Středoškolská technika 2015

Setkání a prezentace prací středoškolských studentů na ČVUT

Dim-Box - Stmívač

Nguyen Son Hai

VOŠ a SPŠE Olomouc

Božetěchova 3, Olomouc

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v seznamu vloženém v práci SOČ.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné. Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) v platném znění

V Olomouci dne ..... podpis: .....

Seznam zkratek

DPS – deska plošného spoje

Poděkování

Tímto bych chtěl poděkovat Ing. Haninovi a Ing. Pospíšilovi za obětavou pomoc a asistenci při realizaci hardwarové části, Ing. Veselé za ochotu a vedení, Jiřímu Hanákovi za asistenci a pomoc při realizaci hardwaru, Jiřímu Jurečkovi za vytisknutí krabičky.

Anotace

Tato práce pojednává o návrhu a realizaci autonomního stmívače. Samotné stmívání je ovládáno pomocí Raspberry Pi, na kterém je spuštěn server napsaný v jazyku Python3 a který komunikuje pomocí JSON příkazů. Rozhraní pro klienta, je vytvořeno v jazycích QML, C++ a Javascript, jako mobilní aplikace spustitelná na systémech Android, iOS a možná Windows Phone 8.1. Aplikace je spustitelná i na Windows i Linuxu, ale není graficky přizpůsobená pro tyto systémy. Stmívač je schopen pracovat do napětí o velikosti 12V a proudu 1A.

Klíčová slova: Raspberry Pi, Python3, stmívač, JSON, QML, C++, Javascript, Android, iOS, Windows Phone 8.1, Windows, Linux

Annotation

This work is dealing with design a construction of autonomous dimmer. Dimming itself is controlled by Raspberry Pi, on which is running server written in language Python3 a communicate with JSON commands. Interface for client is created in languages QML, C++ and Javascript as mobile app runnable on systems Android, iOS and maybe Windows Phone 8.1. App is runnable on Winddows and Linux as well, but it isn't graphically adapted for these systems. Dimmer is able to work with voltage up to 12V and 1A current.

Key words: Raspberry Pi, Python3, dimmer, JSON, QML, C++, Javascript, Android, iOS, Windows Phone 8.1, Windows, Linux

Obsah

1. Úvod

Cílem této práce bylo vytvořit LED stmívač ovládaný pomocí Raspberry Pi. Jako rozhraní ke stmívači jsem zvolil mobilní aplikaci namísto webové stránky z důvodu použitelnosti. Dále velice důležitou prioritou byl design mobilní aplikace i stmívače. Hned na začátku jsem se potýkal se spoustou problémů ať už vytvoření malé DPS, výběr vhodného spínacího prvku, špatný model pouzdra pro stmívač, či absence custom komponent v jazyce QML. Má sice implementované všechny možné vstupy, které jdou jednoduše přestylovat, ale výstupní komponenty nejdou implementovat všechny, tudíž jsem musel vytvořit skoro všechny komponenty pouze pomocí komponent Rectangle, Image, Canvas, Text a Flickable, sice bylo použito více komponent, ale tyto jsou ty nejpoužívanější. Dalším zádrhelem bylo, že Websockety nebyly ještě implementované pro Windows Phone, avšak i tento problém jsem nakonec vyřešil.

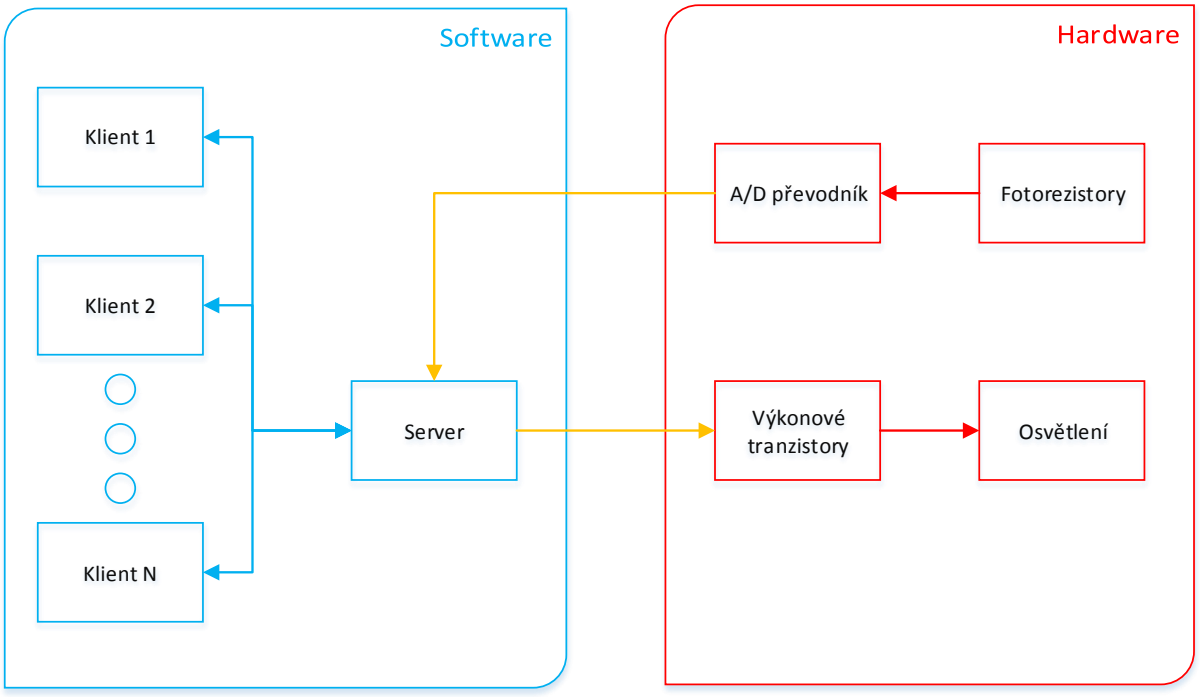
2. Použité technologie

Kvůli požadavkům potenciálních klientů, jsem zvolil popisovací jazyk QML namísto C++. Hlavním důvodem je snížení velikosti aplikace, avšak na úkor potřebné paměti RAM.

Server jsem se rozhodl naprogramovat ve skriptovacím jazyce Python namísto v C++, z důvodu zvýšení efektivity vývoje na Raspberry Pi. Pro přenos příkazů byl použit Javascriptový objektový zápis (JSON).

3. Popis činnosti stmívání

Obrázek č. 1: Blokové schéma stmívače

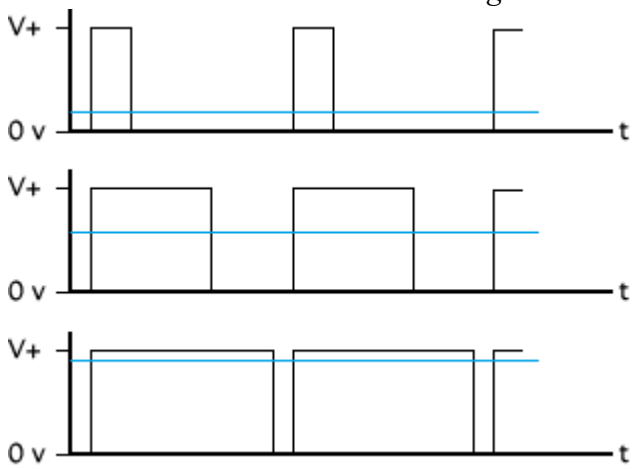


Raspberry Pi, které představuje server, nám zprostředkovává jednoduchý přechod z hardwaru na software, díky hardwarovým vstupně/výstupním pinům.
V jednoduchosti klient pošle příkaz a server ho zrealizuje, přepošle nebo na něj odpoví. Díky komunikaci pomocí websocketů může server nezávisle kdykoliv poslat klientovi zprávu.

3.1 Stmívání

Nejdůležitější část celého systému je stmívání. Stmívání osvětlení je realizováno pomocí pulzní šířkové modulace (PWM) na frekvenci 333Hz.

Obrázek č. 2: Ukázka PWM signálu



O manipulaci s jednotlivými piny se stará třída PWMGenerator, jejíž instance jsou obsaženy ve třídě HardwareContainer. PWMGenerator pouze obaluje metody modulu RPIO.

Úryvek kódu č. 1: Třída PWMGenerator – Server

```
import RPIO.PWM as PWM

PWM.set_loglevel(PWM.LOG_LEVEL_ERRORS)

class PWMGenerator(object):
    def __init__(self, pin, width):
        """
        :param pin: int
        :param width: int
        """

        self.__pin = pin
        self.__width = width
        self.__generator = PWM.Servo(0, 3000, 1)

    @property
    def width(self):
        return self.__width

    @width.setter
    def width(self, value):
        """
        :param value: int
        """

        self.__width = value
        try:
            if value:
                self.__generator.set_servo(self.__pin, pulse_width_us = value * 30)
            else:
                self.__generator.stop_servo(self.__pin)
        except RuntimeError:
            self.__generator.set_servo(self.__pin, 2990)

```

Drážka posuvníku je vykreslena přes Canvas, ale úchyt už je vytvořen z komponenty Rectangle.

Úryvek kódu č. 2: Vykreslení posuvníku - Klient

```
    onPaint: {      //draw groove//
        var ctx = canvas.getContext('2d');
        var centerX = width;
        var centerY = height / 2.0;

        ctx.clearRect(x, y, width, height)

        ctx.lineWidth = lineWidth
        ctx.strokeStyle = color
        ctx.beginPath();
        ctx.arc(centerX, centerY, radius, Math.PI / 2.0, Math.PI * 1.5, false)
        ctx.stroke()
        ctx.closePath();

        ctx.lineWidth = lineWidth
        ctx.strokeStyle = activeColor;
        ctx.beginPath();
        ctx.arc(centerX, centerY, radius, Math.PI / 2.0, Math.PI*((toggleArea.rotation + 180) /180.0) , false)
        ctx.stroke()
        ctx.closePath();
    }


    Rectangle {     //rotating area//
        id: toggleArea

        width: parent.radius + parent.lineWidth / 2 + (toggleSize - lineWidth) / 2
        height: parent.toggleSize

        color: "transparent"
        rotation: -90
        transformOrigin: Item.Right

        anchors.right: parent.right
        anchors.verticalCenter: parent.verticalCenter

        Rectangle {     //toggle button//
            width: canvas.toggleSize
            height: width;
            radius: width

            color: canvas.toggleColor
            antialiasing: true

            anchors.left: parent.left
            anchors.verticalCenter: parent.verticalCenter
        }
    }
```

Stmívání můžeme rozdělit do čtyř fází:

1) Zamykání
2) Stmívání
3) Odemykání
4) Synchronizace

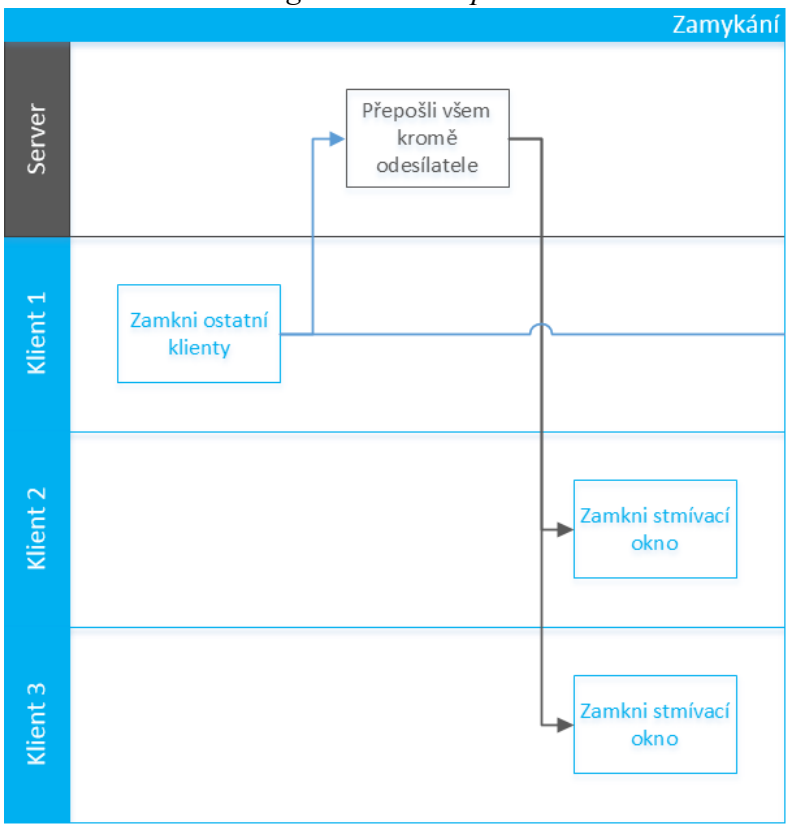
3.1.1 Zamykání

Změníme-li hodnotu na posuvníku, pošle se serveru příkaz, aby přeposlal všem ostatním klientům zprávu, že stmívání nelze nyní ovlivnit, avšak klient, který poslal serveru příkaz, může kdykoliv změnit hodnotu na posuvníku včetně během už probíhané animace stmívání.

Úryvek kódu č. 3: Přeposílání zámků – Server

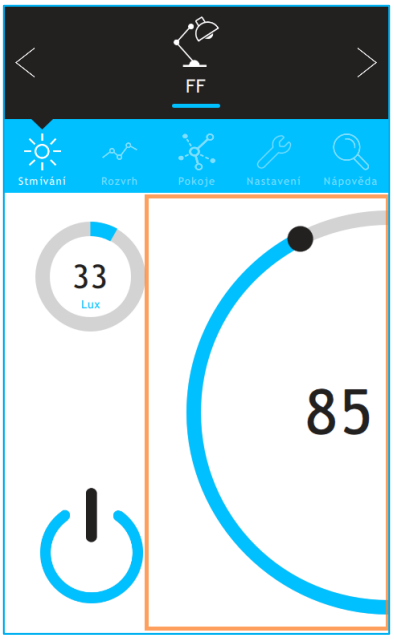
```
def on_message(self, message):
    print(message)
    message = loads(message)

    if message["action"] in ["init_channel", "remove_channel", "lock", "remove_schedule_point", "init_schedule_point"]:
        WSHandler.message_handler.broadcast_data(message, self)
```

Obrázek č. 3: Diagram činnosti při zamčení stmívání



Obrázek č. 4: Zobrazený zámek

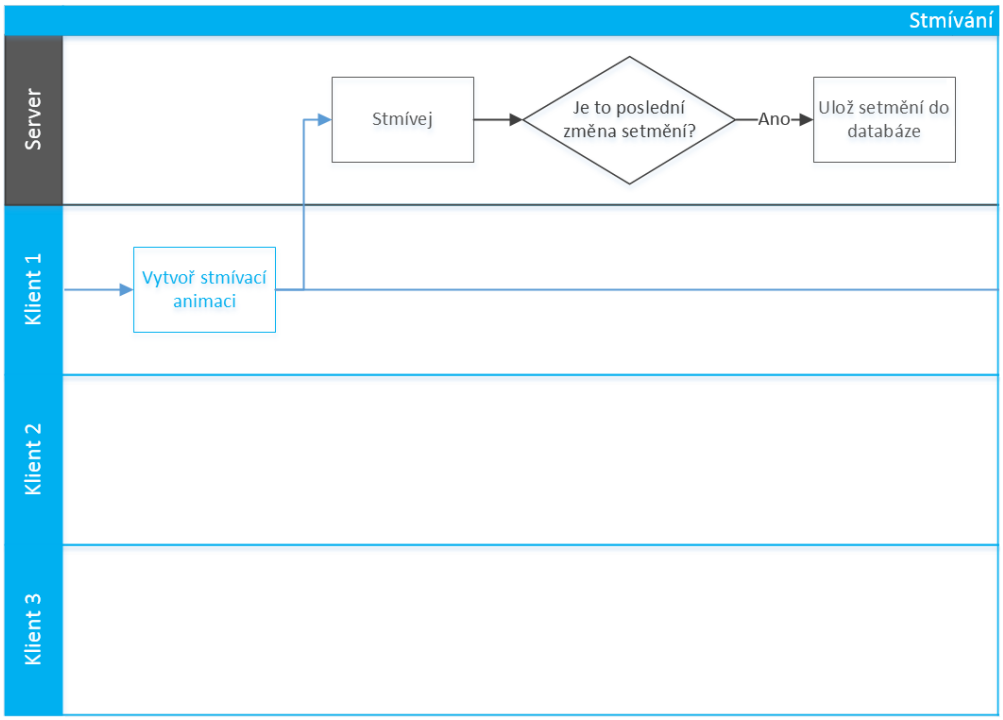
Obrázek č. 5: Pozice posuvníku v aplikaci



3.1.2 Stmívání

Po zamčení ostatních klientů se přechází na hlavní část – stmívání. Hodnota setmění se nemění skokově, ale lineárně a trvá vždy jednu sekundu nehledě na rozdíl mezi původní a novou hodnotou. Stmívací animaci vytváří klient. Pokaždé když se změní setmění, ovlivní se pouze výstup hardwaru, setmění pokoje se uloží až když klient, který vytváří animaci, označí změnu setmění jako poslední.

Obrázek č. 6: Diagram činnosti při stmívání



Úryvek kódu č. 4: Realizace stmívací animace – Klient

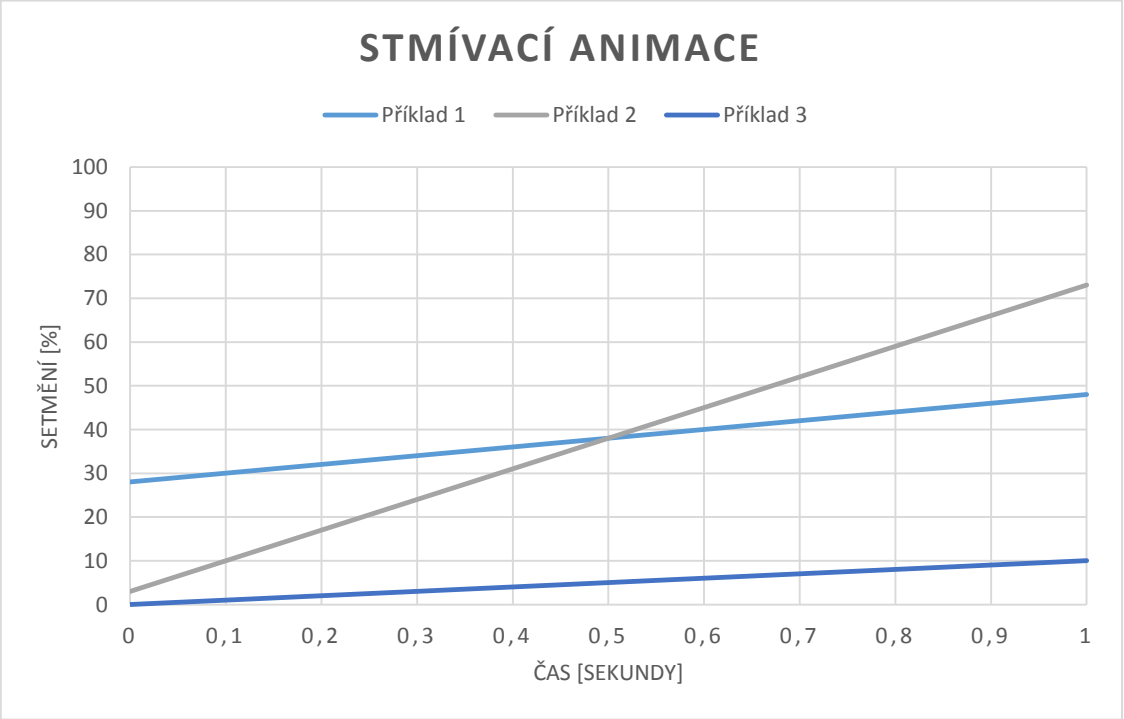
```
30      //----------KŘIVKA STMÍVÁNÍ----------
31      Behavior on value {
32          id: linearChange
33          enabled: !tempData.lockDim  //ochrana proti zpětnovazební smyčce
34          NumberAnimation {
35              duration: 1000
36              onRunningChanged: {
37                  Socket.sendLock(tempData.actualChannel, "dim", running) //zámek
38                  if(!running)
39                      Socket.sendDim(slider.value, tempData.actualChannel, true)
40              }
41          }
42      }
43      //-----------------------------------
```

Úryvek kódu č. 5: Nastavení setmění - Server

```
54      def set_dim(self, pin, dim, last):
55          """
56          :param room_label: string
57          :param dim: int
58          :param last: bool
59          """
60
61          self.__PWMOutputs[pin].width = dim
62          if last:
63              self.__DB.data[str(pin)]["dim"] = dim
64              self.__DB.save()
```

Na řádku číslo 38 a 39 můžeme vidět označení poslední změny setmění. Behaviour blok se vždy vykoná při změně hodnoty na posuvníku, avšak animace setmění neovlivňuje hodnotu na něm.

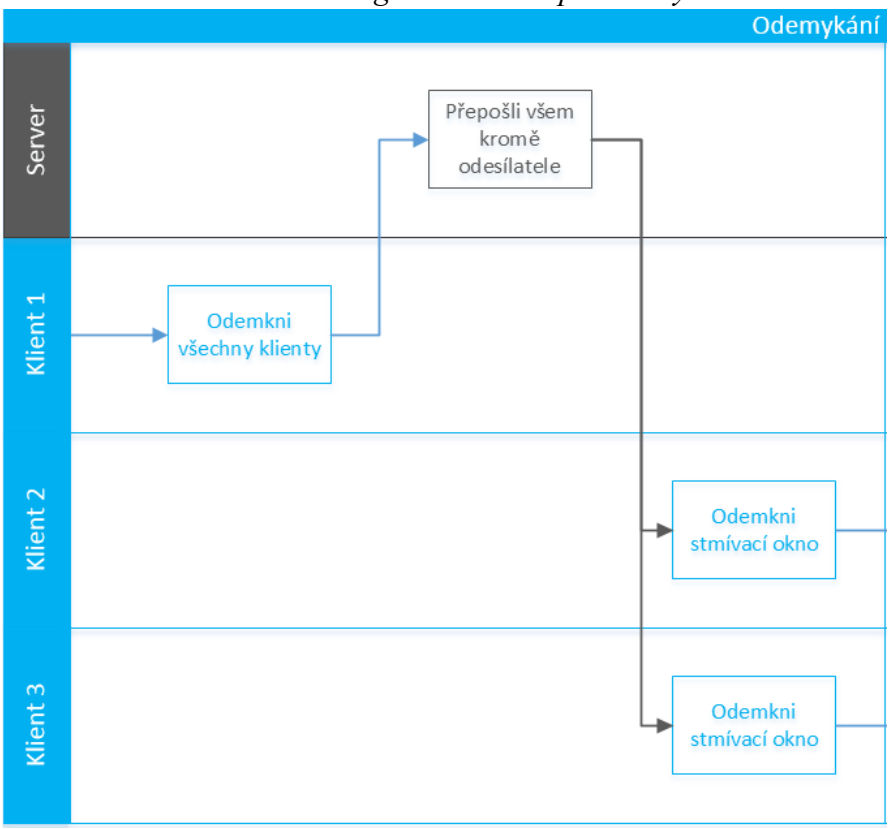
Obrázek č. 7: Graf stmívací animace



3.1.3 Odemykání

Odemykání probíhá obdobně jako zamykání. Jakmile server obdrží příkaz na změnu setmění označený jako poslední, přepošle všem kromě odesílatele změn setmění příkaz na odemčení stmívacího okna.

Obrázek č. 8: Diagram činnosti při odemykání



3.1.4 Synchronizace

Na odemčení okna stmívání je připojena událost, která pošle serveru požadavek na zaslání aktuálního setmění.

Úryvek kódu č. 6: Požadavek připojený na změnu zámku – Klient

```
75      onLockDimChanged: {
76          if(!lockDim)
77              Socket.requestDim()
78      }
79  }
```

RequestDim je metoda, která naformátuje JSON příkaz a odešle ho serveru.

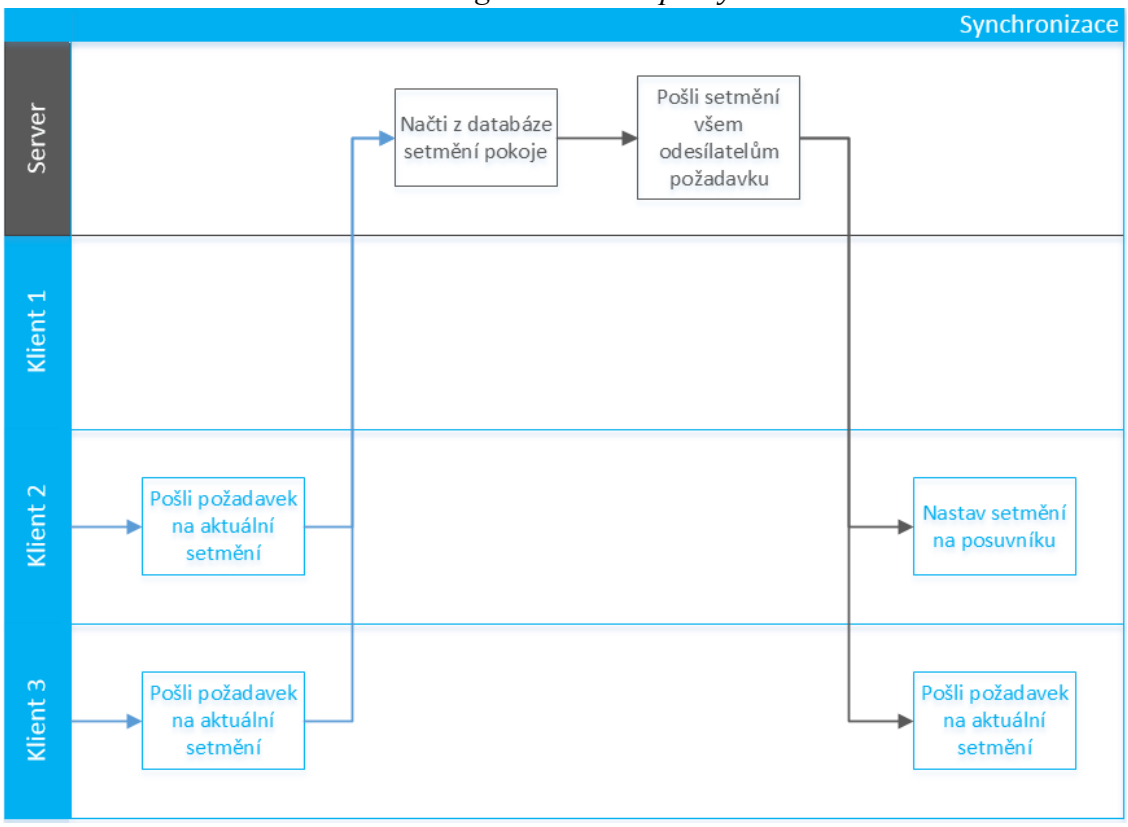
Úryvek kódu č. 7: Metoda requestDim – Klient

```
82  function requestDim() {
83      var data = {
84          "action": "get_dim",
85          "pin": tempData.actualChannel
86      }
87
88      root.socket.sendTextMessage(JSON.stringify(data))
89  }
```

Úryvek kódu č. 8: Odpověď na požadavek na aktuální setmění - Server

```
66      def send_dim(self, pin, requester=None):
67          """
68          :param pin: int
69          """
70
71          data = {
72              "action": "set_dim",
73              "pin": pin,
74              "dim": self.__DB.data[str(pin)]["dim"]
75          }
76
77          if requester:
78              self.send_data_to(data, requester)
79          else:
80              self.broadcast_data(data)
```

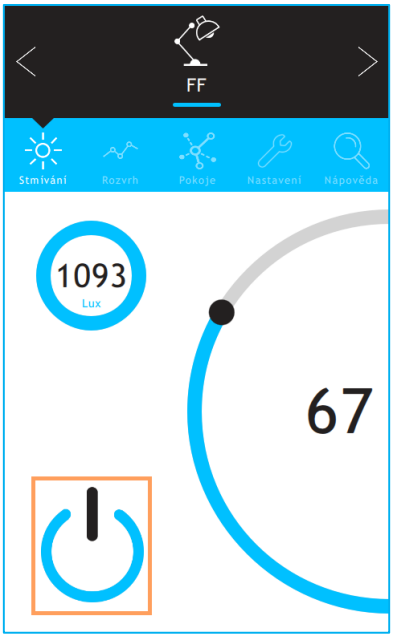
Obrázek č. 9: Diagram činnosti při synchronizaci



3.2 Vypínač

Pro rychlou a jednoduchou manipulaci se setměním, jsem vytvořil vypínač, který když uvedeme do neaktivní polohy, tak nastaví setmění na hodnotu 0 a když ho přivedeme zpět do aktivní polohy, nastaví setmění, které bylo nastaveno, než jsme ho uvedli do neaktivní polohy. Vypínač je vytvořen ze dvou obrázků, které se překrývají.

Obrázek č. 10 vypínač – pozice v aplikaci



3.2.1 Přepnutí do neaktivní polohy

Při uvedení přepínače do neaktivní polohy se nejdříve pošle požadavek na uložení aktuálního setmění a poté až příkaz na změnu setmění viz úryvek kódu č. 9 řádek číslo 60 a 61**.**

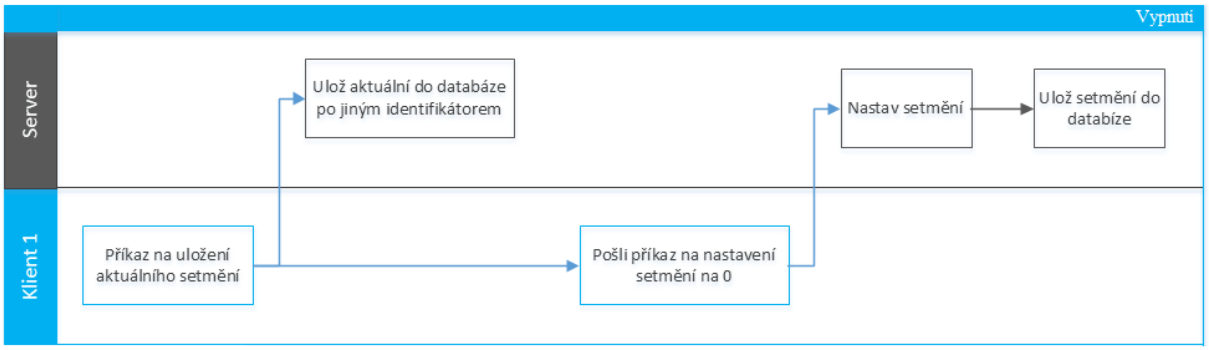
Úryvek kódu č. 9: Implementace funkčnosti vypínače - Klient

```
47      Controls.OffButton {
48          id: powerButton
49
50          width: RL.calcSize("height", 153)
51          height: RL.calcSize("height", 159)
52
53          anchors.bottom: parent.bottom
54          anchors.left: parent.left
55          anchors.bottomMargin: RL.calcSize("height", 40)
56          anchors.leftMargin: RL.calcSize("width", 40)
57
58          onActiveChanged: {
59              if(!active) {
60                  Socket.saveLastDim()
61                  slider.setValue(0, false)
62              }
63
64              else
65                  Socket.setLastDim()
66          }
67      }
```

Úryvek kódu č. 10: Uložení setmění - Server

```
82      def save_last_dim(self, pin):
83          """
84          :param pin: int
85          """
86
87          self.__DB.data[str(pin)]["last_dim"] = self.__DB.data[str(pin)]["dim"]
88          self.__DB.save()
```

Obrázek č. 11: Diagram činnosti při uvedení vypínače do neaktivní polohy

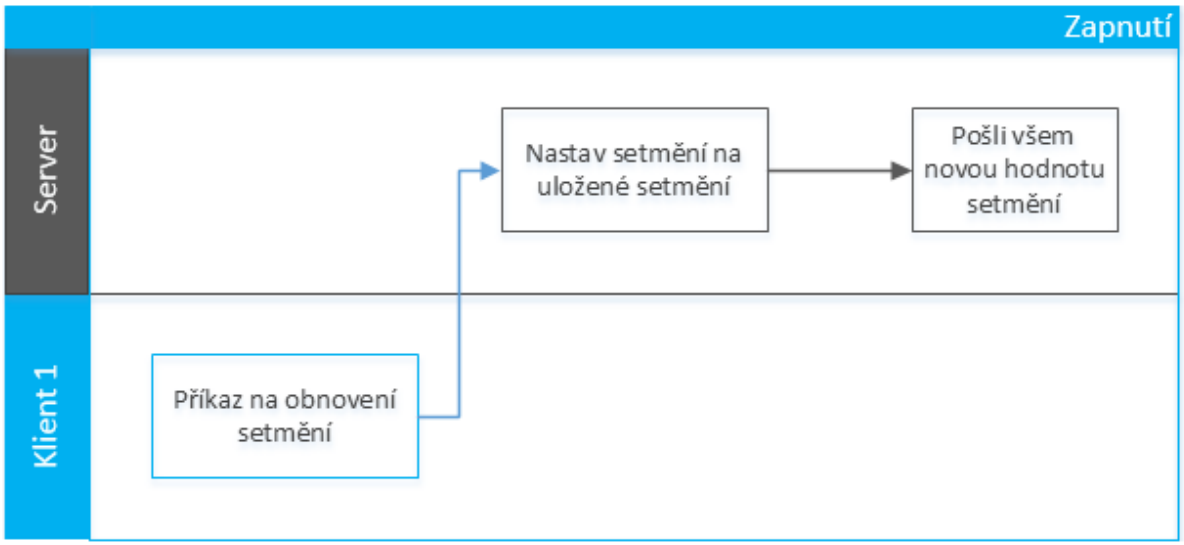


Obrázek č. 12 vypínač - v neaktivní poloze

3.2.2 Přepnutí do aktivní polohy

Při uvedení vypínače do aktivní polohy se pouze pošle příkaz serveru na obnovení uloženého setmění.

Obrázek č. 13: Diagram činnosti při uvedení vypínače do aktivní polohy



Obrázek č. 14 vypínač - v aktivní poloze

4. Popis činnosti senzorů

Senzory svítivosti jsou důležitým rozšířením, umožňují totiž klientovi odhadnout, zda je pokoj osvětlen ať už osvětlením, či Sluncem.

4.1 Čtení hodnot z fotorezistoru

Jak již může být zřejmé, fotorezistor mění svoji rezistenci vůči intenzitě osvětlení. Tudíž fotorezistor je zapojen jako dělič, abychom přeměnili rezistenci na napětí, které už je měřeno pomocí A/D převodníku MCP3008 s SPI interfacem. O čtení napětí se stará třída AnalogReader.

Úryvek kódu č. 11: Třída AnalogReader – Server

```
from settings import Settings
import spidev
import time

class AnalogReader():
    def __init__(self):
        self.__spi = spidev.SpiDev()
        self.__spi.open(0, 0)
        self.__readings = dict()

    @property
    def readings(self):
        return self.__readings

    @readings.setter
    def readings(self, value):
        self.__readings = value

    def read_channel(self, channel):
        adc = self.__spi.xfer2([1, (8 + channel) << 4, 0])
        return ((adc[1] & 3) << 8) + adc[2]

    def read_all(self):
        """
        :param address: int
        :return: list
        """

        self.__readings = dict()

        for channel in range(Settings.NUMBER_OF_CHANNELS):
            self.__readings[channel] = self.read_channel(channel)
            time.sleep(0.1)
```

4.2 Výpočet svítivosti

Z A/D převodníku získáme pouze 10 bitové číslo, tudíž je nutné z něho určit napětí, z napětí rezistenci a z rezistence svítivost. O tyto výpočty se stará třída IlluminanceManager

Úryvek kódu č. 12: Výpočet svítivosti z A/D hodnoty – Server

```
27      def __raw2voltage(self, raw_value):
28          """
29          :param raw_value: int
30          :return: float
31          """
32          raw_value = 0 if raw_value <= 15 else raw_value
33          return Settings.REFERENCE_VOLTAGE * (raw_value / Settings.AD_PRECISION)
34
35      def __voltage2resistance(self, voltage):
36          """
37          :param voltage: float
38          :return: float
39          """
40
41          return (voltage * Settings.RB1) / (Settings.REFERENCE_VOLTAGE - voltage)
42
43      def __resistance2illuminance(self, resistance):
44          """
45          :param resistance: float
46          :return: float
47          """
48
49          if resistance == 0:
50              return 0
51
52          inverted_gama = 1 / Settings.GAMA
53          return ((Settings.R_AT_10_LUX ** (inverted_gama)) * 10 * (resistance ** (-inverted_gama)))
```

4.3 Přenesení dat klientovi

Už máme vypočítané hodnoty a nyní bychom je mohli poslat klientovi. Třída MessageHandler obsahuje metodu send_illuminance, která je spuštěna v dalším vlákně, aby nezpožďovala hlavní programovou smyčku.

Úryvek kódu č. 13: Metoda send_illuminance - Server

```
196     def send_illuminance(self):
197         while True:
198             illuminance = self.__illuminance_manager.readings()
199             data = {
200                 "action": "illuminance_read",
201                 "readings": illuminance
202             }
203             self.broadcast_data(data)
```

Úryvek kódu č. 14: Vytvoření nového vlákna pro čtení svítivosti - Server

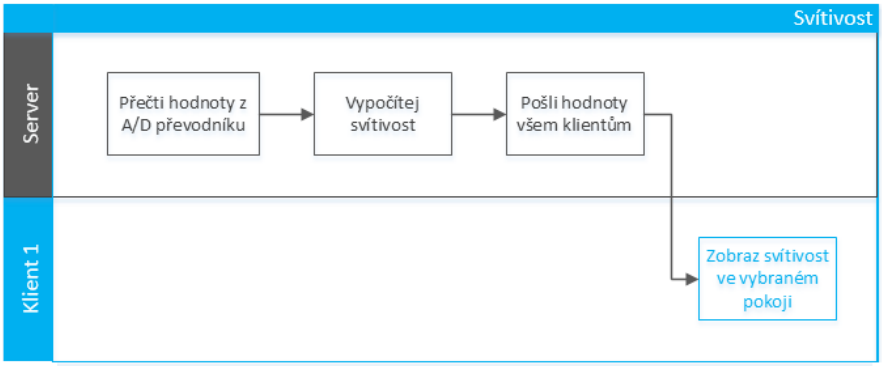
```
70      threading.Thread(target=WSHandler.message_handler.send_illuminance).start()
71      scheduleloop.start()
72      serverloop.start()
```

O prezenci svítivosti se stará komponenta CircleProgressBar, je vykreslena pomocí Canvasu.

Úryvek kódu č. 15: Nastavení komponenty CircleProgressBar - Klient

```
27    Controls.CircleProgressBar {
28        height: RL.calcSize("height", 150)
29        width: RL.calcSize("height", 150)
30
31        precission: (root.illuminance <= 1) ?1 :0
32        maximum: 400.0
33        minimum: 0
34        value: root.illuminance
35        lineWidth: RL.calcSize("height", 20)
36
37        activeColor: root.primaryColor
38        textColor: root.secondaryColor
39        grooveColor: root.lineColor
40
41        anchors.top: parent.top
42        anchors.topMargin: RL.calcSize("height", 40)
43        anchors.left: parent.left
44        anchors.leftMargin: RL.calcSize("width", 40)
45    }
```

Obrázek č. 15: Diagram činnosti při čtení svítivosti

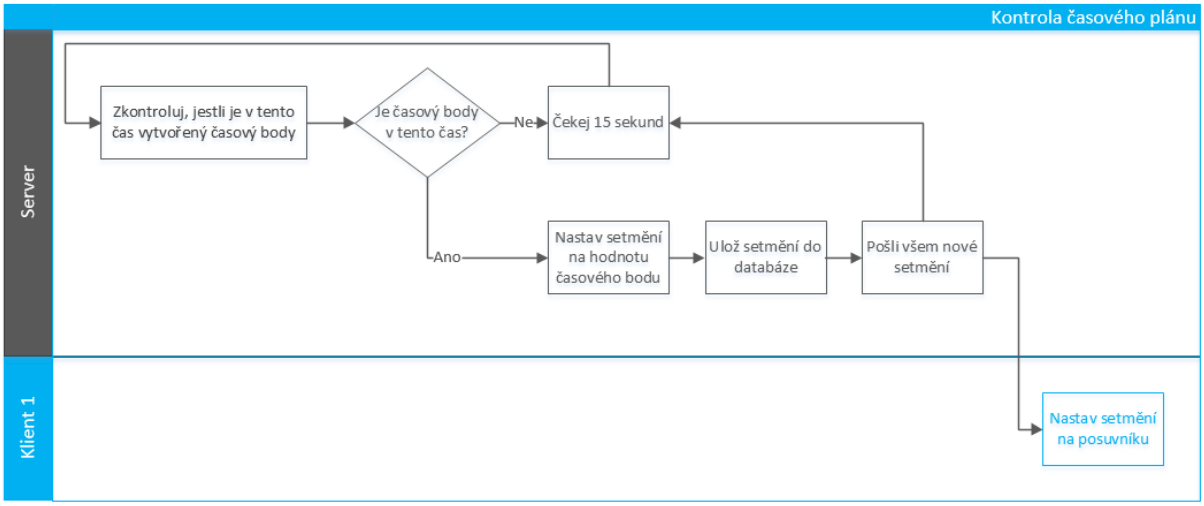


Obrázek č. 16: Ukazatel svítivosti při svítivosti menší než 1

5. Popis činnosti časového plánu

Pokud máme povolený časový plán a máme vytvořené časové body, bude se nám automaticky nastavovat setmění podle předem námi vytvořených hodnot. O kontrolu aktuálního času s časovými body se stará server přesněji třída Scheduler. Časové body mají unikátní indexy vytvořené z času jejich aktivace, tudíž může být vytvořen pouze jeden časový bod v jeden určitý čas, což je žádoucí. Index je vytvořen ve tvaru HHMM.

Obrázek č. 17: Diagram činnosti při kontrole časového plánu



Úryvek kódu č. 16: Třída Scheduler – Server

```
class Scheduler():
    def __init__(self, raw_DB, dim_setter, broadcast_function, time_sender):
        """
        :param raw_DB: dict
        """
        self.__raw_DB = raw_DB
        self.__dim_setter = dim_setter
        self.__broadcast_function = broadcast_function
        self.__time_sender = time_sender

    def get_time(self, send=False):
        """
        :send: bool
        :return: dict
        """
        actual_time = strftime("%H:%M")
        result = {
            "hour": int(actual_time.split(":")[0]),
            "minute": int(actual_time.split(":")[1])
        }
        if send:
            self.__time_sender(result["hour"], result["minute"])
        return result

    def __parse_time(self):
        """
        :return: string
        """

        actual_time = self.get_time()
        hour = actual_time["hour"]
        minute = actual_time["minute"]

        self.__time_sender(hour, minute)

        return str(hour * 100 + minute)

    def __parse_from_DB(self):
        """
        :return: dict
        """
        schedule = dict()

        for k, v in self.__raw_DB.items():
            if(not v["lock_graph"]):
                schedule[k] = v["schedule"]

        return schedule

    def check(self):
        whole_schedule = self.__parse_from_DB()
        timeIndex = self.__parse_time()

        for pin, schedules in whole_schedule.items():
            if timeIndex in schedules.keys():
                self.__dim_setter(int(pin), whole_schedule[pin][timeIndex])
                self.__broadcast_function(int(pin))
```

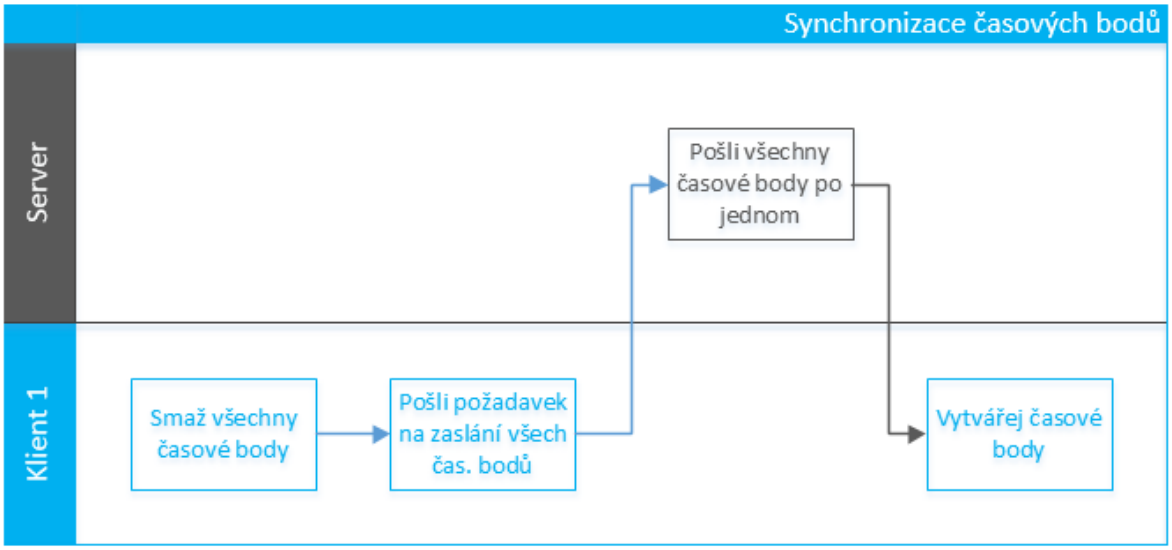
Úryvek kódu č. 17: Periodické volání kontroly času pomocí IOLoop – Server

```
scheduleloop = tornado.ioloop.PeriodicCallback(WSHandler.scheduler.check, 15000)
```

5.1 Synchronizace s aktuálním pokojem

Je velice důležité udržet na všech zařízeních stejná data, zámky tomu velice dopomáhají, avšak se může stát, že se změní data těsně před zamčením, tudíž jsem přidal opětovnou synchronizaci se serverem, k synchronizaci časových bodů dochází vždy při změně aktuálního pokoje a při připojení k serveru.

Obrázek č. 18: Diagram činnosti při synchronizaci časových bodů



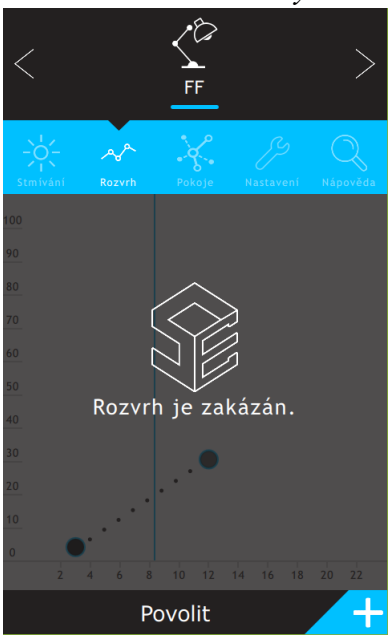
Úryvek kódu č. 18: Metoda, která pošle klientovi všechny časové body pokoje – Server

```
146     def send_all_schedule_points(self, pin, requester):
147         data = {
148             "action": "init_schedule_point"
149         }
150 
151         for k, v in self.__DB.data[str(pin)]["schedule"].items():
152             data["pin"] = pin
153             data["power"] = v
154             data["hour"] = int(int(k) / 100)
155             data["minute"] = int(k) % 100
156             self.send_data_to(data, requester)
157 
158         data = {
159             "action": "lock",
160             "target": "graph",
161             "pin": pin,
162             "lock": self.__DB.data[str(pin)]["lock_graph"]
163         }
```

5.2 Zapnutí/vypnutí časového plánu

Jak můžeme vidět z úryvku č. 16 na řádku 50, pokud není povolený časový plán, server se bude chovat jako by žádný neexistoval.

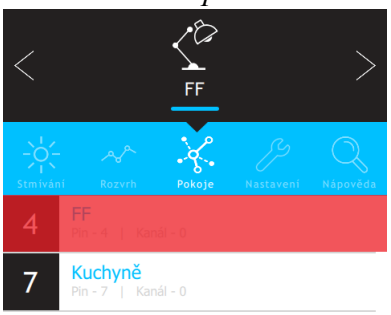
Obrázek č. 19: Zakázaný rozvrh



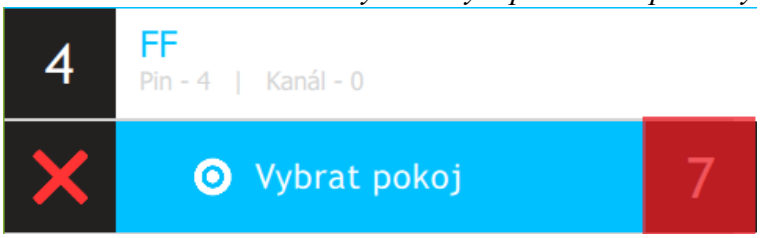
6. Popis činnosti výběru pokojů

Výběr aktuálního pokoje nijak neovlivní data serveru. Zároveň můžeme stmívat pouze jeden pokoj.

Obrázek č. 20: Prostor pro zobrazení nabídky



Obrázek č. 21: Zobrazená nabídka s vyznačeným prostorem pro skrytí nabídky



6.1 Synchronizace pokojů se serverem

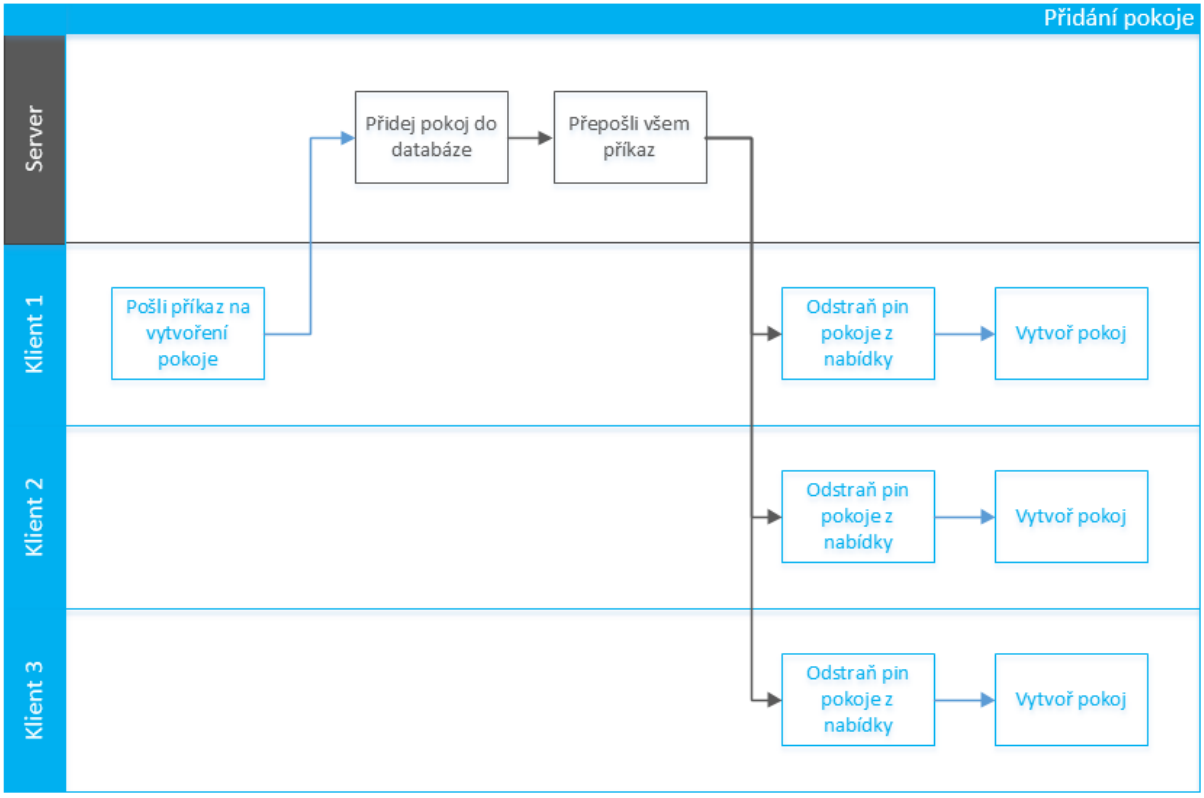
Synchronizace pokojů probíhá podobně jako u časových bodů, avšak s jediným podstatným rozdílem. Když přidáme nebo smažeme pokoj, zavolá se animované přepozicování pokoje. Tudíž se musí pokoje mazat s prodlevou 400 milisekund. O hromadné mazání pokojů se stará objekt ChannelDeleteManager.

Úryvek kódu č. 19: Objekt channelDeleteManager - Klient

```
    Timer {
        id: channelDeleteManager

        signal done()
        signal delay()
        property int counter: 0
        property var listOfDeleting: new Array
        property var synchronizationScreen

        repeat: true
        interval: 450
        triggeredOnStart: true
        running: false

        onRunningChanged: {
            if(!running) {
                listOfDeleting = new Array
                done()
            }
            else {
                Socket.requestAllPins()
                counter = 0
                tempData.syncDone = false
                synchronizationScreen.active = true
            }
        }
        onTriggered: {
            if(counter >= listOfDeleting.length)
                channelDeleteManager.running = false
            else
                CL.popRoom(listOfDeleting[counter], false)
            counter++

        }

        onDone: {
            if(root.socket.status == WebsocketClient.Open)
                Socket.requestAllChannels()
            synchronizationScreen.active = false
        }

        onDelay: SequentialAnimation {
            NumberAnimation { duration: 400 }
            ScriptAction { script: {tempData.syncDone = true} }
        }
    }
```

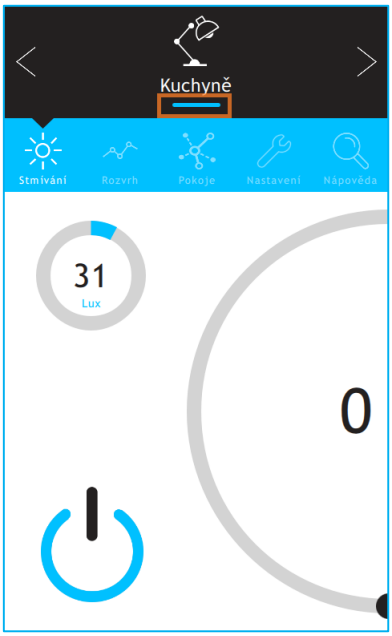
7. **Manipulace s pokoji a časovými body**
   Manipulace jako například smazání nebo přidání jsou v podstatě totožné jak s pokoji, tak s časovými body. Pro zjednodušení jsem vytvořil manipulaci s objekty, tak že když si například vytvoříme pokoj, tak ho samotný klient nevytvoří, ale pouze pošle serveru příkaz, aby vytvořil pokoj. Následně server tento příkaz pošle všem klientům, kteří až nyní vytvoří pokoj.

Obrázek č. 22: Diagram činnosti při přidání pokoje



8. **Připojení klienta k serveru**
   Jak již bylo zmíněno, klienti jsou k serveru připojení přes Websockety a komunikují spolu JSON příkazy. Aby uživatel věděl, že není připojen, v horní části aplikace je umístěn ukazatel připojení. Také první tři okna jsou zamčená, pokud klient není připojen k serveru.

Obrázek č. 23: Umístění ukazatele připojení



8.1 Reimplementace komponenty Websocket

Jak již tomu bývá, nic není dokonalé a již při snaze sestavení projektu na platformu Windows Phone 8.1, se ukázalo, že Windows Phone neobsahuje Websocket modul. Poté jsem byl donucen updatovat Qt framework na beta verzi 5.5, avšak naskytla se další chyba, websockety nebyly ještě implementovány pro jazyk QML, tudíž jsem musel sám napsat tuto komponentu v C++.

Úryvek kódu č. 20: Změna Websocket komponenty po přepsání do C++ - Klient

```
1   1     import QtQuick 2.0
2         - import Qt.WebSockets 1.0
      2   + import WebsocketClient 1.0
3   3
4   4     import "../../logic/channelLogic.js" as CL
5   5     import "../../logic/messageController.js" as Socket
6   6
7         - WebSocket {
      7   + WebsocketClient {
8   8         id: socket
9   9
10        -     signal reconnect()
11        -
12        -     active: false
13        -     //url: "ws://169.254.29.212:8888"
      10  +     Timer {
      11  +         id: reconnectTimer
      12  +         interval: 500
      13  +         running: false
      14  +         repeat: true
      15  +         onTriggered: socket.reconnect()
      16  +     }
14  17
15  18        onStatusChanged: {
16        -         var actualStatus = socket.status
17        -
18        -         switch(actualStatus) {
19        -             case WebSocket.Connecting:
20        -                 console.log("Connecting");
21        -                 break;
22        -
23        -             case WebSocket.Open:
24        -                 console.log("Open");
25        -                 root.connected = true
26        -                 CL.deleteAllChannels()
27        -                 break;
28        -
29        -             case WebSocket.Closing:
30        -                 console.log("Closing");
31        -                 break;
32        -
33        -             case WebSocket.Closed:
34        -                 root.connected = false
35        -                 console.log("Closed");
36        -                 socket.reconnect()
37        -                 break;
      19  +         if(status == WebsocketClient.Open) {
      20  +             root.connected = true
      21  +             reconnectTimer.running = false
      22  +             CL.deleteAllChannels()
      23  +         }
38  24
39        -             case WebSocket.Error:
40        -                 root.connected = false
41        -                 console.log("Error (" + socket.errorString + ")")
42        -                 break;
      25  +         else if(status == WebsocketClient.Closed) {
      26  +             root.connected = false
      27  +             if(!reconnectTimer.running)
      28  +                 reconnectTimer.running = true
43  29          }
44  30      }
45  31
46        -     onReconnect: SequentialAnimation {
47        -         NumberAnimation { duration: 200 }
48        -         ScriptAction { script: { socket.active = false; socket.active = true } }
      32  +     Component.onCompleted: {
      33  +         var ip = fileStream.read()
      34  +         if(ip != "") {
      35  +             socket.url = ip
      36  +             socket.reconnect()
      37  +         }
      38  +         loadingScreen.hide()
49  39      }
50  40
51        -
52  41      onTextMessageReceived: {
53  42          var data = JSON.parse(message)
      43  +         console.log(message)
54  44
55  45          switch(data["action"]) {
56  46              case "illuminance_read":
```

Hunk 2 : Lines 96-99

```
106 96                  break;
107 97          }
108 98      }
109       -     Component.onCompleted: {
110       -         var ip = fileStream.read()
111       -         if(ip != "") {
112       -             socket.url = ip
113       -             socket.active = true
114       -         }
115       -         loadingScreen.hide()
116       -     }
117 99      }
118       -
```

Úryvek kódu č. 21: websocketclient.h – Klient

```
#ifndef WEBSOCKETCLIENT_H
#define WEBSOCKETCLIENT_H

#include <QQuickItem>
#include <QtWebSockets/QtWebSockets>

class WebsocketClient : public QQuickItem
{
        Q_OBJECT
        Q_ENUMS(Statuses)
        Q_PROPERTY(int status READ status WRITE setStatus NOTIFY statusChanged)
        Q_PROPERTY(QUrl url READ url WRITE setUrl NOTIFY urlChanged)

    private:
        int m_status;
        QUrl m_url;
        QWebSocket m_socket;

    public:
        explicit WebsocketClient(QQuickItem *parent = 0);

        enum Statuses {Open, Closed};

        Q_INVOKABLE void sendTextMessage(QString message);
        Q_INVOKABLE void reconnect();
        int status() const;
        QUrl url() const;

    public slots:
        void setUrl(QUrl url);
        void setStatus(int status);

    private slots:
        void resendMessageSignal(QString message);
        void onConnected();
        void onDisconnected();

    signals:
        void textMessageReceived(QString message);
        void statusChanged(int status);
        void urlChanged(QUrl url);
};

#endif // WEBSOCKETCLIENT_H
```

Úryvek kódu č. 22: websocketclient.cpp bez setterů a getterů

```cpp
#include "websocketclient.h"
#include <QtWebSockets/QtWebSockets>

WebsocketClient::WebsocketClient(QQuickItem *parent): QQuickItem(parent)
{
    m_status = -1;
    m_url = QUrl("");
    QObject::connect(&m_socket, SIGNAL(connected()), this, SLOT(onConnected()));
    QObject::connect(&m_socket, SIGNAL(disconnected()), this, SLOT(onDisconnected()));
}

void WebsocketClient::sendTextMessage(QString message)
{
    m_socket.sendTextMessage(message);
}

void WebsocketClient::reconnect()
{
    m_socket.open(m_url);
}


void WebsocketClient::onConnected()
{
    setStatus(Open);
    QObject::connect(&m_socket, SIGNAL(textMessageReceived(QString)), this, SLOT(resendMessageSignal(QString)));
}

void WebsocketClient::onDisconnected()
{
    m_socket.close();
    QObject::disconnect(&m_socket, SIGNAL(textMessageReceived(QString)), this, SLOT(resendMessageSignal(QString)));
    setStatus(Closed);
}
```

9. Ukládání dat do souborů

Protože je důležité uchovat inicializační data jako IP adresa stmívače, abychom ji nemuseli po každém spuštění aplikace znovu zadávat, je nutno uložit tyto data do souboru. Problém s multiplatformností je, že každá platforma má jiná nastavení a přístupová práva, avšak Qt framework poskytuje makra, která signalizují, pro jakou platformu budeme sestavovat projekt, tudíž nemusíme neustále měnit kód, či mít více branchí (větev).

Úryvek kódu č. 23: Použití makra pro změnu cesty souboru – Klient

```cpp
void FileStream::setSource(QString &value) {
    #ifdef Q_OS_IOS
    QString path = QStandardPaths::standardLocations(QStandardPaths::DataLocation).value(0);
    QDir dir(path);
    if (!dir.exists())
        dir.mkpath(path);
    if (!path.isEmpty() && !path.endsWith("/"))
        path += "/";
        value = path + value;
    #endif

    if(p_source != value) {
        p_source = value;
        emit sourceChanged();
    }
}
```

10. JSON příkazy

10.1 Stmívání

```
{
     'action': 'dim',
     'pin': <int>,  //označuje pin pokoje
     'dim': <int>,  //hodnota setmění
     'last': <bool> //je poslední hodnota animace stmívání?
}
```

10.2 Poslání setmění pokoje

Požadavek od klienta

```
{
'action': 'get_dim',
'pin': <int>   //označuje pin pokoje
}
```

10.3 Ukládání setmění

Požadavek od klienta na uložení

```
{
     'action': 'save_last_dim',
     'pin': <int>   //označuje pin pokoje
}
```

Požadavek od klienta na obnovu posledního setmění

```
{
     'action': 'set_last_dim',
     'pin': <int>   //označuje pin pokoje

}
```

10.4 Zaslání všech časových bodů

Požadavek od klienta

```
{
     'action': 'init_all_schedule_points',
     'pin': <int>   //označuje pin pokoje
}
```

10.5 Přidání časového bodu

```
{
     'action': 'init_schedule_point',
     'pin': <int>,  //označuje pin pokoje
     'hour': <int>, //hodina
     'minute': <int>,    //minuta
     'power': <int>      //síla osvětlení
}
```

10.6 Smazání časového bodu

```
{
    'action': 'remove_schedule_point',
    'pin': <int>,  //označuje pin pokoje
    'hour': <int>, //hodina
    'minute': <int>     //minuta

}
```

10.7 Zaslání všech pokojů

Požadavek od klienta na inicializaci všech pokojů

```
{
    'action': 'init_all_channels'
}
```

10.8 Přidání pokoje

```
{
    'action': 'init_channel',
    'pin': <int>,  //označuje pin pokoje
    'sensor_channel': <int>, //číslo senzoru
    'title': <string>   //název pokoje

}
```

10.9 Smazání pokoje

```
{
    'action': 'remove_channel',
    'pin': <int>   //označuje pin pokoje

}
```

10.10 Zámky

```
{
    'action': 'lock',
    'lock': <bool>,     //zamčení nebo odemčení
    'pin': <int>,  //označuje pin pokoje
    'target': <string>  //cíl zámku
    //zámky pouze pro 'dim' || 'schedule' || 'graph'

}
```

10.11 Posílání všech pinů

Požadavek od klienta na poslání všech pinů

```
{
    'action': 'init_all_pins'
}
```

10.12 Posílání aktuálního času

```
{
    'action': 'set_time',
    'hour': <int>  //hodina
    'minute': <int>     //minuta
}
```

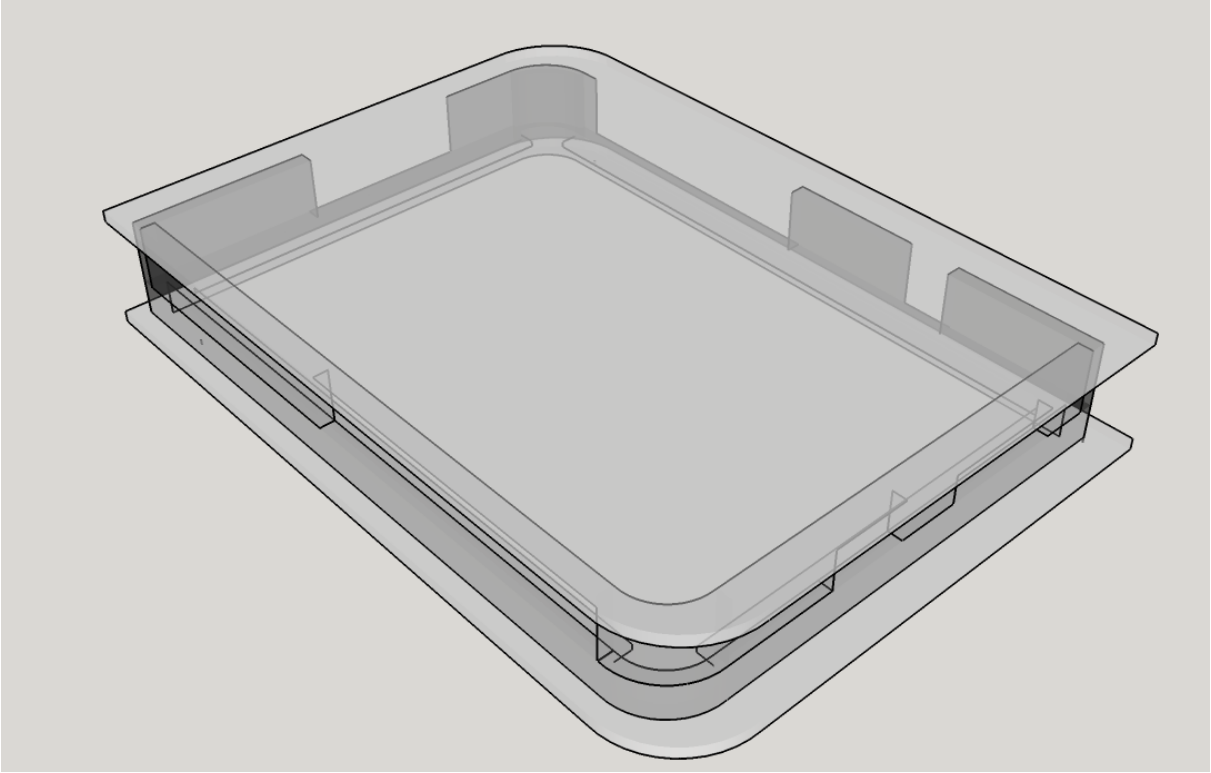
10.13 Posílání svítivosti v pokoji

```
{
    'action': 'illuminance_read',
    'readings': <dict>
    //[<int> číslo senzoru]: lux
}
```

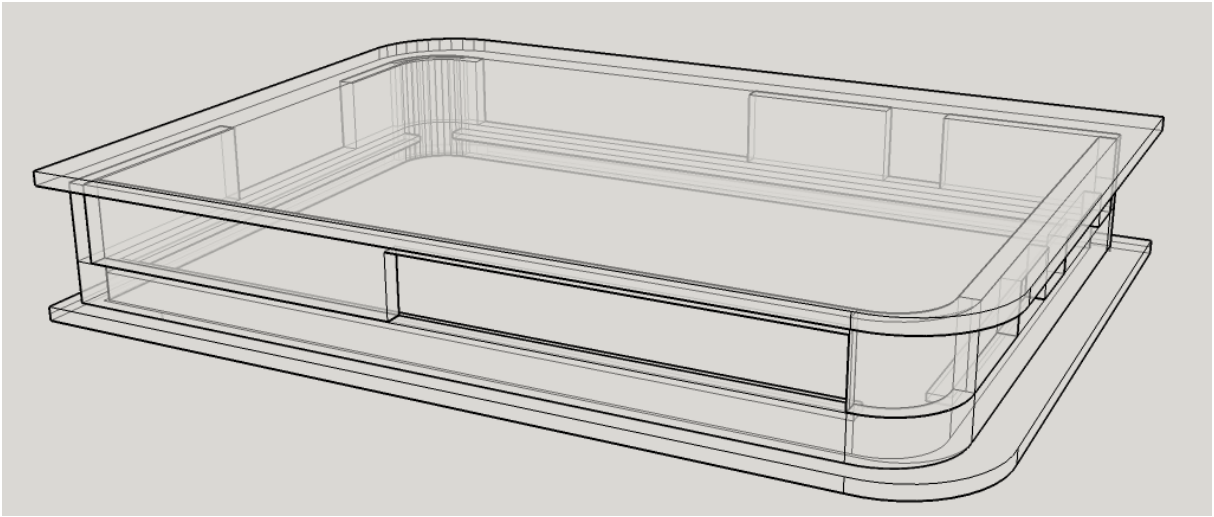
11. Pouzdro pro stmívač

Model pouzdra byl vytvořen v programu SketchUp a následně vytisknut 3D tiskárnou. Přidal jsem na obě strany hliníkový plech kvůli chlazení i designu.

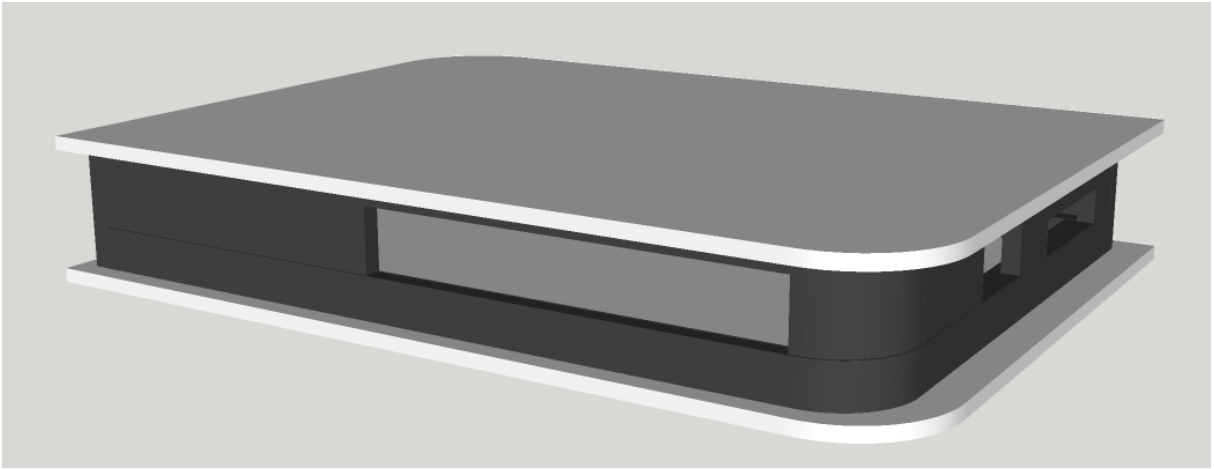
Obrázek č. 24: Pouzdro stmívače v programu Sketch Up - 1



Obrázek č. 25: Pouzdro stmívače v programu Sketch Up - 2



Obrázek č. 26: Pouzdro stmívače v programu Sketch Up 3

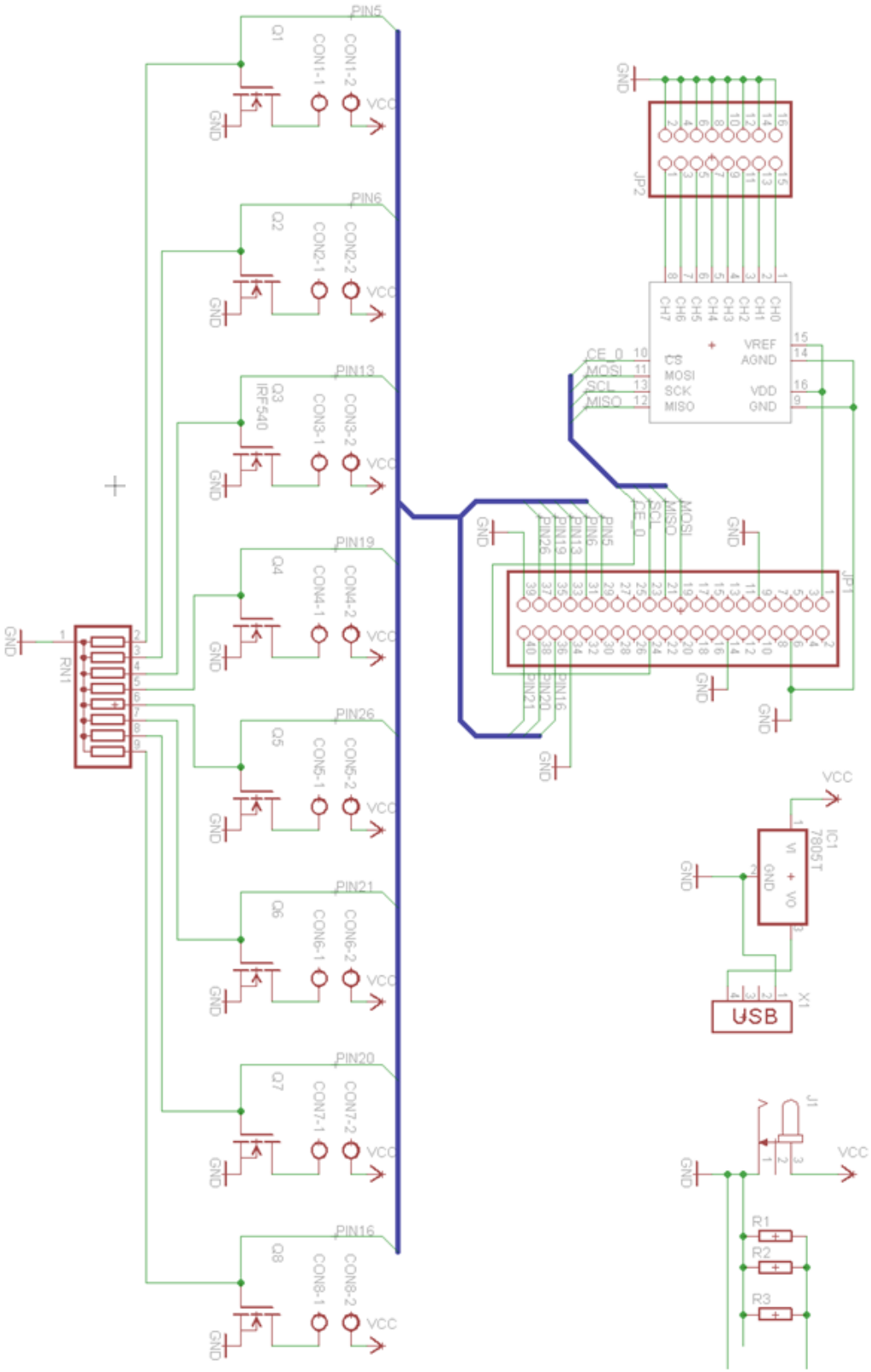


Obrázek č. 27: Pouzdro stmívače

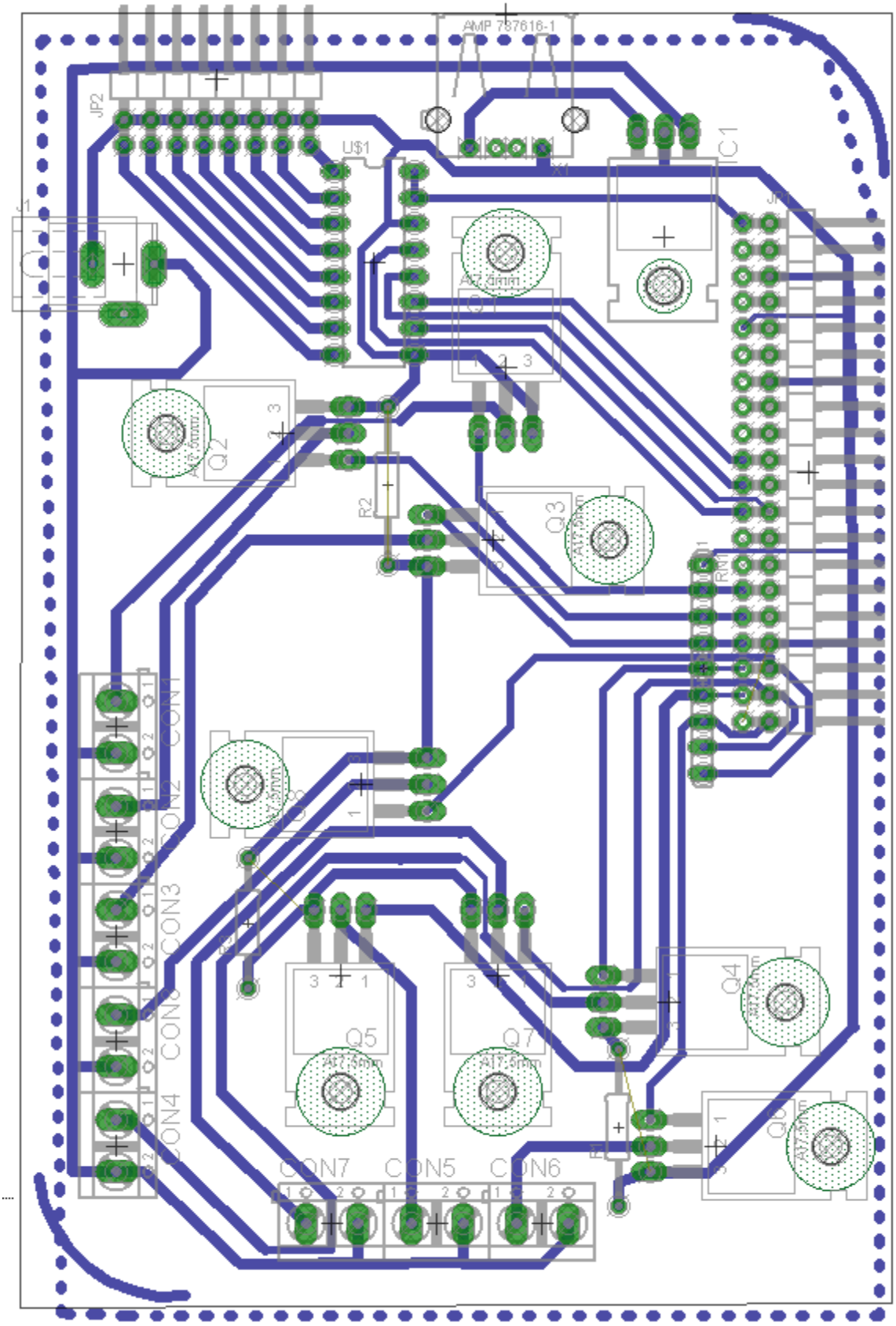
12. DPS

Schéma i návrh desky plošného spoje byly vytvořeny v programu Eagle.

Obrázek č. 28: Schéma stmívače



Obrázek č. 29: Osazovací plán stmívače



13. Závěr

Stmívač je zatím kvůli absenci lepšího chlazení určen pouze pro regulaci osvětlení s nižším výkonem. Maximální otestovaný výkon na jeden pokoj je 24 Wattů.

Přestože jsem narazil na tolik problémů, byla to příjemné zkušenost. Zdokonalil jsem se v jazyce QML, avšak musím se přiznat, že od té doby co jsem napsal stmívač, jsem získal další zkušenosti a nyní bych ho napsal čistějším způsobem.

Další výzvou pro mě bylo vytvořit aplikaci pro všechny tři platformy, přestože jsem doposud uměl vytvořit aplikaci pouze na Android.

Osobně si myslím, že kdyby se zlepšil vzhled pouzdra, mohl by můj stmívač tvořit silnou konkurenci na trhu.

14. Zdrojové soubory

15. Seznam použité literatury a studijních materiálů

[1] Datasheet k IRL640
http://www.vishay.com/docs/91305/91305.pdf [online]

[2] Datasheet k MCP3008
https://www.adafruit.com/datasheets/MCP3008.pdf [online]

[3] Raspberry Pi – SPI komunikace s MCP3008
http://jeremyblythe.blogspot.cz/2012/09/raspberry-pi-hardware-spi-analog-inputs.html
[online]

[4] Qt dokumentace - QML
http://www.qt.io/ [online]

16. Zdrojový kód

https://github.com/SonyPony/Dimmer/tree/develop