
Dokumentace k praktické části

Diplomové práce

Šec David

2016

Anotace

Představa propojit většinu existujících systémů tak, aby poskytovaly uživateli komplexní přehled o všech situacích a umožnit mu tak provádět nejlepší možná rozhodnutí, ať už v interakci s uživatelem, nebo bez jeho přímého zásahu je snahou mnoha velkých firem napříč platformami.

V řídicí technice je patrný výrazný trend přechodu od výkonných centrálních systémů k menším, které jsou rozmístěny podle potřeby a propojeny komunikační sítěmi. Odpadá tak nutnost vést všechny signály do jednoho řídicího bodu. Řízení je tak distribuováno přímo do jednotlivých bodů a ty následně vykonávají vybrané akce. Takto navrženým systémům se říká distribuované řídicí systémy. Jejich využití najdeme v mnoha průmyslových aplikacích IIoT (Industrial Internet of Things), ale také v budovách, automobilech a mnoha dalších.

Obsah

1	Specifikace požadavků na systém	3
2	Specifikace požadavků na software	3
2.1	Minimální systémové požadavky	3
3	Specifikace architektury a software	4
4	Specifikace návrhu software.....	5
4.1	Použitá rozhraní	5
4.2	Databáze.....	7
4.3	Symmetrical DS	8
4.4	Uzly	9
4.4.1	Získávání dat ze senzorů	9
4.4.2	Parsování dat ze služby Openweathermap	11
4.4.3	Změna stavu aktuátoru	11
4.4.4	Pravidla.....	12
4.4.5	Quartz Scheduler.....	12
4.4.6	Socket komunikace - Server	13
4.5	Webové rozhraní.....	14
4.5.1	Entity	14
4.5.2	Socket komunikace – Klient.....	15
4.5.3	Design.....	15
5	Specifikace testování software.....	17

1 Specifikace požadavků na systém

Cílem práce je navrhnout takový systém, který bude snadné integrovat i do stávajících staveb a to s cílem minimálního zásahu do stávajících domovních rozvodů. Systém musí být schopen snímat hodnoty v různých částech domu, zároveň musí umožnit ovládat vybrané elektrické spotřebiče včetně domovního osvětlení. Systém dále musí pracovat tak, aby se daly určité akce automatizovat bez zásahu uživatele. Systém bude rovněž schopný si určité hodnoty zajistit z veřejně dostupných internetových služeb. Jedná se zejména o údaje, které jsou v domácích podmínkách jen obtížně měřitelné, nebo se jedná o předpovědi pro nadcházející období. Tyto údaje mohou být následně využity pro interní potřeby systému a přispět k vyhodnocování určité akce. Uživateli musí být umožněno měnit vybrané parametry (např. požadovaná teplota v místnosti), nebo spínat určité okruhy manuálně prostřednictvím počítače, tabletu, případně mobilní aplikace.

Klíčové požadavky na projekt jsou identifikovány níže:

- Sběr dat ze senzorů
- Zobrazení hodnot pomocí web. aplikace
- Reagovat na mimořádné situace
- Podávat hlášení
- Automatizace vybraných úloh
- Dlouhodobé statistiky
- Možnost uchování a reprezentace dat za určité období.
- Snadná realizace

2 Specifikace požadavků na software

2.1 Minimální systémové požadavky

- tzv. detailní analýza,
- definuje úplný soubor požadavků na SW (i na HW),

Systém je navržen tak, aby byl nezávislý na použitém hardware. Jednotlivé uzly mohou využívat odlišný hardware splňující následující minimální požadavky.

- Operační systém Linux, nebo Windows
- CPU a alespoň 512 MB RAM
- GPIO sběrnice s digitálními vstupy a výstupy

- Připojení k internetu

Vytvořený software je navržen modulárně, tak aby umožňoval snadnou pozdější rozšiřitelnost o další moduly.

3 Specifikace architektury a software

Každou jednotku bude tvořit jedno zařízení Raspberry Pi 2 model B. Pro připojení k síti je zde připraveno rozhraní RJ45. Zároveň umožňuje připojení řady periférií prostřednictvím GPIO sběrnice. Dále obsahuje 4 USB porty vhodné pro připojení dalších periférií a integrovanou síťovou kartu pro připojení k síti internet.

V projektu budou použity celkem tři tyto jednotky. Jedna z jednotek bude označena jako master – hlavní uzel a ostatní jako slave – vedlejší uzly. Na master uzlu poběží webový server, který poslouží pro zobrazení naměřených hodnot, poskytne přehled o jednotlivých elektrických okruzích. Dále bude zpřístupňovat veškeré hodnoty pro další aplikace ve formátu JSON. O vizualizaci dat se postará HTML5 v kombinaci s Java skriptem.

Každý uzel bude periodicky v zadaných intervalech snímat fyzicky připojené senzory. Veškeré hodnoty budou uchovány v databázi, odkud mohou být v případě potřeby získány. Tyto hodnoty se budou následně distribuovat mezi ostatními uzly tak, aby bylo možné efektivně získat přehled o všech senzorech v domácnosti.

Ke snímání budou použity převážně digitální senzory, které mohou být připojeny k libovolnému uzlu prostřednictvím vestavěné GPIO sběrnice. V místnostech budeme bezprostředně měřit pouze teplotu a vlhkost. K tomu nám poslouží senzor teploty DS18B20 a senzor teploty a vlhkosti DHT11. Akční člen byl zvolen relay board HL-85.

Jako zástupce dat, která získávaných z rozhraní třetích stran byly zvoleny předpověď počasí a rychlost větru. Obě hodnoty budou čteny ze služby openweathermap.org ve formátu JSON.

Detailní informace o jednotlivých senzorech a aktuátorech, včetně zdůvodnění jejich použití jsou součástí praktické části diplomové práce.

4 Specifikace návrhu software

Aplikace pro jednotlivé uzly včetně webového rozhraní jsou naprogramovány v jazyce Java. Veškeré informace o návrhu architektury jsou popsány v praktické části diplomové práce.

V následující kapitole budou prezentovány pouze vybrané části kódu. Kompletní zdrojová dokumentace je dostupná v adresáři `\build\docs\javadoc`.

4.1 Použitá rozhraní

System musí umožňovat komunikaci s mobilními aplikacemi. Za tímto účelem bylo vytvořeno REST API. Aplikace mohou využít následující rozhraní:

GET [/node/?format=json](#)

Fields:

"id" : int	- jednoznačný identifikátor uzlu
"ip" : String	- ip adresa uzlu
"name" : String	- název uzlu

Example:

```
[
  {
    "id": 1,
    "ip": "192.168.0.161",
    "name": "node01"
  }
]
```

GET [/sensor/?format=json](#)

Fields:

"id" : int	- ID senzoru
"description" : String	- popis senzoru
"identifier" : String	- jednoznačný identifikátor senzoru
"name" : String	- název senzoru

Example:

```
[
  {
    "id": 1,
    "description": "teplota procesoru uzlu 1",
    "identifier": "cpu01",
    "name": "cpu_node01"
  }
]
```

GET [/value/?format=json](#)

Fields:

"id" : int	- ID hodnoty
"date" : Timestamp	- datum a čas pořízení hodnoty
"value" : float	- naměřená hodnota
"sensor" : Sensor	- Senzor (viz výše)

Example:

```
[
  {
    "id": 97872,
    "date": 1471273202000,
    "value": 23.0,
    "sensor": {
      "id": 5,
      "description": "Senzor DHT11 teplota",
      "identifier": "dht11_01_temp",
      "name": "dht11_temperature"
    }
  }
]
```

GET [/value/sensor/?sensor="identifier"](#)

Fields:

"id" : int	- ID hodnoty
"date" : Timestamp	- datum a čas pořízení hodnoty
"value" : float	- naměřená hodnota
"sensor" : Sensor	- Senzor (viz výše)

Example: [/value/sensor/?sensor=ds_garaz](#)

```
[
  {
    "id": 97875,
    "date": 1471273800000,
    "value": 29.062,
    "sensor": {
      "id": 8,
      "description": "DS18B20 v garáži",
      "identifier": "ds_garaz",
      "name": "ds_garaz"
    }
  }
]
```

GET /relay/?format=json

Fields:

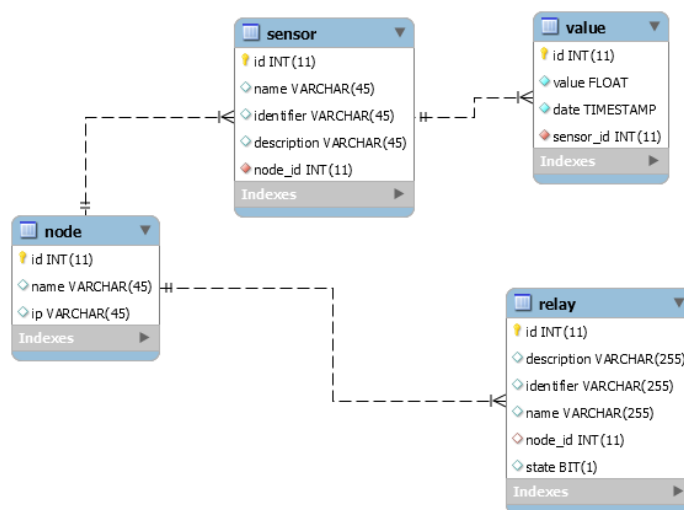
"id" : int	- ID hodnoty
"description" : String	- popis
"identifier" : String	- jednoznačný identifikátor
"name" : String	- název relay
"state" : boolean	- aktuální stav sepnutí senzoru

Example:

```
[
  {
    "id": 1,
    "description": "relayBorad01_1",
    "identifier": "rb_01_1",
    "name": "stresni_okna",
    "state": 1
  }
]
```

4.2 Databáze

Jako databáze byla zvolena MySQL databáze ve verzi 5.5.44. Schéma databáze bez SYM_ záznamů je zobrazeno na Obrázek 1.



Obrázek 1: Schéma databáze

Nástroj Symmetrical DS však neumožňuje synchronizaci sekvencí mezi jednotlivými uzly. Pro zamezení nechtěnému přepisování hodnot následkem shodného ID záznamu musely být vytvořeny sekvence pro každý z uzlů a následně za pomoci procedury hodnota sekvence získána.

Ukázka jedné z procedur pro ukládání jednotlivých hodnot je zobrazena níže.

```
CREATE
  PROCEDURE `insertValuetetoNode1` (IN cur_value INT , IN sensor_id IN)
BEGIN
  INSERT INTO test.value (`id`, `value`, `date`, `sensor_id` )
  VALUES (
    (SELECT value_id_node1 ()), cur_value,
    (SELECT now (), sensor_id
  ) ;
END
```

value_id_node1 odkazuje na sekvenci. Ta je po každém záznamu inkrementována o 1. Počáteční hodnoty jednotlivých sekvencí jsou 1 pro první uzel. Pro každý další uzel je tato hodnota o 600 000 vyšší, než v předchozím uzlu. Každý z uzlů může uložit až 600 000 hodnot. To je při frekvenci 1 záznamu za 10 minut a deseti senzorů na uzel postatečný prostor pro ukládání hodnot za dobu delší, než jeden rok.

4.3 Symmentrical DS

Jedná se o versvu zajišťující synchronizaci dat. Po stažení aktuální verze je nutné nastavit způsob synchronizace a přístup k jednotlivým databázím. To provedeme vytvořením souboru *Název-Nodu.properties* v adresáři *engines*.

Ukázka souboru *master-1.properties* je zobrazena níže:

```
external.id = 1
engine.name = master - 1
auto.config.registration.svr.sql.script = /symmetric-profile-master-2-
master-conf$
sync.url=http\:/ / 192.168.0.161\ : 31415 / sync / master - 1
group.id = master
db.url = jdbc\ : mysql\ :
//192.168.0.161/test?tinyInt1isBit\ =false&zeroDateTimeBehav$
db.driver = com.mysql.jdbc.Driver
db.user = nodeuser
registration.url =
db.validation.query = select 1
db.init.sql =
db.password = enc\ : p + uhOGpKznq7YqUihw8WeQ\ = \ =
db.connection.properties =
```


4.4 Uzly

4.4.1 Získávání dat ze senzorů

Pro získání dat ze senzorů bylo definováno rozhraní *ISensor* s metodou *loadData()*. Následně byly vytvořeny třídy *SensorDS18B20.java* a *DHT11.java* implementující toto rozhraní. Zdrojové kódy níže popisují způsob získání dat ze senzorů.

Metoda *loadData()* třídy *SensorDS18B20.java*

```
/**
 * DS18B20 temperature sensor.
 * Load data from sensor using read line with temperature after "t="
 * from file specified in sensrFile Constrictor.
 * Readed value is divided by 1000.
 * return Number value [in °C];
 */
public Number loadData() {
    Number val = null;
    try (BufferedReader reader = new BufferedReader(new FileReader(
        valueFile))) {
        String tmp = reader.readLine();
        int index = -1;
        while (tmp != null) {
            index = tmp.indexOf("t=");
            if (index >= 0) {
                break;
            }
            tmp = reader.readLine();
        }
        if (index < 0) {
            throw new IOException("Could not read sensor data");
        }
        val = Integer.parseInt(tmp.substring(index + 2)) / 1000f;
    } catch (IOException ex) {

        Logger.getLogger(SensorDS18B20.class.getName()).log(Level.SEVERE, null,
            ex);
    }
    setValue(val);
    return value;
}
```

Metoda *loadData()* třídy *DHT11.java*

```
/**
 * DHT11 temperature and humidity sensor.
 * Load data from sensor using python script which return two values.
 * return Number value [in °C];
 * sensor : 11 for DTH 11 or 22 for DHT22 sensor
 * gpinPin : GPIO pin in which is sensor connected (use GPIO number
 * no pin number)
```

```

*/

public Number loadData() {
    String cmd = "sudo python /home/pi/AdafruitDHT.py " + sensor + " "
                + gpioPin;
    String ret = "";
    String output = "";

    try {
        String line;
        Process p = Runtime.getRuntime().exec(cmd.split(" "));
        p.waitFor();
        try (BufferedReader input = new BufferedReader(new
            InputStreamReader(p.getInputStream()))) {
            while ((line = input.readLine()) != null) {
                output += (line + '\n');
            }
        }
        System.out.println(output);

    } catch (IOException | InterruptedException e) {}

    parseValue(output);
    return this.temperature;
}

/**
 * Split and trim outut string from loadData() to two Numbers -
 * Temaperature and Humidity.
 * First value is temperature in °C, second humidity in %.
 * Trim string is "   " (three spaces) Could be changed in
 * python script.
 */

public void parseValue(String ret) {
    //ret.trim();
    if (ret.length() == 0) // Library is not present
    {
        throw new RuntimeException("LIB_NOT_PRESENT_MESSAGE");
    } // Error reading the the sensor, maybe is not connected.
    // Read completed. Parse and update the values
    String[] vals = ret.split("   ");
    setTemperature(Float.parseFloat(vals[0].trim()));
    setHumidity(Float.parseFloat(vals[1].trim()));
}

```

Z důvodu absence knihovny pro obsluhu DHT senzoru v jazyce Java, bylo přistoupeno k volání skriptu v programovacím jazyce Python, z jehož výstupu jsou následně hodnoty přečteny. Původní záměr použít knihovnu PI4J, pro obsluhu GPIO pinů se bohužel ukázal jako nereálný, protože tato knihovna umožňuje pouze přepínat GPIO piny do stavů logické LOW, nebo HIGH (což by sice stačilo pro inicializaci měření), ale již neumožňuje přečíst zaslanou hodnotu ze senzoru.

4.4.2 Parsování dat ze služby Openweathermap

Pro získávání dat o počasí byla použita knihovna Gson.

Ukázka parsování dat z Openweathermap API. Metoda *parseJSON()*, ve třídě *OpenWeatherMapParser*.

```
/**
 * Parse Json String to specified Objects
 * @params root element
 * @return OpenWeatherMap
 */
private OpenWeatherMap parseJSON(JsonElement root) throws
JsonSyntaxException {
    OpenWeatherMap ow;
    JsonObject rootobj = root.getAsJsonObject();
    JsonArray jarray = rootobj.getAsJsonArray("list");
    JsonObject listObject = jarray.get(0).getAsJsonObject();
    String name = listObject.get("name").toString();
    JsonObject mo = jarray.get(0).getAsJsonObject();

    OpenWeatherMapTemp temp = new
Gson().fromJson(mo.getAsJsonObject("main"),
    OpenWeatherMapTemp.class);
    OpenWeatherMapWind wind = new
Gson().fromJson(mo.getAsJsonObject("wind"),
    OpenWeatherMapWind.class);
    OpenWeatherMapSky sky = new
Gson().fromJson(mo.getAsJsonArray("weather").get(0).getAsJsonObject(),
    OpenWeatherMapSky.class);

    ow = new OpenWeatherMap(name, temp, wind, sky);
    return ow;
}
```

4.4.3 Změna stavu aktuátoru

Třída *ControllGPIO.java*

```
/*
 * Execute python script to switch on/off relay
 * @params: name - Python script name
 */
private void execCommand(String name) {
    String cmd = "python " + SCRIPTS_PATH + name;
    String ret = "";
    String output = "";

    try {
        String line;
        Process p = Runtime.getRuntime().exec(cmd.split(" "));
        p.waitFor();
        try (BufferedReader input = new BufferedReader(new
InputStreamReader(p.getInputStream()))) {
            while ((line = input.readLine()) != null) {

```

```

        output += (line + '\n');
    }
}
System.out.println(output);
} catch (IOException | InterruptedException e) {}
}

```

4.4.4 Pravidla

Systém umožňuje definovat jednoduchá pravidla. Systém následně automaticky kontroluje splnění podmínek a na jejich základě vykonává zadanou akci.

Příkladem může být pravidlo pro automatické otevírání oken v případě, že je jasné počasí a venkovní teplota přesahuje 20°C.

```

public boolean canOpenWindow() {
    if (sky.main.equals("Clear") && temperature.temp_min >= 20) {
        return true;
    } else {
        return false;
    }
}

```

Následně je podmínka v pravidelných intervalech ověřována, případně vykonána zvolená akce.

```

if (wm.canOpenWindow()) {
    windowRelay.switchOn();
} else {
    windowRelay.switchOff();
}

```

4.4.5 Quartz Scheduler

Služba Quartz Scheduler je ideální volbou pro automaticky spouštěné úlohy v předem naplánovaných časových intervalech.

Quartz scheduler pracuje s následujícími třídami:

- *Scheduler* – plánovací služba
- *JobDetail* – údaje o úloze, ve kterých se nachází hlavně vlastní třída úlohy; instance se vytváří pomocí třídy *JobBuilder*
- *Trigger* – mechanismus úlohy, který specifikuje, kdy se má úloha spouštět

Detailní informace lze nalézt v dokumentaci¹.

V našem případě budeme inicializovat měření každých 10 minut. K tomu nám poslouží jednoduchý CRON příkaz. O převod z na Triger se stará třída *CronScheduleBuilder*.

Hlavní spustitelná třída *main.java*, metoda *main()*.

```
public static void main(String[] args) throws Exception {
    JobDetail job = JobBuilder.newJob(TimedJob.class)
        .withIdentity("Job", "group1").build();
    Trigger trigger = TriggerBuilder
        .newTrigger()
        .withIdentity("Trigger", "group1")
        .withSchedule(
            CronScheduleBuilder.cronSchedule("0 0/10 * * * ? *"))
        // exec new TimedJob Thread every 10 minutes
        .build();

    Scheduler scheduler = new StdSchedulerFactory().getScheduler();
    scheduler.start();
    scheduler.scheduleJob(job, trigger);
}
```

4.4.6 Socket komunikace - Server

Pro socket komunikaci je potřeba vytvořit v novém vlákně server naslouchající na zvoleném portu.

Základem je třída *SocketServer.java* dědící z třídy *Thread*. Metoda *run()* nastartuje server, který naslouchá na portu 9999.

```
public void run() {
    while (true) {
        try {
            System.out.println("Waiting for client on port " +
                serverSocket.getLocalPort() + "...");
            Socket server = serverSocket.accept();
            System.out.println("Just connected to " +
                server.getRemoteSocketAddress());
            DataInputStream in = new DataInputStream(server.getInputStream());
            String messString = in.readUTF();
            System.out.println(messString);
            DataOutputStream out
                = new DataOutputStream(server.getOutputStream());

            String message = resolveMessage(messString);
```

¹ <http://quartz-scheduler.org/documentation>

```

        out.writeUTF(message);
        System.out.println("SEND: " + message);

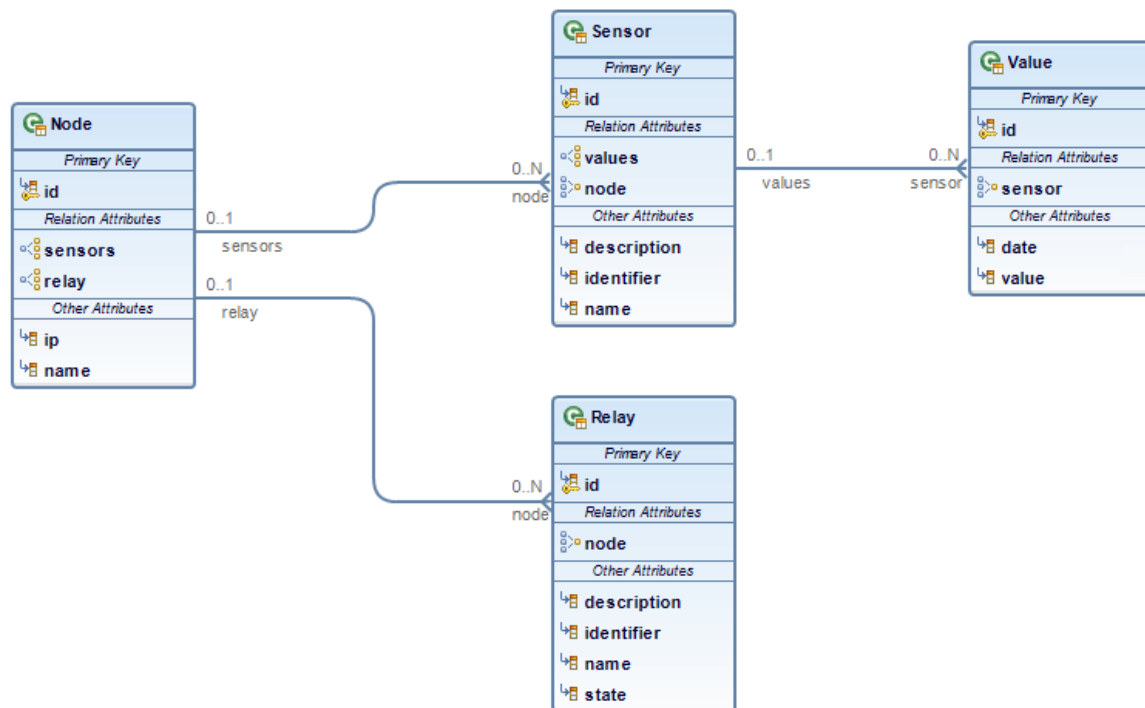
    } catch (SocketTimeoutException s) {
        System.out.println("Socket timed out!");
        break;
    } catch (IOException e) {
        break;
    }
}
}

```

4.5 Webové rozhraní

4.5.1 Entity

Obrázek 2 identifikuje klíčové entity systému.



Obrázek 2: Identifikace jednotlivých entit systému

4.5.2 Socket komunikace – Klient

Klient nejprve vyhledá odpovídající uzel pro daný aktuátor,

```
Relay relay = relayService.findSensorbyName(name);
String serverName = relay.getNode().getIp();
int port = Integer.parseInt("9999");
```

následně na něj zašle zprávu. K tomu slouží metoda *setupServer()*.

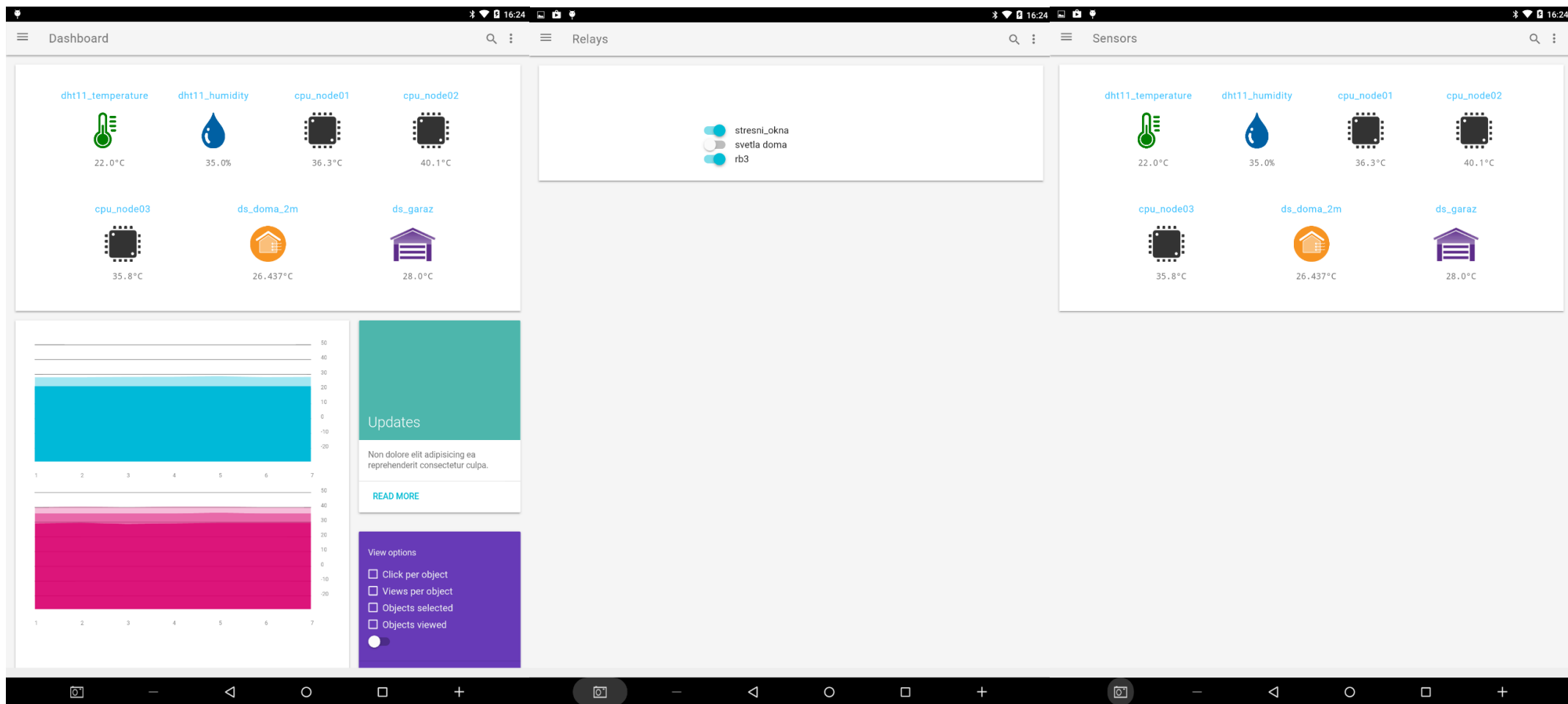
```
private String setupServer(Model model, String serverName, int port,
String message) {
    String receivedMessage = null;
    try {
        System.out.println("Connecting to " + serverName + " on port "
+ port);
        Socket client = new Socket(serverName, port);
        OutputStream outToServer = client.getOutputStream();
        DataOutputStream out = new DataOutputStream(outToServer);
        out.writeUTF(message);
        InputStream inFromServer = client.getInputStream();
        DataInputStream in = new DataInputStream(inFromServer);
        receivedMessage = in.readUTF();
        System.out.println("Server respond: " + receivedMessage);

        client.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return receivedMessage;
}
```

4.5.3 Design

Při vývoji uživatelského rozhraní byla použita šablona od společnosti Google *Material Design Lite*². Ta poskytuje responsivní webdesign včetně podpory mobilních zařízení. Pro potřeby projektu se jeví jako ideální. V rámci Dashboard byla dále použita SVG vektorová grafika pro vytváření grafů. Obrázek 3 zobrazuje ukázky vybraných částí systému.

² <https://github.com/google/material-design-lite>



Obrázek 3: Ukázky uživatelského rozhraní

5 Specifikace testování software

V rámci testování popsaného v praktické části diplomové práce byl vytvořen jednoduchý java script zaznamenávající časy při stisku tlačítka a po obdržení odpovědi ze serveru.

Po stisku příslušného tlačítka dojde k inicializaci metody *sendMessage()*, po jejím dokončení dojde k porovnání systémových časů a vypsání difference.

```
function sendMessage (name, state) {
    var startDate = performance.now();

    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (xhttp.readyState == 4 && xhttp.status == 200) {
            document.getElementById("demo").innerHTML =
            xhttp.responseText;
            document.getElementById("alert").style.visibility =
            "visible";
        }
    };
    xhttp.open("GET", "relay_get?name=" + name + "&state=" + state,
    true);
    xhttp.send();

    var endDate = performance.now();
    var diff = endDate - startDate;
    var str = "doba: " + diff + "\n"
    alert(str);
}
```

Naměřené výsledky jsou uvedeny v diplomové práci (Tabulka 7).