Střední průmyslová škola Třebíč
Manželů Curieových 734, 674 01 Třebíč

Maturitní práce

APLIKACE NA OVLÁDÁNÍ IOT DOMÁCNOSTI

Profilová část maturitní zkoušky

Studijní obor: Informační technologie

Třída: ITB4

Školní rok: 2019/2020

Antonín Svitáček

ABSTRAKT

Moje ročníková práce se zabývá tvorbou IOT aplikace. Aplikace dokáže sdružovat IOT prvky v domácnosti a vytváří tak jednoduché prostředí pro jejich řízení a čtení dat. Tyto prvky jsou připojené k síti pomocí Wi-Fi a jsou provozované na procesoru ESP8266. Všechny tyto moduly pracují jako samostatné jednotky, které mají přístup k databázi, kde ukládají data. Pomocí těchto dat aplikace dokáže vytvořit jednoduchý graf. Aplikace také u řídících prvků dokáže vytvořit makro, podle kterého se bude následně modul rozhodovat, jestli je daná podmínka pravdivá a poté vykoná příslušnou akci.

KLÍČOVÁ SLOVA

IOT domácnost, Wi-Fi moduly, IOT aplikace, Internet věcí, Automatizace domácnosti, chytrá domácnost, řízení domácnosti, IOT zásuvka, IOT sensory

ABSTRACT

My thesis deals with the development of IOT application. The application can associate IOT elements in the home and creates a simple environment for their management and reading data. These elements are connected to the network via Wi-Fi and run on the ESP8266 processor. All these modules work as separate units that have access to the database where they store data. Using this data, the application can create a simple graph. The application can also create a macro for the control module, according to which the module will then decide if the given condition is true and then take the appropriate action.

KEYWORDS

IOT home, Wi-Fi modules, IOT app, Internet of things, Home automation, smart home, home controlling, IOT plug, IOT sensors

PODĚKOVÁNÍ

Děkuji panu Ing. Ladislavovi Havlátovi za odborný dohled nad tímto ročníkovým projektem.

V Třebíči dne 19. března 2020 .....

podpis autora

PROHLÁŠENÍ

Prohlašuji, že jsem tuto práci vypracoval samostatně a uvedl v ní všechny prameny, literaturu a ostatní zdroje, které jsem použil.

V Třebíči dne 19. března 2020 .....

podpis autora

OBSAH

ÚVOD

Mnou vytvořená aplikace má za cíl vytvořit jednoduché prostředí pro řízení IOT prvků domácnosti. Pomocí aplikace máte možnost tyto prvky konfigurovat a číst hodnoty ze senzorů. IOT prvky jsou rozděleny do dvou skupin senzory a spínače. Spínače si dokáží brát hodnoty ze senzorů. Pomocí aplikace také lze nastavit četnost čtení ze senzorů a nastavit řídící proměnné pro řízení spínačů. Aplikace má u každého senzoru možnost napojení databáze a s dostupnými daty dokáže vypracovat graf. Zatímco u skupiny řídících prvků je napojení do databáze zřízeno pro snadné ovládání prvku i když nejste v jeho lokální síti.

1 TEORETICKÁ ČÁST

V této části popíši použité technologie a následně vysvětlím, jak by principiálně měli být použity v mém projektu.

1.1 Použité technologie

1.1.1 C++

C++ představuje multiparadigmatický programovací jazyk, který vznikl na základě rozšíření jazyka C. Spolu s jazykem Java patří mezi nejpoužívanější. Jeho základním rysem je podpora několika programovacích stylů a kombinování syntaxe, což umožňuje objektově orientované, strukturované a generické programování.[1]

V jazyce C++ jsou psané veškeré Wi-Fi moduly, protože jazyk C++ je vysoce výkonný a umožňuje některé velmi výhodné konstrukce. Mezi jazyky na programování jedno čipů jsem měl na výběr mezi MikroPython, C, C++ a Assemblerem. MicroPython je velice zajímavá alternativa, protože není kompilovaný, tudíž se vždy do paměti nahraje pouze zdrojový kód, ale tento postup nese velké využití paměti, protože musíme prvně do paměti nahrát firmware, který bude kód číst a provádět. Assembler je nízkoúrovňový jazyk a programování v tomto jazyce je velice nekomfortní.

1.1.2 PHP

PHP je skriptovací programovací jazyk. Je určený především pro programování dynamických internetových stránek a webových aplikací například ve formátu HTML, XHTML či WML.[2]

V našem případě ve výstupu PHP skriptu nemusí být HTML, protože my nebudeme výstup zobrazovat ve formě webové stránky. PHP nám umožňuje číst data z databáze a v nám určeném formátu je odešle klientovy.

1.1.3 JAVA

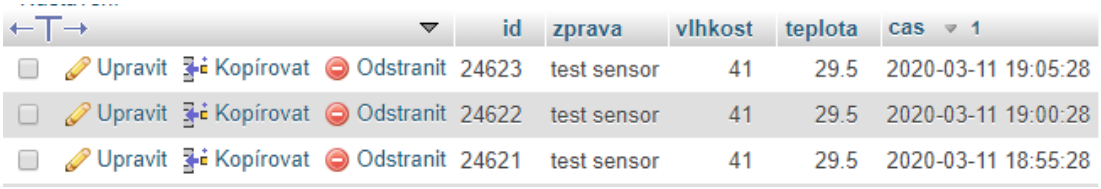
Nespornou výhodou tohoto programovacího jazyka je jeho přenositelnost. Díky ní jej mohou používat programy pracující na různých systémech. Jedná se o platformy pro čipové karty, mobilní telefony, dále samozřejmě také aplikace pro desktopové počítače. Java se stala základním kamenem pro tvorbu open source systému, se kterým se setkáváme v chytrých telefonech a tabletech. Zejména Android umožňuje její široké využití.

Tento objektově orientovaný programovací jazyk do určité míry vychází ze syntaxe C++. Java je charakteristická zjednodušenou syntaxí bez nízkoúrovňových konstrukcí, dále nabízí generační správu paměti, nezávislost na architektuře a přenositelnost v rámci jedné platformy.[1]

1.1.4 MySQL

MySQL je databázový systém vytvořený švédskou firmou MySQL AB. Jedná se o multiplatformní databázi a komunikace s ní probíhá za pomoci jazyka SQL. Je možné ji instalovat na různé operační systémy např. Linux, Windows, atd. Velmi často bývá nasazována v kombinaci s PHP, Apache jako základ webového serveru.[3]

MySQL je jedna z nejpoužívanějších databázových systémů. Díky MySQL můžeme přistupovat k datům a ukládat je do přehledných tabulek. Tyto tabulky můžeme řadit a filtrovat.



		id	zprava	vlhkost	teplota	cas 1
Upravit Kopírovat Odstranit		24623	test sensor	41	29.5	2020-03-11 19:05:28
Upravit Kopírovat Odstranit		24622	test sensor	41	29.5	2020-03-11 19:00:28
Upravit Kopírovat Odstranit		24621	test sensor	41	29.5	2020-03-11 18:55:28

Obrázek 1 MySQL tabulka

1.1.5 SQLite

SQLite je knihovna psaná v C, implementuje malý, samostatný, vysoce stabilní, plně vybavený SQL databázový engine. SQLite je nejvíce užívaný databázový engine na světě. SQLite je vestavěn do všech telefonů a většiny počítačů a je dodáván v nespočtu dalších aplikací, které lidi používají každý den. [4]

SQLite umožňuje v naší aplikaci ukládání jednotlivých prvků podle id. SQLite je také snadno přenositelná a jelikož není spuštěna na serveru, ale přímo v našem mobilním telefonu, tak nepotřebuje připojení k internetu. Soubor s obsahem databáze bývá označen koncovkou dbm.

1.1.6 SQL

Structured Query Language (Strukturovaný dotazovací jazyk) - je obecný nástroj pro manipulaci, správu a organizování dat uložených v databázi počítače. Je v prvé řadě určen uživatelům, i když jej v mnoha směrech využívají i tvůrci aplikací. Je adaptovatelný pro jakékoliv prostředí.[5]

Tento dotazovací jazyk požívá v mém projektu jak serverová část, tak i Wi-Fi moduly, které posílají strukturované dotazy na MySQL server ať už za účelem uložení nebo čtení dat z databáze.

```
SELECT * FROM `to ti nereknu` ORDER BY `to ti nereknu`.`cas` DESC;
```

1.2 Použité editory

1.2.1 Arduino IDE

Vývojové prostředí Arduina sestává z textového editoru pro psaní kódu, prostoru pro zprávy, textové konzole, lišty s příkazy pro běžné funkce a řadu nabídek. Připojuje se na hardware Arduina, nahrává programy a komunikuje s nimi.[6]

Tento editor je velice jednoduchý a poskytuje jednoduchou kontrolu a žádný stupeň nápovědy. Editor poskytuje velice jednoduchou správu knihoven a snadné přidávání desek jako je NodeMcu 1.0. Bohužel při testování většího počtu rozdílných desek editor při změně desky v jednom okně, změní desku ve všech dalších a znepřehledňuje situaci, proto jsem byl nucen přejít na editor větších rozměrů a komplexity.

1.2.2 Atmel Studio

Atmel Studio je nástroj na programování jednočipových procesorů. Nabízí prostředí, ve kterém se dá snadno orientovat. Programovací jazyk je C/C++ nebo Assembler. Je zadarmo ke stažení a na stránce http://www.atmel.com/ je více než 1600 hotových projektů také zdarma ke stažení. Atmel rovněž nabízí širokou škálu jednočipových procesorů různých velikostí a pamětí.[7]

Na rozdíl od Arduino IDE nepodporuje nativně Arduino knihovny, ale umí napovídat relevantní příkazy. Díky tomu usnadňuje práci. Absenci Arduino desek jsem vyřešil doinstalováním pluginu Visual Micro.

Visual Micro je tak zvané rozšíření pro Microsoft Visual Studio 2017 a 2019 a pro Atmel Studio 7, které umožňuje vyvíjet, kompilovat a následně libovolně rozšiřovat jakýkoli projekt Arduino na jakoukoli desku Arduino, přičemž využívá výhod výkonných funkcí Visual Studio a Atmel Studio.[8]

I po instalaci tohoto rozšíření je stále zapotřebí Arduino IDE, protože Visual Micro využívá jeho knihovny a desky. Toto rozšíření však není zadarmo.

1.2.3 Android Studio

Android Studio je oficiální integrované vývojové prostředí (IDE) pro vývoj aplikací pro Android, založené na IntelliJ IDEA. Kromě výkonných editorů a vývojářských nástrojů IntelliJ, nabízí Android Studio ještě více funkcí, které zvyšují vaši produktivitu při vytváření aplikací pro Android, jako rychlý emulátor bohatý na funkce, sjednocené prostředí, ve kterém můžete vyvíjet pro všechna zařízení Android, použít změny pro změnu kódu a zdrojů ve spuštěné aplikaci bez restartování aplikace [9]

Díky těmto vlastnostem je Android studio ideálním prostředkem pro vývoj celé škály aplikací. Celé prostředí je velice intuitivní a umožňuje změnu vzhledu podle našich představ.

1.3 Jednoduché vysvětlení principu

Všechny IOT moduly, co jsem vytvořil pro tento projekt fungují jako jednoduchý socket server. Jejich obslužný program byl napsán v jazyce C++. Všechny tyto servery mají maximální počet klientů. Moduly vždy vysílají data, co se změnila všem připojeným klientům. Senzory mají tu funkčnost, že odesílají jenom data. která se změnila a klient si o ně při úvodním příkazu řekl. Tímto principem nebude zahlcena síť zbytečnými hodnotami ze senzorů, které jsou stéle stejné. Každý senzor má možnost připojení k MYSQL serveru, kterému posílá SQL příkaz s vkládanými hodnotami. Řídící prvky mají možnost i z databáze číst v určitém časovém intervalu.

Aplikace byla vytvořena za pomocí Android Studia. Grafická část této aplikace je definována v XML souborech, zatímco obslužná část je psána v programovacím jazyce JAVA. Aplikace má plochu na, které lze přidat omezené množství prvků, protože při velkém počtu prvků se ikony začínají překrývat a jejich stisknutí je značně ztíženo. Každému prvku lze nastavit jméno, síťovou adresu a adresu stránky, ke které se budeme připojovat pro zjištění hodnost z databáze. Při nedostupnosti prvku v lokální síti aplikace pracuje jenom s klientem databáze.

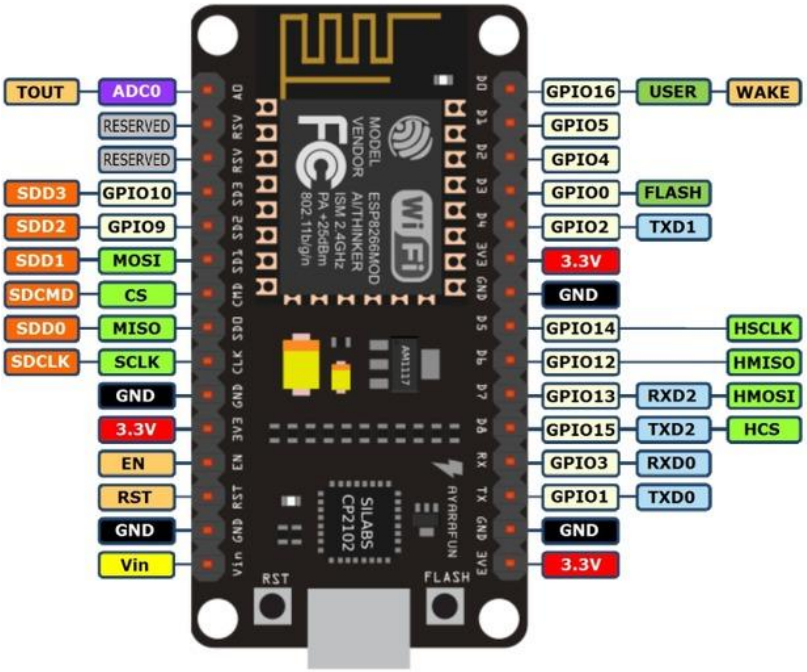
Webový server hostí MYSQL databázi. V databázi si modul vytvoří tabulku, která obsahuje pro něj potřebná pole. Na web mezi soubory přidáme PHP script, který funguje jako prostředník mezi telefonem a databází. Telefon zasílá přihlašovací údaje a časový úsek, ze kterého chce hodnoty ze senzorů. U modulů, které ovládají spínače systém funguje podobně. Telefon zašle přihlašovací údaje spolu s dotazem na aktuální hodnoty na spínačích nebo příkaz změnit určité hodnoty.

2 PRAKTICKÁ ČÁST

V této části probereme podrobně všechny programové části mojí práce.

2.1 Wi-Fi moduly

Pro jednoduchou přemístitelnost jsem použil Wi-Fi moduly, a to sice ESP 8266. Wi-Fi moduly mají oproti LPWAN prvkům velkou spotřebu energie, a proto nejsou vhodné pro venkovní použití. Venkovní senzory musí počítat s obtížným přístupem k elektrické zásuvce, a tak většinou pracují na baterku. Naproti tomu tu jsou domovní moduly, které mohou používat ethernetové připojení nebo bezdrátové připojení.



Obrázek 2 Ilustrační obrázek modulu

Využil jsem možnosti programovat toto zařízení za pomoci ARDUINO knihoven. Okolo těchto zařízení se pohybuje velké množství vývojářů, kteří svůj kód poskytli jako opensource. Což byl další důvod proč použít tuto platformu.

2.1.1 Základní konfigurace

Při prvním zapnutí je nutné zařízení nastavit některé údaje jako název, jméno Wi-Fi sítě, heslo k Wi-Fi síti atd., tyto údaje jsou po sléze uloženy pomocí simulované EEPROM paměti. Modul jako takový EEPROM paměť nemá, proto je zapotřebí ji simulovat ve FLASH paměti procesoru. Základní konfiguraci je zapotřebí provést přes sériový monitor s rychlostí přenosu 115 200 baudů. Všechny tyto údaje budou postupně použity. Nejprve je zapotřebí se připojit k síti. Program umožňuje i použití bez databáze pouze v lokální síti. Po zapnutí, načtení všech potřebných údajů od uživatele nebo z paměti a po úspěšném připojení k síti, se modul pokusí připojit k databázi a vytvoří tabulku, pokud již neexistuje. Pro připojení do databáze jsem použil knihovnu MySQL_Connector_Arduino, která je volně dostupná na github.com. Konfigurační sekvenci zahájíme zprávou „ahoj“.

```
switch (stavKomunikace) {
      case 1: {
          vycistiEepromku();
            for (byte i = 0; i <= n; i++)
              blaznivyNapad[jmeno][i]= pole[i];
          zapisDataDoEpromky(stavKomunikace, n);
          stavKomunikace++;
          Serial.println("zadej jmeno wifi site");
          oscanuj=1;
          break;
        }
```

V této ukázce kódu načítám jméno modulu a ukládám je do dvou rozměrného pole. Poté údaj zapíši do simulované EEPROM paměti, přepnu stav komunikace a nastavím příznak k oskenování dostupných Wi-Fi sítí. Takto načítám celkem 7 údajů, a to jméno modulu, jméno Wi-Fi sítě, heslo Wi-Fi sítě, jméno databáze, přihlašovací jméno k databázi, přihlašovací heslo k databázi a adresu databáze spolu s portem. Po skončení úvodní konfigurace modul přejde do pracovního režimu a připojí se k Wi-Fi.

```
    void pripojSa() {
      Serial.println(udaje[1]);
      Serial.println(udaje[2]);
      WiFi.begin(udaje[1], udaje[2]);//předám údaje funkci pro připo-
jení
      byte pocitadlo = 10;
      while (WiFi.status() != WL_CONNECTED && pocitadlo > 0) {
        delay(1000);
        Serial.println("Connecting..");
        pocitadlo--;
      }
      Serial.print("Connected to WiFi. IP:");
      Serial.println(WiFi.localIP());
    }
```

V této ukázce se modul pokouší připojit k Wi-Fi síti. Funkce *WiFi.begin(const char* jmenoSite, const char* hesloSite)* zahájí sekvenci připojení a vyšle první segment 4-way handshaku. Pokud jsou naše připojovací údaje správné, proběhne kompletní 4-way handshake. Naše čekací smyčka se ukončí, protože jsme se připojili a pomocí sériového kanálu vypíši naši přidělenou IP adresu. Pokud se modulu po 10 vteřinách nepodaří spojit s Wi-Fi access pointem vypíši prázdnou IP adresu a další pokus o připojení začne až za 5 minut. Po připojení k Wi-Fi síti se pokusíme připojit k databázi.

```
    const char zaloz[] = "USE %s; CREATE TABLE IF NOT EXISTS `%s` (
`id` BIGINT UNSIGNED NOT NULL AUTO_INCREMENT, `pohyb` BOOLEAN DEFAULT
NULL, `svetlo` MEDIUMINT UNSIGNED DEFAULT NULL, `cas` timestamp NOT
NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,PRIMARY
KEY (`id`)) ENGINE=MyISAM DEFAULT CHARSET=utf8;";

    void zalozTabulku() {
      Serial.println("Connecting...");
      if (conn.connect(server_addr, port, blaznivyNapad [PjmenoDB],
blaznivyNapad [PhesloDB])) {
        char query[360];// zprava co odešlu
```

```
        MySQL_Cursor *cur_mem = new MySQL_Cursor(&conn);

        // zapisu hodnoty do sablony query

        sprintf(query, zaloz, blaznivyNapad [jmenoDB], blaznivyNa-
pad[jmeno]);

        cur_mem->execute(query); // zapise do db

        delete cur_mem; //uvolni pamet

        Serial.println("Tabulka s nazvem: .");

        Serial.println( blaznivyNapad[jmeno]);

        Serial.println("byla zalozena");

      }

      else Serial.println("Connection failed.");

      conn.close();

    }
```

V této ukázce probíhá první připojení do databáze. Pokud se připojení úspěšně naváže, tak se do šablony SQL dotazu vloží jméno databáze, kterou používáme a jméno modulu. Jméno modulu slouží jako název tabulky, do které budeme vkládat řádky s údaji. Pokud tabulka již existuje, databáze nic neudělá.

2.1.2 Práce s klienty

Soket server jsem otevřel na portu č. 8888. Tento port jsem vybral, protože není mezi prvními 1023 dobře známými porty a nemá jej zaregistrovaný žádná významná služba.

```
    for (byte i = 0 ; i < pocetKlientu ; ++i) {
          if (clients[i]->connected()) {
            if (clients[i]->available() > 0) {
              byte n = clients[i]->available();
              for (byte ij = 0; ij < n; ij++) {
                inputs[i][ij] = clients[i]->read();
              }
              Serial.println(inputs[i]);  //čte přicházející zprávy
              zpracujPozadavek(inputs[i], i);}
```

Na modul se může maximálně připojit deset klientů. Při připojení nového klienta se přidá jeho záznam do pole. Tito klienti jsou postupně obsluhování jeden za čas. Pokud přišla od klienta zpráva, tak byla zapsána vnitřní funkcí do bufferu a funkce *available* nám vrací délku příchozí zprávy. Z tohoto bufferu čteme pomocí funkce *read* na instanci třídy *WifiClient*. Tuto zprávu poté předáme ke zpracování.

2.1.3 Funkce senzorů

Aby se zbytečně nezahlcovala síť stejnými hodnotami, tak je klient v roli odběratele, kdy si na začátku nebo kdykoliv v průběhu připojení zvolí, co si přeje odebírat za hodnoty. U teploměru jsou tyto hodnoty teplota [°C] a vlhkost [%], zatímco u senzoru pohybu jsou tyto hodnoty pohyb [1:0] a intenzitu osvětlení [lux]. Hodnoty jako intenzita osvětlení, teplota nebo vlhkost, jsou měřeny každých 5 sekund. Na pin PIR senzoru je nastaveno přerušení. Takto vždy, když se změní hodnota pohybu, tak se nastaví příznak, který zapříčiní odeslání nových hodnot všem klientům, kteří odebírají tuto hodnotu.

```
if(millis()-casmereni>=5000){
        casmereni=millis();
        float t1,v1;
         v1 = dht.readHumidity();
         t1 = dht.readTemperature();
         if( isnan(v1) || isnan(t1) )
         {
               // Don't do anything, if data is invalid
               Serial.println("DHT11 data in invalid");
         }
         else{
               if(v1!=v||t1!=t){
                     t=t1;
                     v=v1;
                     zmenaRvi();
               }
         }
      }
```

Do databáze odesílá data v pravidelném intervalu 5 minut. Při změně hodnot pohybu odesílá data častěji, kvůli zvýšení bezpečnosti. Všechna tato data jsou uložena v tabulce spolu s aktuálním časem, kdy byla naměřena.

```
    const char INSERT_DATA[] = "USE %s; INSERT INTO `%s` (zprava, vlhkost,
teplota) VALUES ('%s',%1.2f,%1.2f);";
```

```
void uloz() {
  Serial.println("Connecting...");
  if (conn.connect(server_addr, port, blaznivyNapad [PjmenoDB], blazni-
vyNapad [PhesloDB])) {
    char query[255];
    // zprava co odešlu
    MySQL_Cursor *cur_mem = new MySQL_Cursor(&conn);
    // nactu hodnoty
    float v1 = dht.readHumidity();
    float t1 = dht.readTemperature();
    if( isnan(v1) || isnan(t1) )
    {
      // Don't do anything, if data is invalid
      Serial.println("DHT11 data in invalid");
    }
    else
    {
            v=v1;
            t=t1;
      sprintf(query, INSERT_DATA,  blaznivyNapad [jmenoDB],blaznivyNa-
pad[jmeno],"test sensor", v, t);
      //Serial.println(query);
      // zapise do db
      cur_mem->execute(query);
      delete cur_mem;
      Serial.println("Data recorded.");
      }

  }
  else
    Serial.println("Connection failed.");
  conn.close();
    }
```

2.1.4 Funkce řídících modulů

Tyto moduly mají pod sebou nějaké spínací prvky jako relátka nebo některé tranzistory. U těchto modulů nefungují žádné role odběratele. Modul odesílá veškeré informace o spínačích, co se změnily všem připojeným klientům. Všechna tato data uchovává v tabulce v databázi. Tabulku kontroluje každou sekundu, aby bylo možné ovládat prvek i mimo dosah lokální sítě. Časování těchto procedur je napojeno na interní funkci *millis()*. Tato funkce má za sebou čítač, který počítá každou milisekundu od zapnutí až do maximální velikosti bezznaménkové 32bitové proměnné a při zavolání vrací aktuální počet milisekund.

```
    if (millis() - casek >= 1000) {

       casek = millis();

       zmeniloSeDb();   }
```

Z databáze se snažím odpojovat co nejméně a nepřipojovat se každou vteřinu. Tento postup zabírá více paměti, ale je správný z hlediska mnohem menšího síťového provozu. Pokud se v databázi něco změnilo, tak se nastaví příznak na odeslání dat síťovým klientům a náležitě so podle nového nastavení přenastaví piny procesoru.

Síťoví klient má možnost nastavit sérii podmínek, které se budou vyhodnocovat jednou za jím specifikovaným časovým intervalem. Tyto podmínky si mohou brát interní data jako je systémový čas nebo momentální nastavení pinů. Reálný čas modul získává z NTP serveru. Tento čas je uplynulá doba ode dne 1. 1. 1970 v sekundách. Čas se synchronizuje každou hodinu. Tyto podmínky se také mohou řídit pomocí dat načtených ze senzorů. Z každého senzoru můžeme číst maximálně dvě hodnoty. Podmínky se nastaví podle zprávy, která musí být v následujícím tvaru.

S192.168.1.1:8888[tep:vlh];192.168.1.112:8888[hyb]{v00>=v01?R1:0:c1000-P000XX11X:1}

‚S' značí počátek zadání podmínky. Poté následuje IP adresa a port zřízení, ze kterého bude odebírat hodnoty. Při připojení k tomuto zařízení použije příkaz, který je zapsán v hranatých závorkách. Jednotlivé zařízení jsou oddělena středníkem. Před složenou závorkou středník už být nemá. Za složenou závorkou začíná část deklarace podmínek. K datům přijatým z modulů přistupujeme pomocí písmene ‚v' první číslo za písmenem udává pořadí od počátku ‚S'. Druhé číslo udává pořadí hodnoty ve tvaru, v jakém přišly. Maximální počet hodnot čtení ze senzorů není více než dva.

Pokud ze senzorů číst nechceme tak namísto IP adres napíšeme ‚t'.

St{R1=1?R1:0:c1000-R1:1:c1000;c00-12-12-1-1-2020=t?R8:1:1-PXXX11XX1:c1000}

V takovém to případě můžeme využít vnitřních hodnot modulu. Stavy na pinech uvodíme písmenem ‚R' a poté pořadím. Aktuální čas uvodíme ‚t' a konstantní čas uvodíme malým ‚c'. Konstantní čas můžeme zadat ve dvou formátech. Jenom hodiny sekundy-minuty-hodiny nebo celé datum sekundy-minuty-hodiny-dny-měsíce-roky. Jako operátory podmínky používáme znaménka nerovnosti a to: „>, <, >=, <= ‚=, !".

Příkazy poté píšeme za otazník. Při kladném výsledku se vykoná hned první příkaz za otazníkem. Při záporném výsledku se použije příkaz za pomlčkou.

Příkaz pro ovládání pouze jednoho pinu začíná velkým ‚R‘, poté následuje číslo pinu, ke kterému přistupujeme. Za první dvojtečkou definujeme stav na pinu a za druhou dvojtečkou definujeme čas, za jak dlouho se daná podmínka znovu vykoná. Tento čas uvodíme písmenem ‚c‘ a za ním následuje čas v milisekundách. Pokud podmínku chceme vykonat v daném případě pouze jednou, za dvojtečku dosadíme pouze 1, tím se podmínka po vykonání příkazu odstraní ze seznamu.

Příkaz pro ovládání více pinů zaráz uvodíme velkým ‚P‘. Po něm následuje osm znaků, každý představuje jeden pin. Na pin můžeme zapsat 1 nebo 0, pokud pin měnit nechceme, zapíšeme velké ‚X‘.

Podmínek může být maximálně 20 a senzorů, ke kterým se připojuje maximálně 8. Celková délka jedné zprávy by neměla přesahovat 50 znaků. Pokud přesahuje tuto délku je potřeba ji rozdělit na menší úseky, přičemž první začíná normálně a každý další vzkaz uvozujeme „SP“. Dokud nedorazí ukončovací složená závorka, můžeme posílat další části podmínky. Celková délka jedné celé složené podmínky nesmí přesahovat 1000 znaků.

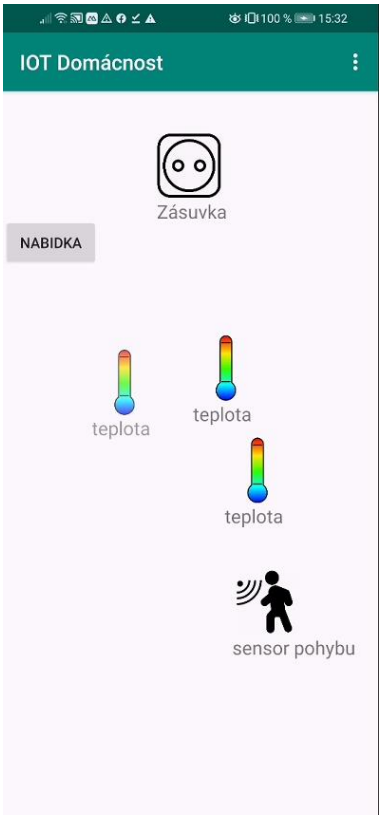
Samotné vyhodnocení podmínky probíhá za použití šablony funkce. Díky tomu nemusím psát více stejných funkcí pouze s jiným typy proměnných. Takto je funkce multifunkční na všechny typy proměnných, dokud mají definované operátory.

```
template<typename T>
bool rozhodovacka(const char* podminka, T prvniCastPodminky, T druhaCastPod-
minky) {
  bool a = false;
  switch (podminka[0]) {
    case '<': {
        if (podminka[1] == '=') {
          a = prvniCastPodminky <= druhaCastPodminky;
        }
        else {
          a = prvniCastPodminky < druhaCastPodminky;
        }
        break;
      }
    case '>': {
        if (podminka[1] == '=') {
```

```
          a = prvniCastPodminky >= druhaCastPodminky;
        }
        else {
          a = prvniCastPodminky > druhaCastPodminky;
        }
        break;

      }
    case '=': {
        a = prvniCastPodminky == druhaCastPodminky;
        break;

      }
    case '!': {
        a = prvniCastPodminky != druhaCastPodminky;

        break;
      }
  }
  return a;
    }
```

2.2 Mobilní aplikace

Mobilní aplikaci jsem tvořil s důrazem na snadnou modifikovatelnost vzhledu, kterou jsem docílil tím, že všechny prvky, které přidáme můžeme libovolně přesouvat. Plocha má dva režimy pracovní a editační. V editačním režimu má každý prvek na sobě metodu *onTouch*. Tato metoda zajišťuje vytvoření stínu prvku, který můžeme přetáhnout.



Obrázek 3 Hlavní aktivita

Pomocí tlačítka nabídka se nám zobrazí nabídka dostupných prvků, které můžeme dlouhým kliknutím vybrat a následně přetáhnout na námi zvolené místo. Při výběru tohoto prvku se vytvořila nová instance a základní konstrukce se uložila do databáze. Následná konfigurace prvku se nachází v nabídce, kterou vyvoláme podržením daného prvku a kliknutím na nastavení. Tím se spustí nová aktivita pro výsledek. Když se tato aktivita úspěšně ukončí, vrátí nám nastavení prvku uložené v proměnné typu *UlozCoPujde*. V následující ukázce je zobrazeno nastavení funkce pro vyvolání kontext menu, které obsahuje dvě položky nastavení vlastností a odstranit.

```
neco.setVolejNastaveniPrvku(new KliknutiPrvku() {
    @Override
    public void klikAkce(final Prvek coKlik) {
        final View v=coKlik.convertView;
        PopupMenu popup=new PopupMenu(v.getContext(),v);
        MenuInflater inflater =popup.getMenuInflater();
        inflater.inflate(R.menu.munu,popup.getMenu());
        popup.show();
        popup.setOnMenuItemClickListener(new PopupMenu.OnMenuItem-
ClickListener() {
            @Override
```

```
            public boolean onMenuItemClick(MenuItem item) {
                int id=item.getItemId();
                switch (id){
                    case R.id.vlastnosti:{
                        Intent funkcni=new Intent(MainActivity.this
                                ,vlastnosti.class);
                        funkcni.putExtra("Vlastnosti"
                                ,coKlik.getProsteVsecko());
                        MainActivity
                                .this
                                .startActivityForResult(funkcni
                                        ,NASTAVENI_DOPLNENOS);
                        break;

                    }
                    case R.id.odstranit:{
                        plocha.removeView(v);
                        prvke.remove(coKlik);
                        coKlik.deleteTask(coKlik.getProsteVsecko());

                        break;

                    }
                }
                return true;
            }
        });
    }
});
```

Při návratu z této aktivity se vyvolá metoda *onActivityResult*. Tato metoda pouze najde pomocí id prvku správný prvek a přiřadí mu nové aktualizované informace pomocí třídy *UlozCoPujde*.

2.2.1 Ukládání prvků do databáze za použití Room

Room poskytuje abstrakční vrstvu nad SQLite a přináší tak jednodušší přístup k databázi. Room obsahuje 3 hlavní komponenty. Za prvé je zapotřebí mít přístupový bod k databázi, což by měla být abstraktní třída, která dědí od třídy RoomDatabse. Tato třída by měla být anotována *@Database* a listem entit, které budou obsaženy v této databázi.

```
@Database(entities = {UlozCoPujde.class}, version = 1)
public abstract class mojeDatabase extends RoomDatabase {
    public abstract PrvkeDao taskDao();
}
```

Za druhé je zapotřebí definovat entitu. Entita reprezentuje tabulku v databázi.

V našem případě je tato entita označena *UlozCoPujde* a zároveň implementuje *Serializable*. Implementování *Serializable* umožňuje přenášení prvku v rámci *Intentu* do jiné aktivity. Tato třída by měla obsahovat pouze jednoduché datové typy. Všechny atributy této třídy mají definované gettery a settery. Jako primární klíč tabulky jsem použil id prvku.

```
@Entity
public class UlozCoPujde implements Serializable {
    @PrimaryKey(autoGenerate = true)
    private int id;
    @ColumnInfo(name = "popis")
    public String popis="volovina";
    @ColumnInfo(name = "typPrvku")
    public String typPrvku= "Teplomer";
    @ColumnInfo(name = "url")
    public String url="1.1.1.1";
    @ColumnInfo(name = "jmeno")
    public String jmeno="Teplomer";
    @ColumnInfo(name = "portak")
    public int port=80;
    @ColumnInfo(name = "souradniceX")
    protected float soradniceX=0;
    @ColumnInfo(name = "souradniceY")
    protected float soradniceY=0;
```

Za třetí objekt zpřístupňující data (DAO). Tento objekt obsahuje metody používané k zpřístupnění databáze. Tyto metody mohou být anotovány naším vlastním query nebo použijeme některé předdefinované.

```
@Dao
public interface PrvkeDao {
    @Query("SELECT * FROM ulozcopujde")
    List<UlozCoPujde> getAll();

    @Insert//(onConflict = OnConflictStrategy.REPLACE)
    void insert(UlozCoPujde prvek);

    @Delete
    void delete(UlozCoPujde prvek);

    @Update
    void update(UlozCoPujde prvek);

    @Query("DELETE FROM ulozcopujde")
    public void nukeTable();
}
```

Za použití těchto tří komponent můžeme potom snadno vkládat, upravovat a odebírat prvky z databáze. Všechny práce s databází nejsou doporučovány provádět na

hlavním vlákně, protože zabírají čas a aplikace by mohla zamrznout.

```
@Override
protected Void doInBackground(Void... voids) {
    DbClient.getInstance(nejakyContext).getAppDatabase()
            .taskDao()
            .delete(task);
    return null;
}
```

2.2.2 Třída Prvek

Třída prvek je abstraktní třída, která má za úkol sdružovat všechny ostatní prvky. Uvnitř této třídy nalezneme proměnou třídy *UlozCoPujde*. Tato proměnná sdružuje veškeré informace o tomto prvku. Pomocí této proměnné lze znovu vytvořit každý prvek. Pro vytvoření správné instance je na této třídě metoda. Zde v ukázce si ji můžeme prohlédnout.

```
static public Prvek zalozSpravnyPrvek(UlozCoPujde neco){
    Prvek novy=null;
    switch (neco.getTypPrvku()){
        case "Teplomer":{
            novy=new Teplomer(neco);
            break;
        }
        case "Zarovka":{
            novy=new Zarovka(neco);
            break;
        }
        case "Kamera":{
            novy=new Kamera(neco);
            break;
        }
        case "Zasuvka":{
            novy=new Zasuvka(neco);
            break;
        }
        case "SensorPS":{
            novy=new SensorPS(neco);
            break;
        }
    }
    return novy;
}
```

Na této třídě jsou také některé metody pro práci s prvkem v databázi jako

saveTask, *updateTask* nebo *deleteTask*. Tyto metody využívají asynchronního zpracování a běží mimo hlavní program.

```
protected void updateTask(final UlozCoPujde task) {


    class UpdateTask extends AsyncTask<Void, Void, Void> {

        @Override
        protected Void doInBackground(Void... voids) {
            DbClient.getInstance(nejakyContext).getAppDatabase()
                    .taskDao()
                    .update(task);
            return null;
        }

        @Override
        protected void onPostExecute(Void aVoid) {
            super.onPostExecute(aVoid);
            //Toast.makeText(nejakyContext, "Updated", Toast.LEN-
GTH_LONG).show();
        }
    }

    UpdateTask ut = new UpdateTask();
    ut.execute();
}
```

Tato třída také skrývá signatury metod *Fachej* a *uzNeFachej*. Tyto metody slouží jako funkční metody k zavolání v aktivitě *FunkcePrvku*. Zatímco metoda Fachej je volaná na začátku aktivity, tak metoda *uzNeFachej* je volána na konci pro ukončení funkcí, které běží na pozadí a při odchodu z aktivity je potřeba je ukončit zvlášť.

Při vkládání nového prvku využíváme funkci *dejmiNovyNahled*, která založí v databázi záznam a vrací nově vytvořené *View* prvku, které posléze vložíme na plochu. Pro definování vlastních akcí v *MainActivity* při dlouhém kliknutí a při stisku v editačním režimu. Třída Prvek také obsahuje dvě proměnné typu *KliknutiPrvku*. *Kliknutí prvku* je interface, který muže být v *MainActivity* implementován. Interface je objekt podobný třídě, ale jeho metody jsou abstraktní, aby třída, která tento interface implementuje, jej mohla přepsat.

2.2.3 Aktivita Vlastnosti

Této aktivitě je v *Intentu* předán *UlozCoPujde* prvku. Pomocí těchto dat je vytvořena správná instance určitého prvku. Z této instance se poté vezme layout a předá se metodě aktivity *setContentView*. Tato metoda vykreslí uživatelské rozhraní aktivity. Potom se zavolá přetížená metoda prvku *nastavSiVlastnosti*, která dostane jako parametr aktuální aktivitu. Díky přetížení tento postup mohu použít na všech prvcích.

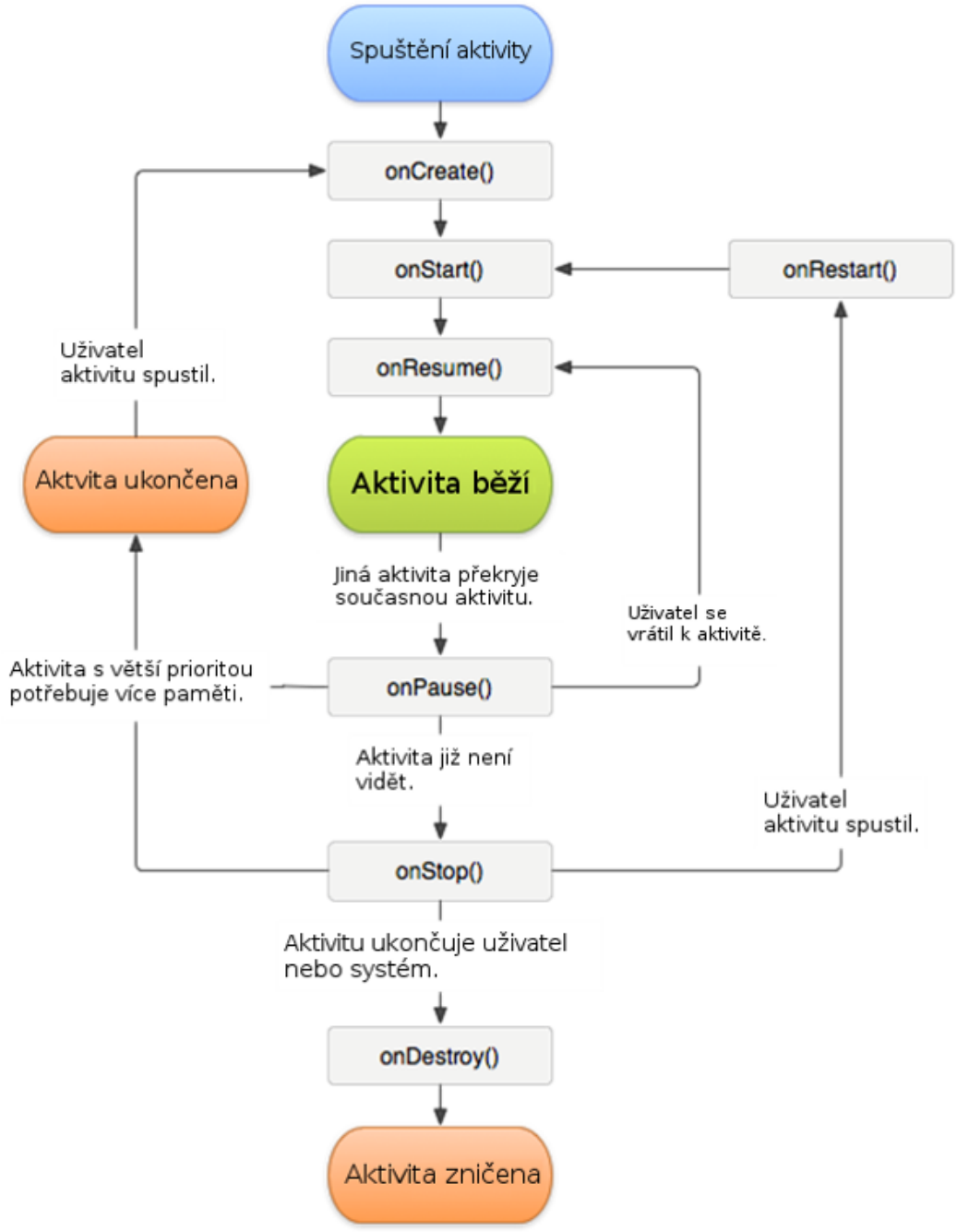
```
public class Vlastnosti extends AppCompatActivity {

    Prvek coNastavuji=null;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Intent jeTamNejaky=this.getIntent();
        UlozCoPujde nejaka=(UlozCoPujde) jeTamNejaky.getSerializa-
bleExtra("Vlastnosti");
        if(nejaka==null){
            Intent zpatky=new Intent(this,MainActivity.class);
            startActivity(zpatky);
        }
        coNastavuji=Prvek.zalozSpravnyPrvek(nejaka);
        setContentView(coNastavuji.getVlastnostiLayout());
        coNastavuji.nastavSiVlastnosti(this);
    }
    public void nacpyToTam(View v){
        coNastavuji.vemSiCoPotrebujes(this);
        Intent intent=new Intent();
        intent.putExtra("Vysledkos",coNastavuji.getProsteVsecko());
        setResult(RESULT_OK,intent);
        finish();
    }
}
```

V ukázce si můžeme povšimnout i funkce, která se zavolá v případě, že uživatel stiskne tlačítko uložit. Na prvku, který nastavujeme se zavolá metoda. Tato metoda ze všech polí načte informace a uloží je. Na konci této aktivity se nastaví výsledek s příslušným kódem, že aktivita proběhla v pořádku.

2.2.4 Aktivita Funkce prvku

Pracuje na podobném principu jako aktivita vlastností. Proces vytvoření instance je stejný, ale poté se hned nevolá metoda, která by zpracovala všechny prvky, které jsou na ploše. Tato metoda se volá až ve funkci *onStart*. Jelikož v této funkci budeme sledovat síťový provoz, na pozadí musíme všechny tyto funkce ukončit, když opustíme

aktivitu, abychom se vyhnuli nechtěným např. zprávám s novou teplotou.



Obrázek 4 Schéma životního cyklu aktivity

Jak vidíme ze schématu událostí, které nastanou od startu aktivity po její konec, vždy když aktivita přejde do pozadí, vyvolá se funkce *onStop* a pokud je aktivita následně obnovena, projde funkcí *onStart*. Zavolání metod prvku v těchto místech zajistí,

že posluchač na portu se může nastavit, vždy jen když je aktivita v popředí. V následující ukázce lze vidět přetížení metod *onStart* a *onStop*.

```
@Override
protected void onStart() {
    super.onStart();
    kteryLeti.fachej(this);
}

@Override
protected void onStop() {
    super.onStop();
    kteryLeti.uzNeFachej(this);
}
```

2.2.5 Třída Teploměr a SensorPS

Tyto třídy jsou potomky třídy Prvek. Tyto třídy mají také řadu společných vlastností. Nejsou řídící prvky a z dat v databázi vykreslují graf. Při kliku na některý z prvků tohoto typu se spustí funkční aktivita. V této aktivitě se zavolá metoda *fachej*, která založí instanci podtřídy *funkcnost*. Tato třída má na starosti veškeré akce spojené funkcemi v této aktivitě.

Obrázek 5 Funkční aktivita čidla

V horní části této aktivity můžeme vidět dvě pole, která se obnoví pokaždé, když dorazí data. Při vstupu do aktivity se spustí akce tlačítka pro připojení a po sekundě se spustí druhé tlačítko, které odesílá příkaz na zjištění hodnot ze senzoru.

Tyto akce jsou zajištěny dalšími podtřídami. Tyto třídy implementují *Runnable* takže mohou být spuštěny na dalším vlákně. *Thread1* založí nový soket, využije IP adresu prvku a jeho port. Dále vytvoří *BufferedReader* pro přijímání zpráv a *PrintWriter* pro odesílání. Při úspěšném spojení se spustí *Thread2*. *Thread2* je nekonečná smyčka, která čte příchozí zprávy a zobrazuje je na obrazovce. Aby bylo možné vypsání dat na hlavní obrazovku, musíme zase přejít na hlavní vlákno, to zajistíme funkcí *runOnUiThread*. Odesílání úvodního příkazu zajišťuje podtřída *Thread3*. *Thread3* vezme příkaz zapíše ho do *PrintWriter* a odešle ho.

```
class Thread3 implements Runnable {
    private String message;
    Thread3(String message) {
        this.message = message;
    }
    @Override
```

```
    public void run() {
        if(pripojen){
            output.write(message);
            output.flush();
            kdeToDelam.runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    Toast.makeText(kdeToDelam,
                       "poslána teplota",Toast.LENGTH_SHORT).show();
                }
            });
        }

    }
}
```

Pakliže je aktivita nějakou okolností přerušena, je zavolána metoda *uzNeFachej*. Tato metoda zajistí ukončení nekonečné smyčky v *Thread2*.

Pokud prvek má databázi, uvidíte tlačítka na výběr datumů a vykreslení grafu. Tlačítka na výběr datumů vyvolají dialog. Tento dialog má horní hranici momentu, kdy tento dialog vyvolal. Dialogy jsou nastavené tak, aby počáteční datum bylo vždy menší než datum konce časového období. Po nastavení časových údajů lze stisknout tlačítko vykreslení grafu. Stisknutím tlačítka se spustí nová aktivita.

2.2.6 Aktivita vykreslení grafu

Této aktivitě je předán Intent s polem údajů. V tomto poli se nachází url, přihlašovací jméno databáze, heslo databáze, počáteční datum a konečné datum. Tato aktivita po spuštění otevře asynchronní úkol, který odešle POST požadavek na server s přihlašovacími údaji a datumy. Po dokončení přijímání všech dat, se tato data vloží do liniového grafu a naformátují se osy. Pro účely této aplikace jsem využil grafovou knihovnu MPAndroidChart. Tato knihovna obsahuje velké množství typů grafů. Tato knihovna je zdarma, pokud se nechystáme o dynamický graf, který by se měnil v reálném čase. Následná aplikovatelnost grafu je velmi jednoduchá. Musíme nadefinovat závislosti na grafovou knihovnu v *build.gradle*. Dále musíme vložit grafovou komponentu do xml layoutu aktivity. Následující ukázka zobrazuje xml komponentu grafu.

```
<com.github.mikephil.charting.charts.LineChart
    android:id="@+id/graf"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Abychom mohli formátovat data na osách, musíme rozšířit třídu *ValueFormatter*. Tato třída zajišťuje defaultní formátování os. V defaultním nastavení vkládá pouze čísla. Pro zobrazení teploty v závislosti na čase, musíme vkládat na osu x časové údaje a na osu y teplotu nebo naše jiné kýžené hodnoty. Časové údaje jsou otočeny o 15°, aby nedocházelo k vzájemnému překrývání. V následující ukázce si můžeme prohlédnout rozšířenou třídu *ValueFormatter* pro formátování osy X.

```
class MyValueFormatter extends ValueFormatter {
    String hodnoty;
    public MyValueFormatter(String hodnoty){
        this.hodnoty=hodnoty;
    }

    @Override
    public String getAxisLabel(float value, AxisBase axis) {
        switch (hodnoty){
            case "teplota":{
                return String.format("%1.2f°C",value);
            }
            case "vlhkost":{
                return String.format("%1.0f%c",value,'%');
            }
            case "svetlo":{
                return String.format("%1.0flux",value);
            }
            case "pohyb":{
                return String.format("%s",value==1?"pohyb":"žádný
pohyb");
            }
        }
        return "nic";
    }
}
```

2.2.7 Třída Zásuvka

Třída *Zasuvka* je potomkem třídy Prvek. Je řídícím prvkem a pomocí dat v databázi lze řídit modul, který k ní je připojen. Při kliku na některý z prvků tohoto typu, se spustí funkční aktivita. V této aktivitě se zavolá metoda *fachej*, která založí instanci podtřídy *funkcnost*. Tato třída má na starosti veškeré akce spojené funkcemi v této aktivitě.

Obrázek 6 Funkční aktivita zásuvek

V aktivitě můžeme vidět 8 tlačítek. Tato tlačítka představují zásuvky. Tlačítko info odesílá příkaz pro zjištění aktuálního stavu zásuvek. Tato aktivita dokáže pracovat v režimech po lokální síti nebo přes databázi. Po příchodu do této aktivity se pokusí telefon připojit k modulu, pokud se spojení zdaří, je s modulem komunikováno přímo, pokud se spojení nezdaří, je s modulem komunikováno přes databázi. Komunikace po lokální síti probíhá podobně jako u teploměru. Třída obsahuje podtřídu funkcnost, která obsahuje podtřídy Thread1, Thread2 a Thread3, jejich funkčnost je stejná jako u třídy Teplomer, kde jsou náležitě popsány. Na obrázku si lze povšimnout tlačítka makra. Toto tlačítko otevře novou aktivitu s přehledem aktuálního makra. Toto makro lze odeslat pouze přes lokální síť.

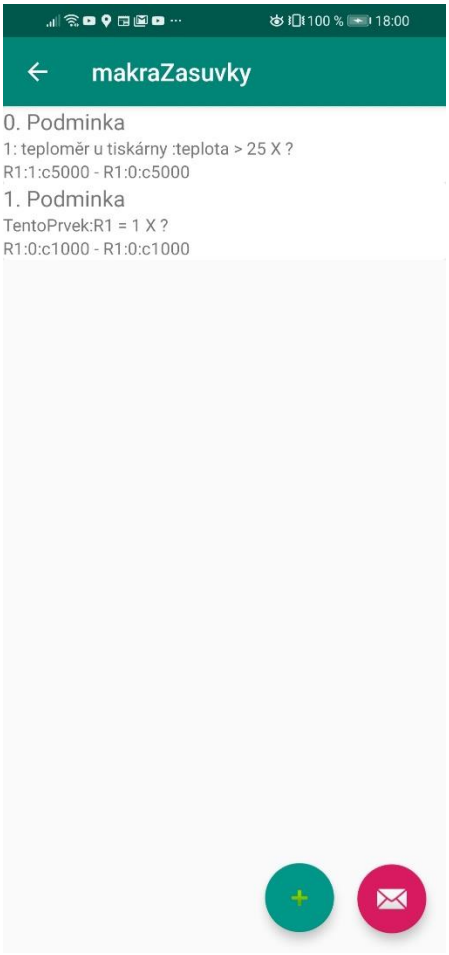
Při ovládání modulu přes databázi aplikace používá třídu *RelePresDb*. Tato třída dědí od *AsyncTask* a přetěžuje funkce *doInBackground* a *onPostExecute*. Funkce *doInBackground* je rozvětvena do dvou částí podle účelu buď získání aktuálního stavu z databáze nebo zapsání našeho stavu do databáze. V obou případech se naváže spojení s prostředníkem s databází a do POST požadavku se zakóduje jméno uživatele

v databázi, heslo k databázi a akce, kterou bude provádět. V případě zjištění stavu je to takhle plně dostačující požadavek. Tento požadavek je odeslán a ze serveru je získána odpověď, které je posléze předána funkci *onPostExecute*. V této funkci je formátovaná zpráva rozebrána a podle ní je nastavena barva tlačítek. Pokud ale měníme stav některého ze spínačů, tak se do POST požadavku také zapíší stavy tlačítek. Tento požadavek je poté odeslán PHP prostředníkovi s tím, že již neočekáváme žádnou odpověď. V ukázce si můžeme prohlédnout funkci, pomocí které zakódujeme hodnoty do POST požadavku.

```
httpURLConnection.setRequestMethod("POST");
String data = URLEncoder.encode("jmeno","UTF-8")+
        "="+URLEncoder.encode(login_name,"UTF-8")+"&"+
        URLEncoder.encode("heslo","UTF-8")+
        "="+URLEncoder.encode(login_pass,"UTF-8")+"&"+
        URLEncoder.encode("akce","UTF-8")+
        "="+URLEncoder.encode(strings[0],"UTF-8");
```

2.2.8 Aktivita Makra zásuvky

Tato aktivita zobrazuje nastavené podmínky na modulu. Při příchodu do této aktivity, telefon pošle žádost o vypsání stávajícího makra. Telefon příchozí zprávu rozdělí na jednotlivé podmínky a pomocí komponenty *recyclerView* zobrazí.



Obrázek 7 Aktivita maker

Pokud chceme s podmínkami dále pracovat, například použít jednu jako vzor, můžeme dlouze podržet kartu s touto podmínkou, když tak učiníme, zobrazí se nám kontextové menu, které nám dává na výběr mezi dvěmi možnostmi smazat a kopírovat. V pravém dolním rohu můžeme vidět dvě tlačítka. Zelené tlačítko slouží k přidání zcela nové podmínky, zatímco růžové tlačítko slouží k odeslání makra do modulu. Jestliže chceme upravit již existující podmínku stiskneme kartu podmínky, které vyvolá aktivitu s touto podmínkou.

Obrázek 8 Aktivita tvorby podmínky

Řádky podmínky se dynamicky přidávají. Mezi jednotlivými stranami podmínky máme na výběr ze znamének nerovnosti. Na konci řádku můžeme vybrat logický operátor, pokud chceme řádek přidat. Pokud nechceme další řádek nebo od určitého řádku chceme odstranit všechny další, zvolíme znak X. Na jednotlivých stranách podmínky je komponenta *autoCompleteTextView*. Tato komponenta po napsání prvního znaku napovídá možné komponenty. Mezi nápovědami jsou všechny neřídící komponenty a proměnné na samotném řídícím modulu. Dále zde také můžeme nastavit, co se má stát v případě splnění či nesplnění podmínky. Po dokončení podmínky ji můžeme uložit nebo zahodit.

Při vytvoření všech kýžených podmínek, můžeme stisknout tlačítko pro odeslání makra. Před odesláním se musí znovu vytvořit formátovaná zpráva makra. Tato zpráva se vytvoří jako asynchronní úkol. Nejprve se vyčtou z databáze prvky, které byly použity, poté se za tyto prvky dosadí jejich IP adresa a port, spolu s příkazem pro zjištění

potřebných hodnot a poté se dosadí do podmínek odkaz na tyto hodnoty. Výsledkem je formátovaná zpráva, která se rozdělí podle maximální délky zprávy na modul.

2.3 PHP webový prostředník

Jsou to webové PHP soubor, které nám umožňují přístup k databázi. Při spuštění těchto souborů není výstupem běžné html, ale pouze formátovaný text. Tyto soubory reagují na specifické POST požadavky.

2.3.1 Čtení hodnot čidel

Tento soubor je sestaven tak, aby bylo snadné jej modifikovat pro jakýkoli senzor. Jediné, co je nutné změnit je jméno databáze, ke které se připojujeme a jméno tabulky. Po změnách těchto veličin stačí do aplikace zadat odkaz na tento soubor.

```
if($_POST){
            require_once('Db.php');
            Db::connect('localhost', 'prihlaseni', $_POST['jmeno'],
                $_POST['heslo']);
            $coChce=$_POST['hodnota'];
            $zacatek="SELECT `id`,`$coChce`,`cas` FROM `to ti
                nereknu` WHERE `cas` BETWEEN '";
            $od = new DateTime($_POST['od']);
            $do = new DateTime($_POST['do']);
            $rozdil=date_diff( $do,$od, TRUE );
            $dny= $rozdil->format('%a');
```

Pro připojení do databáze používám wrapper. Tento wrapper je takový obal, který usnadňuje práci s databází. Jak vidíme z ukázky, pro úspěšné načtení dat je nutné v POST požadavku odeslat jméno a heslo pro přístup do databáze a také datum, od jakého data, k jakému datu budeme číst údaje z tabulky. Čas uchovávám pomocí třídy DateTime. Tato třída dokáže čas formátovat a přičítat k němu libovolné časové úseky. Moduly odesílají průměrně každých 5 minut řádky do tabulky, což činí za den něco přes 250 záznamů. Abychom nepřenášeli z databáze více jak 1000 řádků, při časovém úseku delším více než 3 dny je načítání údajů omezeno. Pokud je úsek do tří dnů délky,

jsou poslány všechny řádky v tomto čase. Pokud je úsek od 3 dnů po 7 dnů, tak se čte z každé hodiny minimum a maximum, a jestliže je to více než sedm dnů, tak se bere z každého dne minimum a maximum. Tímto způsobem by to zabralo časový úsek delší než jeden a půl roku, aby počet všech řádků byl větší než 1000. Každý jeden řádek je vypsán ve tvaru:“ id,hodnota,datum;“.

2.3.2 Čtení a zápis stavů pinů

Tento soubor je určen především ke zjištění aktuálního stavu a upravení těchto stavů. Tabulka, se kterou tento soubor pracuje, obsahuje pouze jeden řádek se stavy všech pinů. Při čtení tabulky vypíše se nám řádek podobný následující ukázce.

R1:0;R2:1;R3:1;R4:0;R5:0;R6:1;R7:1;R8:0

Můžeme si všimnout, že na konci není středník, aby při rozdělení podle něj se nevytvořilo na konci prázdné pole. Pro připojení k databázi zase využívám wrapperu.

Při změně pinů můžeme měnit jakékoli piny v jakémkoli počtu. Skript změní jenom ty, které jsme nastavili. Ty, co nechceme měnit, nastavíme v POST požadavku na N.

```
if($_POST['akce']=="update"){
    $sql= "UPDATE `lidskypocitac` SET ";
    if(isset($_POST['R1'])&&strcmp($_POST['R1'],"N"))
      $sql=$sql."`R1`=" .$_POST['R1'].", ";
    if(isset($_POST['R2'])&&strcmp($_POST['R2'],"N"))
      $sql=$sql."`R2`=" .$_POST['R2'].", ";
    if(isset($_POST['R3'])&&strcmp($_POST['R3'],"N"))
      $sql=$sql."`R3`=" .$_POST['R3'].", ";
    if(isset($_POST['R4'])&&strcmp($_POST['R4'],"N"))
      $sql=$sql."`R4`=" .$_POST['R4'].", ";
    if(isset($_POST['R5'])&&strcmp($_POST['R5'],"N"))
      $sql=$sql."`R5`=" .$_POST['R5'].", ";
    if(isset($_POST['R6'])&&strcmp($_POST['R6'],"N"))
      $sql=$sql."`R6`=" .$_POST['R6'].", ";
    if(isset($_POST['R7'])&&strcmp($_POST['R7'],"N"))
      $sql=$sql."`R7`=" .$_POST['R7'].", ";
    if(isset($_POST['R8'])&&strcmp($_POST['R8'],"N"))
      $sql=$sql."`R8`=" .$_POST['R8'].", ";
    $sql=$sql." `casUpravy`=CURRENT_TIMESTAMP WHERE `id`=1;";
    Db::query($sql);
```

ZÁVĚR

Cílem tohoto projektu bylo vytvořit jednoduché prostředí na ovládání IOT prvků. Aplikaci jsem vytvořil a je plně funkční. Projekt jako celek byl velice rozsáhlý a celkem povedený. Pouze mě mrzí, že jsem neměl více času na vytvoření více modulů, které bych mohl ovládat. Všechny prvky byli zkonstruovány poměrně rychle a občas bylo obtížné držet si odstup a vidět věci jako celek. Také bych příště nevolil nákup všech součástek z Číny. Párkrát mě zaskočilo jejich chování a projekt se díky tomu dokázal zaseknout. Jelikož jsem dělal projekt ze své vlastní iniciativy, tak jsem velmi spokojen s výsledným provedením zásuvek. Práce na tomto projektu byla místy velice obtížná. Zkoumání pro mě doposud neznámých oblastí, bylo velkou výzvou a velkým přínosem. Aplikaci bych chtěl ještě rozšířit o přihlášení a rozdělení prostoru do místností. Využití tohoto projektu v domácnosti bude pro mou maličkost velkým přínosem, jelikož jsem již dlouho snil o tom, jak budu moci ovládat věci doma, i když doma nebudu a celkově to bude znamenat méně chození, což považuji za výhodu.

SEZNAM LITERATURY

[1] RACEK, David. Programovací jazyky Java, C Sharp a C++: důležité informace a využití. *JAVA portál* [online]. Česká republika: amaio, 2017 [cit. 2020-03-18]. Dostupné z: http://www.jug.cz/article/java-42

[2] PHP. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-01-17]. Dostupné z: https://cs.wikipedia.org/wiki/PHP

[3] MySQL. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-01-18]. Dostupné z: https://cs.wikipedia.org/wiki/MySQL

[4] What Is SQLite? *SQLite* [online]. SQLite, 2010 [cit. 2020-03-18]. Dostupné z: https://www.sqlite.org/index.html

[5] Úvod do SQL. *Sylaby, elektronické učebnice* [online]. Česká Republika: VŠB, 2006 [cit. 2020-03-18]. Dostupné z: http://books.fs.vsb.cz/sqlreference/sadovski/sql-prvn.htm

[6] ARDUINO IDE. *ARDUINO.CZ* [online]. Česká Republika: hwkitchen, 2014 [cit. 2020-03-18]. Dostupné z: https://arduino.cz/arduino-ide/

[7] Co to je Atmel Studio? *Atmel Studio* [online]. Česká Republika: eStránky, 2019 [cit. 2020-03-18]. Dostupné z: https://atmelstudio.estranky.cz/clanky/co-to-je-atmel-studio-.html

[8] What is Visual Micro? *Visual Micro* [online]. England: Visualmicro, 2019 [cit. 2020-03-18]. Dostupné z: https://www.visualmicro.com/page/User-Guide.aspx?doc=what_is_visual_micro.html

[9] Meet Android Studio. *Android Studio* [online]. [cit. 2020-03-18]. Dostupné z: https://developer.android.com/studio/intro

SEZNAM OBRÁZKŮ