižnjice.

V operacijskem sistemu niso na voljo naprednejše funkcije, kot so npr. gonilniki naprav, uporabniški računi in delo z omrežjem, zato ga lahko interpretiramo kot knjižnjico za delo z nitmi. Za preklop med nitmi uporablja strojni števec, ki proži prekinitvne v naprej definiranem časovnem intervalu (npr. vsako tisočinko sekunde). Pri preklopu se upošteva prioriteta niti in shema krožnega razvrščanja, s katero zagotavlja enakomerno porabo procesorskega časa. Za izmenjavo podatkov med opravili FreeRTOS uporablja vrste [11].

### 2.5.2 Implementacija

Pripravili smo tri opravila, ki opravljajo naloge opisane v predhodnih poglavjih. Glavno opravilo smo poimenovali *vTaskControl* in je zadolženo za inicializacijo GPRS modula in pridobivanje podatkov iz vrste, ter klic funkcije *gprsSend(dhtData,gpsData)*, ki pošlje podatke na strežnik. Opravilu smo določili najvišjo prioriteto in nastavili, da pridobiva in pošilja podatke vsakih 60 sekund.

```
1  while (1) {
2     // prejmi podatke iz vrste
3      statusDHT = xQueueReceive(dht11Que,dhtData,0);
4      statusGPS = xQueueReceive(gpsDataQue,gpsData,0);
5      // ce si pridobil podatke, jih poslji na streznik
6      if (statusGPS == pdTRUE && statusDHT == pdTRUE){
7          gprsSend(dhtData,gpsData);
8      }
9      // pocakaj 60 sekund
10     vTaskDelay(60000 / portTICK_RATE_MS);
11 }
```

V opravilu *vTaskDHT11* smo implementirali zajem podatkov iz senzorja DHT11 in shranjevanje podatkov o meritvah v vrsto, ki se izvede vsakih 45 sekund.

```
int readDataDht[2]={0,0};
while(1){
    vTaskDelay(5000 / portTICK_RATE_MS);
    a=readDht11(data);
    readDataDht[0]=data[0];
    readDataDht[1]=data[2];
    xQueueSendToBack(dht11Que,readDataDht,portMAX_DELAY);
    vTaskDelay(40000 / portTICK_RATE_MS);
}
```

V opravilu *vTaskGPS*, ki se izvede vsakih 45 sekund, smo pridobivali podatke o lokaciji, kot je opisano v razdelku 2.2.4 in jih shranili v vrsto.

## 2.6 Napajanje

Pri izbiri baterije za napajanje smo želeli doseči avtonomijo delovanja, ki jo ima večina današnjih pametnih telefonov, torej minimalno 24 ur neprekinjenega delovanja. Zaradi enostavnosti uporabe smo se odločili, da za napajanje uporabimo polnilno postajo, saj omogoča enostaven priklop in menjavo z uporabo mini USB priključka.

Po pregledu trga smo izbrali polnilno postajo Xiaomi Mi, ki ima kapaciteto 160000 mAh, podjetje pa zagotavlja, da je dejanska kapaciteta postaje vsaj 15460 mAh. S pomočjo multimetra smo izmerili porabo sledilne naprave, ki v povprečju znaša 220 mAh. Pri idealnih pogojih bi tako dobili avtonomijo delovanja približno 70 ur oz. skoraj tri dni. Zaradi vpliva zunanjih dejavnikov je omenjeno avtonomijo težko doseči.

# Poglavje 3

# Strežnik

Kot vmesno točko med sledilno napravo in mobilno aplikacijo, smo vzpostavili strežnik, ki sprejema podatke iz sledilne naprave in jih lokalno shrani. Podatki so nato na voljo aplikaciji, ki jih prikaže uporabniku na zahtevo. V sledečih odsekih je predstavljena vzpostavitev strežniškega okolja, podatkovna baza, strežniška aplikacija, ki je prejmala podatke od naprave in strežniška aplikacija, ki je podatke posredovala odjemalcu.

## 3.1 Gostovanje strežnika

Za gostovanje strežnika smo izbrali ponudnika DigitalOcean, ki ponuja oblačne strežnike pripravljene za razvijalce. Ponudnika smo izbrali, saj ponuja ugodno gostovanje strežnikov, ki temeljijo na SSD diskih. Za pet dolarjev mesečno smo uporabljali strežnik s 512MB pomnilnika, enojedrnim procesorjem, 20GB SSD diska in 1TB vključenega prometa. Strežnik se nahaja v Amsterdamu oz. natančneje v okrožju z oznako AM3, saj je bila lokacija v trenutku priprave strežnika najbližje Sloveniji in tako ponujala minimalno latenco zaradi oddaljenosti.

Izbrali smo operacijski sistem Ubuntu različice 14.04 LTS, ki je poleg Debian, Fedora, CentOS in CoreOS na voljo za hitro namestitev. Po namestitvi operacijskega sistema smo na lokalnem računalniku generirali par

SSH ključev in shranili javno različico ključa na strežniku. Ključa sta nam z uporabo SSH protokola omogočala kriptirano povezavo in s tem varno upravljanje strežnika na daljavo.

## 3.2 Podatkovna baza

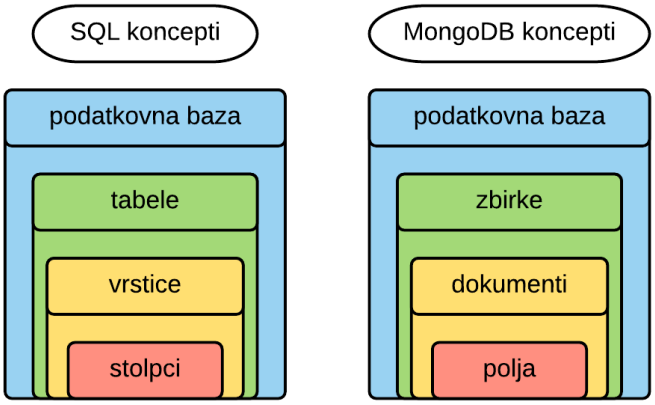
Prejete podatke smo na strežniku hranili v podatkovni bazi realizirani v obliki nerelacijske baze MongoDB. Poleg podatkov o meritvah smo v bazi hranili tudi podatke o uporabnikih in sledilnih napravah. V pričujočem razdelku je opisano delovanje podatkovne baze in njena struktura, več o shranjevanju podatkov in pridobivanju podatkov iz baze pa je opisano v naslednjih razdelkih.

### 3.2.1 Nerelacijske podatkovne baze

Nerelacijske podatkovne baze oz. NoSQL so podatkovne baze, ki ponujajo mehanizme shranjevanja in pridobivanja podatkov, ki ne temeljijo na klasičnih tabelaričnih relacijah. Motivacija za njihov nastanek temelji na močno povečani količini shranjenih podatkov, povečani frekvenci dostopa do podatkov in povečanih procesnih zmogljivostih. Relacijske podatkovne baze niso bile izdelane z namenom skaliranja, kot je potrebno v mnogih modernih aplikacijah, niti niso bile izdelane tako, da bi lahko izkoristile prednosti poceni hrambe podatkov in procesne zmogljivosti, ki nam je na voljo danes. Na sliki 3.1 je prikazana primerjava strukture SQL podatkovne baze in ene izmed NoSQL podatkovnih baz.

Danes najbolj uporabljeni tipi nerelacijskih podatkovnih baz so naslednji:

- 'Baze dokumentov' temeljijo na parih ključa in kompleksne podatkovne strukture - dokumenta. Dokumenti lahko vsebujejo več različnih parov ključ-vrednost, ključ-seznam in vgnezdenih dokumentov. V to kategorijo spada MongoDB.




Slika 3.1: Primerjava konceptov SQL in MongoDB.

- 'Hrambe grafov' se uporabljajo za shranjevanje informacij o omrežjih. Primera tovrstnih baz sta Neo4J in HyperGraphDB.
- 'Hrambe ključ-vrednost' so najenostavnejše nerelacijske podatkovne baze, kjer so vsi podatki v bazi shranjeni kot par atributa in vrednosti. Primeri takih baz so Riak, Voldemort in Redis.
- 'Širokostolpične shrambe' kot sta naprimer Cassandra in HBase, so optimizirane za poizvedbe na veliki podatkovnih virih.

Ena izmed prednosti nerelacijskih podatkovnih baz so dinamične sheme. Klasične relacijske podatkovne baze zahtevajo predhodno definicijo podatkovne sheme, torej v primeru dodajanja novega atributa je potrebno spremeniti celotno bazo. V primeru velikih podatkovnih baz je ta postopek dolgotrajen, baza pa je v času posodabljanja neaktivna. To slabost NoSQL baze odpravljajo, saj namesto vnaprej definiranih shem uporabljajo dinamične sheme, ki omogočajo enostavno dodajanje, odstranjevanje in spreminjanje atributov.

Zaradi načina implementacije so relacijske podatkovne baze narejene za vertikalno skaliranje, pri katerem se morajo vsi podatki nahajati na enem

strežniku zato je tovrstno skaliranje izredno omejeno. Nerelacijske podatkovne baze to težavo rešujejo z horizontalnim skaliranjem, kjer uporabimo več strežnikov na katere razpršimo naše podatke. Celoten postopek razprševanja podatkov in skaliranja se izvaja na nivoju podatkovne baze, tako da se aplikacija tega ne zaveda in vidi podatkovno bazo kot celoto.

Na podlagi predstavljenih prednosti smo se odločili, da pri implementaciji uporabimo odprtokodno dokumentno orientirano nerelacijsko podatkovno bazo MongoDB.

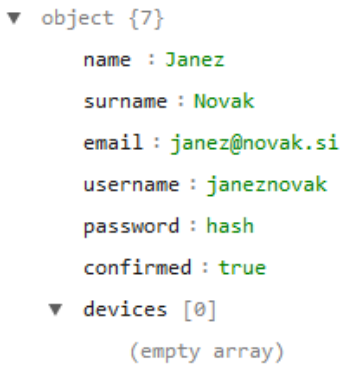
### 3.2.2 MongoDB

MongoDB je dokumentno orientirana podatkovna baza, ki so jo leta 2007 začeli razvijati v podjetju 10gen in dve leti kasneje prešli na odprtokodni razvojni model. Dostopi do podatkovne baze so podprti z mnogimi aplikacijskimi programski vmesniki za različne programske jezike. Pri implementaciji smo uporabili dva vmesnika, in sicer PyMongo, ki ponuja enostaven vmesnik za programski jezik Python, ter uradni vmesnik za programski jezik Node.js. Podrobnejši opis implementacije se nahaja v razdelku 3.2.3.

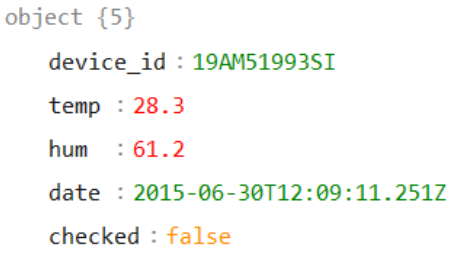
### 3.2.3 Podatkovne sheme

V podatkovni bazi smo ustvarili tri zbirke dokumentov, in sicer zbirko uporabnikov za hranjenje dokumentov o uporabnikih, zbirko meritvev za hranjenje podatkov senzorjev in zbirko za hranjenje podatkov o lokaciji sledilne naprave. MongoDB shranjuje dokumente v obliki JSON formata, zato moramo podatke shraniti v obliki sintaktično pravilnega JSON dokumenta. Na sliki 3.2 je prikazan dokument za shranjevanje uporabnikov.

Za vsakega uporabnika smo shranili njegovo ime, priimek, email in uporabniško ime v klasični tekstovni obliki. Vrednost *confirmed* nam pove, ali je uporabniški račun aktiviran ali ne. V polju *devices* hranimo identifikacijske številke naprav, do katerih ima uporabnik dostop. Geslo je shranjeno v obliki zgoščene vrednosti, ki smo jo ustvarili z uporabo zbirke *bcrypt*. Zbirka upo-




Slika 3.2: Dokument za shranjevanje uporabnikov.

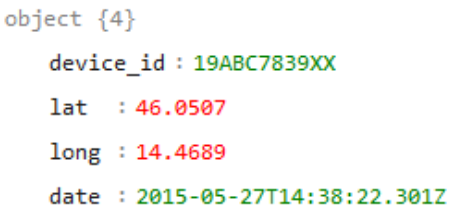



Slika 3.3: Dokument za shranjevanje meritev senzorjev.

rablja Blowfish bločno šifro, ki je bila predstavljena leta 1999, in se uporablja za shranjevanje gesel. Za zaščito pred tabelaričnimi napadi uporablja sol, z večanjem števila iteracij pa se zavaruje pred takoimenovanimi 'brute force' napadi. Vsako geslo je tako varno shranjeno v obliki 60 mestne zgoščene vrednosti.

Na sliki 3.3 je prikazan dokument za shranjevanje podatkov o meritvah senzorjev. Za vsako meritev shranimo identifikacijsko številko naprave, izmerjeno temperaturo v stopinjah celzija in vlažnost v odstotkih. Poleg meritev shranimo tudi časovni žig v ISO 8601 UTC formatu, ki je sestavljen iz datuma in časa. Polje *checked* nam pove, ali je odjemalec že preveril meritev in prikazal obvestilo v primeru neželenega stanja.

Na sliki 3.4 je prikazan dokument za shranjevanje lokacijskih podatkov.




Slika 3.4: Dokument za shranjevanje lokacijskih podatkov.

Poleg podatkov o zemljepisni širini in dolžini shranimo še enolični identifikator sledilne naprave in časovni žig v ISO 8601 UTF formatu.

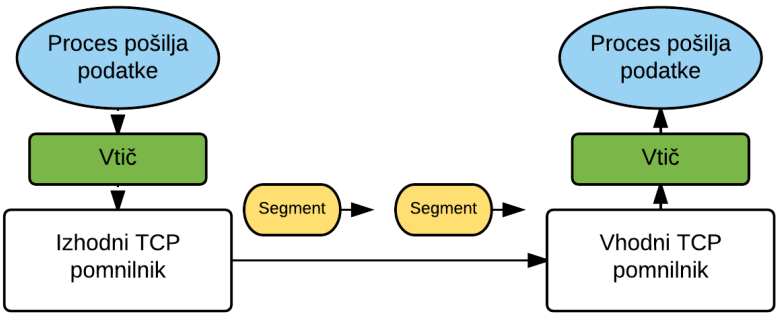
## 3.3 Prejmanje podatkov iz sledilne naprave

Naprava pošilja podatke o meritvah senzorjev in lokacije na strežnik preko GPRS modula na strežnik preko TCP protokola, kot je bilo omenjeno v predhodnem poglavju. V ta namen smo na strežniku pripravili aplikacijo realizirano v programskem jeziku Python, ki izpostavlja TCP vtič na katerega se naprava poveže, ko želi oddati podatke. Podroben opis protokola in implementacije prejemanja podatkov je predstavljen v naslednjih podpoglavjih.

### 3.3.1 Protokol TCP

TCP oz. protkol za nadzor prenosa je skupek protokolov, ki tvorijo protokolski sklad preko katerega teče medmrežni promet. Protokol je povezavno orientiran, kar pomeni, da morata procesa, ki želita komunicirati, izvesti 'rokovanje' v katerem si izmenjata segmente s parametri za vzpostavitev povezave. Za vzpostavitev povezave mora odjemalec svoji transportni plasti posredovati naslov strežnika s katerim se želi povezavi in vrata na katerih strežnikov proces pričakuje podatke.

Ko je povezava vzpostavljena lahko procesa začneta izmenjevati podatke. Povezava deluje v full-duplex načinu, kar pomeni, da če imamo na prvem gostitelju proces X in na drugem gostitelju proces Y, potem lahko podatki




Slika 3.5: Prikaz enosmernega pošiljanja.

hkrati potujejo od procesa X k procesu Y in obratno. Podatki se pri pošiljanju shranjujejo v izhodni TCP pomnilnik, kjer se razbijejo na manjše segmente, ki se nato posredujejo omrežni plasti, kjer se enkapsulirajo v datagrame in pošljejo po omrežju. Na sliki 3.5 je prikazan primer enosmernega pošiljanja podatkov.

TCP omogoča zanesljiv prenos podatkov po nezanesljivem omrežju. Za zagotavljanje zanesljivosti omogoča ponovno pošiljanje podatkov v primeru, da preteče vnaprej določen časovni interval (timeout), brez prejetega potrdila (ACK). Do omenjenega primera lahko vodita dva scenarija, bodisi se je izgubil paket s podatki, bodisi se je izgubil paket s potrditvijo. V primeru, da se je izgubil paket s podatki, bo prejemnik ob ponovnem pošiljanju prejel paket in potrdil prejem s potrditvijo pošiljatelju. V primeru, da se je izgubil paket s potrditvijo, bo prejemnik zavrgel duplikat paketa s podatki in ponovno poslal potrditev. Pri protokolu TCP se uporablja kumulativno potrjevanje, kar pomeni, da potrditev paketa X hkrati potrjuje vse predhodne pakete.

### 3.3.2 Implementacija

Za implementacijo TCP strežniške aplikacije smo izbrali splošno namenski visoko nivojski programski jezik Python. Izbrali smo ga, ker ponuja vmesnik

za podatkovno bazo MongoDB in enostavno delo z nizi. Vtič smo implementirali z uporabo vgrajene knjižnice *socket*, ki predstavlja nizko nivojski vmesnik za delo z omrežnimi komponentami.

Strežniško aplikacijo smo nastavili tako, da posluša na vratih 8124 in v primeru vzpostavljene povezave pokliče funkcijo, ki bo podatke sprejela, obdelala in shranila v podatkovno bazo. Omenjena funkcija je prikazana v sledečem izseku programske kode.

```
1  def response(data):
2      // razbij prejete podatke
3      mydata = data.split(',')
4
5      // vstavi podatke v bazo
6      sensorJSON = {"device_id" : dev_id,
7                    "temp" : float(mydata[1]),
8                    "hum"  : float(mydata[0]),
9                    "date" : datetime.datetime.utcnow(),
10                   "checked" : false}
11     trackJSON = {"device_id" : dev_id,
12                  "lat" : float(mydata[2]),
13                  "long" : float(mydata[3]),
14                  "date" : datetime.datetime.utcnow()}
15
16     mDB.sensors.insert_one(sensorJSON);
17     mDB.tracks.insert_one(trackJSON);
18
19     return 'OK'
```

Funkcija sprejme podatke v obliki niza, ki vsebuje posamezne podatke ločene z vejico, zato jih s klicom funkcije *split* razbijemo v seznam podnizov. V seznamu si podatki sledijo v sledečem vrstnem redu: vlažnost, temperatura, zemljepisna širina in zemljepisna dolžina. Podatke nato shranimo v dva ločena JSON dokumenta. V prvi dokument shranimo podatke o meritvah temperature in vlažnosti. Meritvam dodamo polje *checked*, ki se bo uporabilo pri prikazu obvestil o stanju naprave. V drugi dokument shranimo lokacijske meritve. Oba dokumenta dopolnimo z identifikatorjem naprave in

časovnih žigom v ISO 8601 UTC formatu.

Dokumenta shranimo v podatkovno bazo MongoDB s klicem funkcije *insert_one*, ki jo ponuja aplikacijski programski vmesnik PyMongo. PyMongo je eden izmed mnogih aplikacijskih programskih vmesnikov, ki so na voljo za delo s MongoDB, in ponuja osnovna orodja za shranjevanje, pridobivanje in brisanje podatkov iz baze.

Povezava s strežnikom je vzpostavljena dokler jo ne zapre odjemalec s klicem lokalne funkcije za zapiranje povezave oz. pošlje na strežnik niz "close". V primeru, da odjemalec ne zapre povezave, se bo le ta zaprla sama po določenem časovnem intervalu neaktivnosti.

## 3.4 Strežniška aplikacija za odjemalca

Za posredovanje podatkov mobilni aplikaciji smo pripravili strežniško aplikacijo, ki navzven izpostavlja REST storitve, ki jih kliče odjemalec, ko želi pridobiti oz. poslati podatke na strežnik.

REST predstavlja arhitekturni stil, ki zagotavlja skalabilnost, splošnost in neodvisnost aplikacij. Aplikacije, zgrajene po tem arhitekturne stilu, uporabljajo HTTP zahteve za pošiljanje, branje in brisanje podatkov. Na sliki 3.6 je prikazan primer klica REST storitve. Ob klicu storitve mora odjemalec poslati na vnaprej določen naslov HTTP zahtevo, pri kateri mora definirati kakšen tip podatkov pošilja oz. sprejema. Z nastavljanjem parametrov v naslovu lahko vpliva na odziv storitve (npr. izvede filtriranje ali pa poda dodatne podatke pri poizvedbi). REST je iz vidika odjemalca enostaven, saj mora poznati le naslove storitev in tipe podatkov, ki jih storitev zahteva oz. vrača.

Za realizacijo aplikacije smo uporabili programski jezik Node.js. Več o implementaciji je zapisano v razdelku 3.4.2.


Slika 3.6: Primer klica REST storitve.

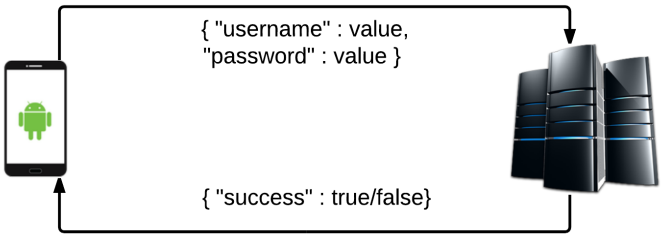
### 3.4.1 Node.js

Node.js je odprtokodno izvajalno okolje, ki je namenjeno predvsem strežniškim in omrežnim aplikacijam. Okolje ponuja dogodkovno vodeno arhitekturo in neblokirajoče vhodno-izhodne operacije, kar optimizira prepustnost in skalabilnost aplikacije. Node.js temelji na programskem jeziku Javascript in za izvajanje skript uporablja Googleov V8 Javascript engine. S pomočjo vgrajenih knjižnic omogoča aplikacijam, da igrajo vlogo spletnega strežnika brez uporabe namenskih aplikacij (npr. Apache HTTP Server).

Za hitrejšo implementacijo najbolj uporabljenih funkcionalnosti je na voljo že več okvirjev, npr. Express.js, Socket.io in Connect. V projektu smo uporabili okvir Express.js, ki ponuja minimalističen okvir za gradnjo spletnih aplikacij in integracijo s podatkovno bazo. Poleg okvirja smo uporabili tudi nekaj dodatnih modulov.

Za dodajanje modulov smo uporabili *npm*, ki omogoča enostavno dodajanje preko ukazne vrstice. Uporabili smo naslednje dodatne module:

- *bcrypt* za varno shranjevanje gesel v obliki zgoščenega zapisa (podrobnejši opis delovanja se nahaja v razdelku 3.2.3)
- *morgan* za arhiviranje podatkov o dostopih do strežnika in klicanih servisih
- *forever* za zagon strežnika v ozadju in nadzor delovanja




Slika 3.7: Primer prijave v sistem.

### 3.4.2 Implementacija aplikacije

Strežniška aplikacija pričakuje zahteve na vratih 7777. Za uporabo prijavnega sistema je moral odjemalec klicati servis na podnaslovu */login*. Ob klicu servisa je moral HTTP zahtevi dodati veljaven JSON dokument, ki vsebuje polji *username* in *password*. Servis je nato primerjal zgoščeno vrednost prejetega gesla z zgoščeno vrednostjo pravega gesla in vrnil odgovor z JSON dokumentom, ki je vseboval polje *success*, ki je vsebovalo logično vrednost odvisno od primerjave gesel. Primer izvedbe poizvedbe je razviden na sliki 3.7.

Za dodajanje naprav uporabniku mora odjemalec poklicati servis, ki je dostopen na podnaslovu */add_device* in poslati JSON dokument, ki vsebuje uporabniško ime uporabnika in identifikacijsko številko naprave. V odgovoru aplikacija posreduje uspešnost izvedbe servisa.

Za pridobivanje podatkov o lokaciji naprave je dostopen servis na podnaslovu */get_track*. Ob klicu servisa je potrebno podati identifikacijsko številko naprave in datum. Strežnik bo odgovoril z vrsto meritev, ki so bile pridobljene od podanega datuma do današnjega dne. V primeru da je vrsta prazna pomeni, da za iskano napravo ni bilo meritev oz. uporabnik nima pravic, da bi poizvedel po meritvah iskane naprave. Na enak način deluje tudi servis za pridobivanje podatkov o meritvah senzorjev, ki je dostopen na */get_sensors*.

Odjemalcu je na voljo tudi servis za pridobivanje seznama vseh naprav

uporabnika na */get_devids*. Servisu moramo podati uporabniško ime, od njega pa prejmemo seznam naprav podanega uporabnika.

Za prikaz obvestil smo pripravili servis */get_notif*, ki sprejme zahtevo s podatki o napravi, najvišji in najnižji želeni vlažnosti, ter najvišji in najnižji želeni temperaturi. Aplikacija odgovori z dokumentom, v katerem nam vrednost spremenljivke *alarm* pove ali se naprava nahaja v kritičnem stanju ali ne.

# Poglavje 4

# Mobilna aplikacija

Za nadzor delovanja sledilne naprave in vizualizacijo meritev smo pripravili mobilno aplikacijo za pametne telefone ali tablice z operacijskim sistemom Android verzije 5.0 ali višje. Meritve temperature in vlažnosti nam aplikacija prikaže v obliki grafa za izbrano časovno obdobje, lokacijske podatke pa prikaže kot točke na zemljevidu. V primeru, da se temperatura ali vlažnost v okolici sledilne naprave nahaja zunaj želenega intervala, nas bo aplikacija na to opozorila s prikazom obvestila.

Podrobnejši opis delovanja in implementacije aplikacije se nahaja v sledečih razdelkih.

## 4.1 Android

Android je mobilni operacijski sistem, ki temelju na modificirani verziji Linux jedra. Z razvojem je začelo istoimensko startup podjetje Android Inc., ki ga je leta 2005 kupil Google s strategijo prehoda na trg mobilnih naprav. Google želi, da je Android odprt in brezplačen, zato je večina izvorne kode prosto objavljena pod odprtokodno licenco. Mnogi proizvajalci mobilnih naprav to dejstvo izkoristijo in v operacijski sistem, ki ga namestijo na svoje naprave, vključijo tudi lastniške aplikacije. Z začetkom leta 2015 je Android postal najširše uporabljen mobilni operacijski sistem, zato predstavlja velik trg za

vse razvijalce mobilnih aplikacij.

Od svojih začetkov, do danes je operacijski sistem prešel čez večje število verzij. V trenutku pisanja diplomskega dela je zadnja stabilna verzija 5.1 z razvojnim imenom *Lollipop*, na voljo pa je tudi razvijalski predogled za naslednjo verzijo z oznako "M".

Arhitektura operacijskega sistema je sestavljena iz štirih slojev oz. petih sekcij, in sicer

- Linux monolitnega jedra, ki vsebuje nizko nivojske gonilnike za zunanje naprave in drugo podporo za strojno opremo,
- knjižnice, ki ponujajo glavne funkcije za delo z operacijskim sistemom,
- Android izvajalno okolje, ki se nahaja na istem sloju kot knjižnice in omogoča pisanje aplikacij v programskem jeziku Java,
- aplikacijski okvir, ki izpostavlja določene funkcionalnosti operacijskega sistema razvijalcem aplikacij,
- aplikacije.

### 4.1.1 Aktivnosti

Aktivnost je komponenta aplikacije, ki ponuja okno preko katerega je uporabnik v interakciji z aplikacijo. Okno predstavlja uporabniški vmesnik, ki ponavadi obsega celoten zaslon ali manjši del zaslona. Aplikacija je sestavljena iz ene ali več aktivnosti, od katerih mora biti ena določena kot glavna aktivnost, ki se prikaže ob prvem zagonu aplikacije. Aktivnost lahko zažene drugo aktivnost, z namenom izvršitve določene naloge. Zagon nove aktivnosti se izvrši z uporabo razred *Intent*, ki mu lahko podamo dodatne podatke s katerimi naj se aktivnost zažene. Ob zagonu nove aktivnosti je predhodna aktivnost ustavljena, njeno stanje pa je shranjeno na vrh sklada. Ko se uporabnik želi vrniti na predhodno aktivnost je le ta obnovljena iz vrha sklada, kar omogoča hiter preklop med aktivnostmi.

Vsaka aktivnost ima vnaprej določen življenjski cikel, ki je definiran s stanji, med katerimi prehajamo s klicem funkcij življenjskega cikla

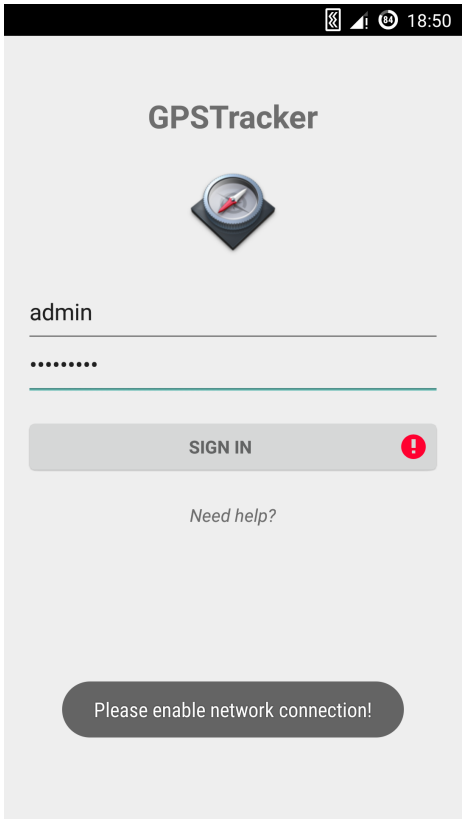
- *onCreate*: ob prvem kreiranju aktivnosti,
- *onStart*: preden aktivnost postane vidna uporabniku,
- *onResume*: ko aktivnost postane vidna uporabniku,
- *onPause*: ko druga aktivnost prevzame fokus,
- *onStop*: ko aktivnost ni več vidna uporabniku,
- *onDestroy*: ko se aktivnost "uniči" (npr. ko zapremo aplikacijo).

## 4.2 Prijavno okno

Prvo okno, ki se prikaže uporabniku po kliku na ikono aplikacije, je prijavno okno. V ozadju se sprva zažene glavna aktivnost, ki v prvi funkciji svojega življenjskega cikla *onCreate()*, preveri, ali je uporabnik prijavljen v sistem ali ne. V primeru, da uporabnik ni prijavljen zažene prijavno aktivnost preko razreda *Intent* in počaka na njegov rezultat, kot je prikazano v sledečem izseku programske kode.

Podatki o prijavi uporabnika so shranjeni v razredu *SharedPreferences*[13], ki predstavlja okvir za shranjevanje in pridobivanje parov ključ-vrednost primitivnih podatkovnih tipov. Podatki shranjeni v tem razredu so na voljo vsem aktivnostim znotraj aplikacije in se ohranjajo tudi, ko aplikacijo zapremo.

```
1 private SharedPreferences mySettings;
2 public void checkLogin(){
3    needsLogin = mySettings.getBoolean("needsLogin",true);
4    if(needsLogin){
5       Intent intent = new Intent(this,LoginActivity.class);
6       startActivityForResult(intent, LOGIN_REQUEST);
7    }
8 }
```




Slika 4.1: Zaslonska slika prijavnega okna.

Prijavna aktivnost ob kliku na gumb *Sign in* preveri, če so bili vnešeni vsi potrebni podatki in če ima aplikacija dostop do omrežja. V primeru napake izpiše obvestilo v obliki pojavnega okna in ob robu gumba prikaže rdeč klicaj, kot je prikazano na sliki 4.1.

Aktivnost izvede prijavo z uporabo razreda *AsyncTask*[12], ki je namenjen izvajanju dolgotrajnih operacij v ozadju in vračanju rezultatov glavni niti. V omenjenem razredu zgradimo JSON dokument z zajetimi prijavnimi podatki in kličemo REST storitev za prijavo uporabnika na strežniku. Strežnik odgovori z dokumentom, v katerem aktivnost seznani z uspešnostjo prijave. Če je bila prijava uspešna se prijavna aktivnost zaključi. Glavna aktivnost preveri status, ki ga je vrnila prijavna aktivnost in posodobi vrednosti *SharedPreferences* in shrani uporabniško ime prijavljenega uporabnika.

## 4.3 Glavno okno

Po prijavi se uporabniku prikaže uporabniški vmesnik glavne aktivnosti, ki je prikazan na sliki 4.2. Ob prvem prikazu glavnega vmesnika je potrebno klikniti na gumb za osvežitev podatkov, ki se nahaja v opravilni vrstici. Pri tem se s strežnika pridobijo podatki za vse naprave uporabnika in se shranijo v lokalno podatkovno bazo v mobilni napravi. Vmesnik nam omogoča izbiro naprave uporabnika, za katero želimo prikazati podatke, in izbiro časovnega intervala prikazanih podatkov. Pri izbiri časovnega intervala lahko izberemo prikaz vseh podatkov, ali pa prikaz podatkov od izbranega datuma dalje. Pri izbiri druge možnosti se nam odpre pogovorno okno v katerem izberemo želeni datum. S klikom na gumb *SHOW SENSOR DATA* nam aplikacija izriše graf spreminjanja temperature in vlažnosti za izbrani časovni interval, s klikom *SHOW MAP* pa se zažene nova aktivnost, ki prikaže zemljevid z lokacijski podatki.

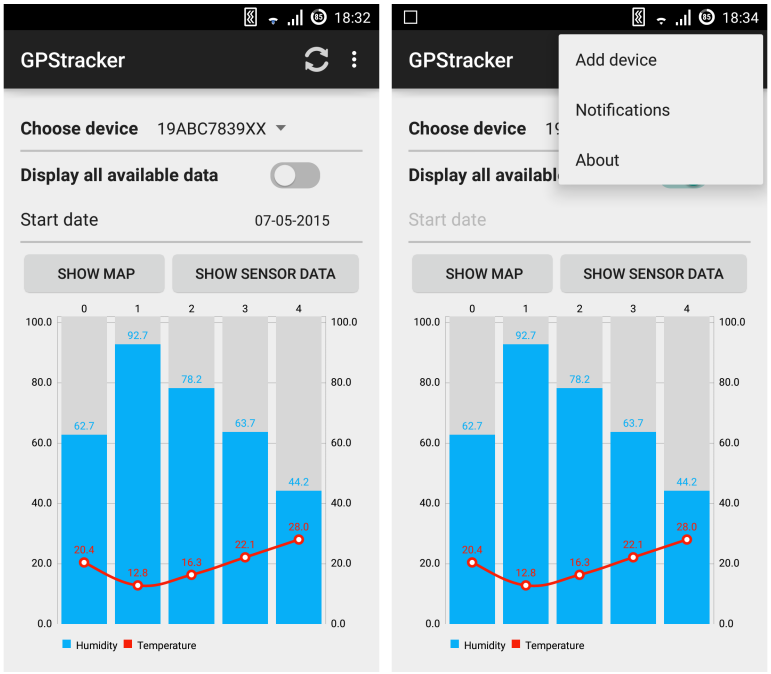
Ob kliku na ikono treh pik v opravilni vrstici se odpre meni nastavitev, ki omogoča dodajanje nove naprave uporabniku, spreminjanje nastavitev obvestil in glavne podatke o aplikaciji.

### 4.3.1 Lokalna podatkovna baza

Ena izmed možnosti za shranjevanje podatkov, ki jih ponuja operacijski sistem Android, je lokalna podatkovna baza SQLite. SQLite je odprtokodna podatkovna baza, ki podpira standard relacijskih podatkovnih baz in SQL sintakso. Za delovanje potrebuje le manjšo količino delovnega pomnilnika (okoli 250KB), zato je primerna za vgradnjo v druge aplikacije.

Podatkovna baza je vgrajena v vse Android naprave, zato nam pri uporabi ni potrebno upravljati baze. Za vstavljanje, brisanje in posodabljanje podatkov se uporablja razred *SQLiteOpenHelper*, v katerem pripravimo SQL stavke, ki jih nato kličemo s klicem funkcij. Manipulacija podatkovne baze se tako prevede na klice funkcij razreda.

Ob posodobitvi lokalnih podatkov so se s strežnika prenesli vsi podatki za




Slika 4.2: Zaslonski sliki glavnega okna.

naprave uporabnika, ter se shranili v lokalno podatkovno bazo. Pri prenosu podatkov je potrebno podatke pretvoriti iz JSON dokumenta v osnovne podatkovne tipe, ki jih podpira SQLite (nizi, cela števila in realna števila). Ob klicu prikaza podatkov se poizvedba izvede nad lokalno bazo, kar omogoča prikaz podatkov tudi kadar nimamo na voljo omrežne povezave.

### 4.3.2 Prikaz meritev

Del glavnega okna aplikacije predstavlja graf za vizualizacijo, ki je sestavljen iz stolpičnega diagrama za prikaz vlažnosti in lomljenke za prikaz temperature. Kot osnovo smo uporabili knjižnico *MPAndroidChart*[14], ki poleg prikaza omogoča tudi manipulacijo grafa (povečava, pomanjšanje, premikanje, ...). V primeru, da podatkov za izris ni na voljo, se graf ne bo izrisal, prikazalo pa se bo obvestilo.

## 4.4 Zemljevid

Za prikaz lokacijskih podatkov smo uporabili Googlove zemljevide, ki so za Android na voljo preko Google Maps aplikacijskega programskega vmesnika. Ob kliku na gumb *SHOW MAP* na glavnem oknu aplikacije se sproži nova aktivnost, ki prikaže zemljevid z označenimi točkami lokacijskih meritev, kot je prikazano na sliki 4.3.

```
Intent intent = new Intent(this,MapsActivity.class);
intent.putExtra("lats",lats);
intent.putExtra("longis",longis);
intent.putExtra("dates",dates);
startActivity(intent);
```

Ob zagonu aktivnosti je potrebno dodati seznam zemljepisnih dolžin, zemljepisnih širin in datumov, ki jih aktivnost nato prikaže na zemljevidu. Za vsako točko se izriše marker, ki ob kliku izpiše tudi datum meritve, vsakemu markerju pa smo dodali tudi krožnico z radijem 50m, ki označuje varianco meritve.

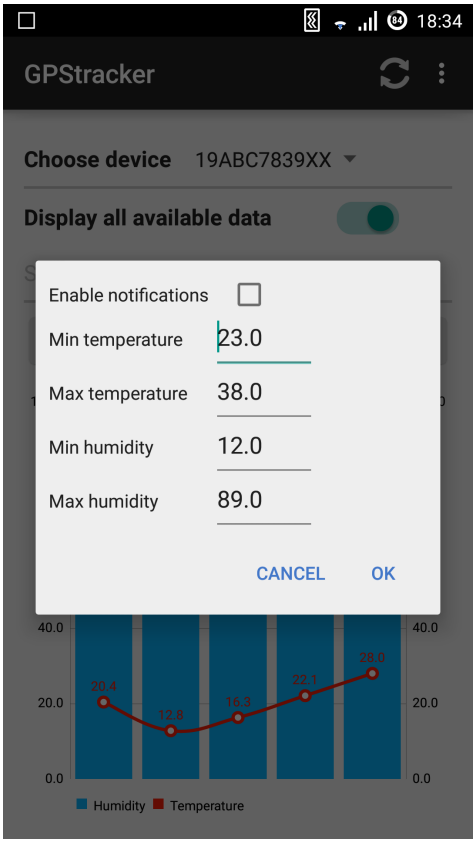
```
private void setUpMap() {

   for(int i = 0; i < lats.length; i++){
      mMap.addMarker(new MarkerOptions()
      .position(new LatLng(lats[i],longis[i]))
      .title(dates[i]));
      mMap.addCircle(new CircleOptions()
      .center(new LatLng(lats[i],longis[i]))
      .radius(50)
      .fillColor(Color.argb(60,23,170,254))
      .strokeColor(Color.BLUE)
      .strokeWidth(3));
    }

   mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(new LatLng(
       lats[0],longis[0]),10));
}
```


Slika 4.3: Zaslonski slika zemljevida.

Za uporabo zemljevidov je potrebno aplikacijo registrirati na spletni strani *Google Developers*, kjer nato pridobimo API ključ za uporabo v aplikaciji. Google Maps API za Android avtomatično upravlja z dostopi do Google Maps strežnikov, prenosom podatkov in izrisi zemljevidov. S klici funkcij vmesnika lahko na zemljevid dodajamo markerje, lomljenke in spreminjamo uporabnikov pogled na zemljevid.

## 4.5 Prikaz obvestil

Aplikacija omogoča tudi možnost prikaza obvestil v primeru, da je meritev temperature ali vlažnosti zunaj želenega intervala. Do nastavitev obvestil




Slika 4.4: Zaslonska slika pogovornega okna.

pridemo preko menija nastavitev, ki je del glavnega okna. Ob kliku se odpre pogovorno okno, ki je prikazano na sliki 4.4. V oknu označimo, ali želimo prejemati obvestila in kakšne so mejne temperature in vlažnost za prikaz obvestila. V primeru, da omogočimo obvestila, bo aplikacija vsakih 15 minut izvedla klic strežniške storitve in tako preverila, ali se naprava nahaja v kritičnem stanju.

### 4.5.1 Implementacija

Prikaz obvestil smo realizirali z uporabo razredov *Service*, *BroadcastReciever* in *AlarmManager*. Ko označimo, da želimo prejemati obvestila aktiviramo instanco razreda *AlarmManager*, ki periodično na vsakih 15 minut proži

alarm, ki ga ujame instanca razreda *BroadcastReciever*. Omenjena instanca ob prejetem alarmu zažene novo aktivnost razreda *Service*, ki teče v ozadju. Aktivnost od strežnika pridobi stanje naprave, na podlagi katerega se odloči, ali bo prikazala obvestilo.

Omenjeni način implementacija omogoča delovanje v ozadju tudi takrat, ko aplikacija ni aktivna in hkrati minimizira porabo energije, saj je servis aktiven le nekaj sekund na vsakih 15 minut.

# Poglavje 5

# Zaključek

V okviru diplomskega dela smo razvili celovito rešitev, s katero lahko izbrani napravi ali izdelku dodamo funkcionalnost lokacijskega sledenja in merjenja temperature ter vlažnosti. Sledilno napravo smo izdelali z uporabo mikrokrmilnika STM32F4-Discovery, ki smo ga razširili z GPS in GPRS modulom, ter senzorjem za temperaturo in vlažnost. Sistem nam ponuja natančne in zanesljive podatke, ki se pošiljajo na strežnik v naprej določenem intervalu. Podatki se na strežniku procesirajo in shranijo v podatkovno bazo, iz katere jih pridobi mobilna aplikacija. Aplikacija pa poleg prikazovanja podatkov omogoča tudi prikaz obvestil v primeru kritične vrednosti temperature in vlažnosti, tako da lahko takoj ukrepamo.

Pri testiranju sledilne naprave smo ugotovili, da lokacijske meritve sovpadajo z meritvami, ki jih pridobimo iz namenske naprave za GPS lociranje, prav tako niso odstopali podatki o temperaturi in vlažnosti.

Možnosti za nadgradnje sledilne naprave so predvsem povezane z optimizacijo delovanja. Z uporabo vgrajenega pospeškomera bi lahko aktivirali sledenje oz. ga deaktivirali v primeru, da se naprava dlje časa ne bi premaknila. Tako bi dosegli manjšo porabo baterije in posledično daljšo avtonomijo delovanja.

# Literatura


[1] ARM infogram za Q1 2015. [Online]. Dosegljivo: http://media.corporate-ir.net/media_files/IROL/19/197211/Results/Q1/Q1-2015-ARM-infographic-1024x512.png. [Dostopano 30. 6. 2015].

[2] ARM letni načrt 2015. [Online]. Dosegljivo: http://ir.arm.com/phoenix.zhtml?c=197211&p=irol-reportsother. [Dostopano 30. 6. 2015].

[3] Satelitska navigacija (Wikipedia). [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Satellite_navigation. [Dostopano 1. 7. 2015].

[4] Podatki o seriji LEA-6. [Online]. Dosegljivo: http://www.u-blox.com/images/downloads/Product_Docs/LEA-6_ProductSummary_%28GPS.G6-HW-09002%29.pdf. [Dostopano 1. 7. 2015].

[5] Specifikacije GPS protokolov u-blox modulov. [Online]. Dosegljivo: http://www.u-blox.com/images/downloads/Product_Docs/u-blox6_ReceiverDescriptionProtocolSpec_%28GPS.G6-SW-10018%29.pdf. [Dostopano 1. 7. 2015].

[6] GPS knjižnica za STM32F4-Discovery. [Online]. Dosegljivo: http://stm32f4-discovery.com/2014/08/library-27-gps-stm32f4-devices/. [Dostopano 1.7.2015].

[7] Specifikacije senzorja DHT11. [Online]. Dosegljivo: http://www.micropik.com/PDF/dht11.pdf. [Dostopano 1.7.2015].

[8] Weiping Liu, Yanwen Liu, Ru Li, and Pai Wang. Research and development of communication between pc and mobile base on embedded system and gprs. In *Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), 2011 2nd International Conference on*, pages 4180–4183, Aug 2011.

[9] Dokumentacija AT ukazov. [Online]. Dosegljivo: http://www.adafruit.com/datasheets/sim800_series_at_command_manual_v1.01.pdf. [Dostopano 2.7.2015].

[10] Dokumentacija SIM800H. [Online]. Dosegljivo: http://www.simcom.eu/media/files/SIM800H.PDF. [Dostopano 2.7.2015]

[11] FreeRTOS. [Online]. Dosegljivo: http://www.freertos.org/. [Dostopano 3.7.2015].

[12] Android AsyncTask. [Online]. Dosegljivo: http://developer.android.com/reference/android/os/AsyncTask.html. [Dostopano 5.7.2015].

[13] Android Storage Options. [Online]. Dosegljivo: http://developer.android.com/guide/topics/data/data-storage.html. [Dostopano 5.7.2015].

[14] MPAndroidChart repozitorij. [Online]. Dosegljivo: https://github.com/PhilJay/MPAndroidChart. [Dostopano 6.7.2015].