
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV INTELIGENTNÍCH SYSTÉMŮ**
**DEPARTMENT OF INTELLIGENT SYSTEMS**


# DOCHÁZKOVÝ SYSTÉM - KOMPLETNÍ ŘEŠENÍ S MODULY ESP A RASPBERRY PI

**ATTENDANCE SYSTEM - COMPLETE SOLUTION WITH ESP AND RASPBERRY PI MODULES**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** **PETR KŘEHLÍK**
**AUTHOR**

**VEDOUCÍ PRÁCE** **Doc. Ing. VLADIMÍR JANOUŠEK, Ph.D.**
**SUPERVISOR**



**BRNO 2020**

**Vysoké učení technické v Brně**
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2019/2020

# Zadání bakalářské práce


Student: **Křehlík Petr**
Program: Informační technologie
Název: **Docházkový systém - kompletní řešení s moduly ESP a Raspberry Pi**
**Attendance System - Complete Solution with ESP and Raspberry Pi Modules**
Kategorie: Vestavěné systémy

Zadání:

1. Seznamte se s platformami Raspberry Pi, Arduino a moduly ESP. Prostudujte způsoby realizace systémů pro kontrolu docházky.
2. Navrhněte a realizujte přístupový bod za pomocí Arduino, ESP modulu, displeje a RFID čtečky. Povolení přístupu se ověřuje na základě RFID čipu (popř. zadání hesla pro dvoufázové ověření). Data se získávají pomocí WiFi připojení z Raspberry Pi a veškerá data o používání se posílají zpět.
3. Navrhněte a realizujte způsob ukládání dat a webové rozhraní pro správu systému za pomocí Raspberry Pi. Systém slouží pro kontrolu přístupů, správu uživatelů apod. Uživatelé v systému mají několik úrovní (např. zaměstnanec, správce apod.) a podle toho jim může i nemusí být umožněn fyzický přístup u přístupového bodu. Navrhněte aplikační rozhraní pro integraci do většího systému.
4. Navrhněte způsob testování vytvořeného systému, testování proveďte a vyhodnoťte výsledky. Navrhněte možnosti budoucích rozšíření systému.

Literatura:

- Espressif Systems. URL: https://www.espressif.com/
- Raspberry Pi. URL: https://www.raspberrypi.org/

Pro udělení zápočtu za první semestr je požadováno:

- Bod 1 a návrh bodů 2 a 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Janoušek Vladimír, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 31. července 2020
Datum schválení: 31. října 2019

# Abstrakt

Cílem této bakalářské práce je navržení a implementace docházkového systému založeného na platformě Arduino, ESP a RaspberryPi, které spolu komunikuji pomocí WiFi sítě. Přístupová stanice poskytuje přístup uživatelům pomocí RFID čipů a centrální jednotka informační systém pro správu celého systému v podobě webového rozhraní.



# Abstract

The main goal of this bachelor's thesis is to design and implement an attendance system based on the Arduino, ESP and RaspberryPi platforms, which communicate with each other using a WiFi network. The access station provides users access using RFID tokens and a central unit information system for managing the entire system in the form of a web interface.



# Klíčová slova

raspberrypi, esp, rfid, informační systém, arduino, docházkový systém, nette, latte, api

# Keywords

raspberrypi, esp, rfid, information system, arduino, attendance system, nette, latte, api


# Citace

KŘEHLÍK, Petr. *Docházkový systém - kompletní řešení s moduly ESP a Raspberry Pi.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. Vladimír Janoušek, Ph.D.

# Docházkový systém - kompletní řešení s moduly ESP a Raspberry Pi

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. Ing. Vladimíra Janouška Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Petr Křehlík
13. července 2020


## Poděkování

Tímto chci poděkovat Doc. Ing. Vladimíru Janouškovi Ph.D. za pomoc při psaní této práce.

# Obsah

# Kapitola 1

# Úvod

V dnešní době každá firma od těch největších po nejmenší potřebuje přehled o zaměstnancích, proto je potřeba vytvořit jednoduchý systém. Systém by neměl zatěžovat zaměstnance ani zaměstnavatele a měl by poskytovat komplexní řešení všech požadavků. Široké uplatnění může najít i v domácnosti, kde může sloužit jako část chytré domácnosti.
Cílem projektu je navrhnout jednoduchý systém na bázi vývojových desek Arduino, ESP a RaspberryPi, který bude umožňovat evidenci příchodů a odchodů zaměstnanců, resp. členů domácnosti. Evidence uživatelů probíhá pomocí RFID čipů na přístupových stanicích. Těchto stanic může být v systému více, a jsou primárně řízeny vývojovou deskou Arduino a ESP. Tyto desky komunikují s centrální jednotkou, kterou představuje RaspberryPi. Přístupová stanice může fungovat jen jako evidenční stanice, nebo také umožňovat otevření dveří, pokud má uživatel dostatečná oprávnění.
Správu evidence a uživatelů zajišťuje informační systém, který se nachází na centrální jednotce, resp. RaspberryPi.
V rámci 1. bodu zadání, tedy nastudování literatury k tématu práce, jsem nastudoval, jak fungují dané technologie hlavně z katalogových listů zařízení a součástek. Dále jsem se seznámil s existujícími řešeními na trhu a popsal je v další kapitole.

## 1.1 Existující řešení

Existujících řešení je celá řada, ať už založená na podobných technologiích jako toto nebo úplně odlišných. Většina komerčních řešení je velice komplexní a obsahuje přístupové stanice, aplikace, informační systém s širokou škálou nastavení a další. Naproti tomu ale jejich pořizovací a poté i provozní náklady jsou mnohem vyšší.
Pro porovnání jsem zkoumal docházkový systém Aktion dostupný na adrese https://www.aktion.cz/produkty/dochazkovy-system.html. Ten poskytuje přístupové terminály na čip nebo biometrické senzory. Má kompletní aplikaci pro mobil, tablet desktop. Systém má možnosti obrovského rozšíření různými moduly. Je k němu poskytována kompletní technická podpora. Nicméně nic z toho není levné, např.: jeden přístupový terminál který má malý dotykový displej a bezkontaktní čtečku čipů stojí 15900 Kč + 490 Kč měsíční poplatek.

## 1.2 Motivace

Jak již bylo zmíněno v předchozí kapitole, komplexní řešení může být velice drahé, a pro menší společnosti a domácnosti není vůbec výhodné. Cílem je tedy přinést řešení, které bude příznivé cenou a zároveň poskytovat všechny potřebné funkce.
V případě řešení, popsané v této práci, jsou nutné jen relativně nízké počáteční náklady, které by neměly přesáhnout 2000 Kč. Provozní náklady jsou vzhledem k spotřebě elektronických součástek a údržbě velice nízké.
Další výhodou je, že celé řešení využívá licenci MIT. Ta umožňuje komukoliv upravit řešení dle svých představ a jelikož jde o licenci open source, tak jsou veškeré zdrojové kódy veřejné a každý je může zkontrolovat, či pomoct s opravou chyb a vytvářením další funkcionality.
Ačkoliv toto řešení popisuje přesný hardware, cílem není vytvořit něco co bude fungovat jen při striktně daných podmínkách, ale cílem je řešení, které bude použitelné při jakékoliv konfiguraci, tedy nebude nijak vázáno na hardware, což do budoucna komukoliv, kdo by chtěl toto řešení použít, usnadňuje nasazení, jelikož může použít jakýkoliv hardware, který má k dispozici a je kompatibilní s technologiemi, které jsou potřebné.

# Kapitola 2

# Použité prostředky

## 2.1 Hardware

V této kapitole se budu věnovat popisu všech zařízení a součástek, která jsou v práci použita.

### 2.1.1 Přístupová stanice

Přístupová stanice je realizovaná z několika součástek, které jsou propojené pomocí propojek bez pájení.

#### Arduino a ESP

Arduino [1] je open-source vývojová deska, která se programuje pomocí programu Arduino IDE v programovacím jazyku C/C++. Díky velkému rozšíření existuje mnoho kvalitních modulů a knihoven pro práci s nimi. V tomto projektu je použito konkrétně Arduino MEGA+WiFi R3 [8] od společnosti RobotDyn. Ukázka desky je na obrázku 2.1. Tato varianta obsahuje mikroprocesor Atmel ATmega2560 a modul Espressif Systems ESP8266EX [4]. Pro převod USB-TTL se používá CH340G[1]. Ty jsou propojené pomocí UART (popsáno v kapitole 2.2.1) a režim provozu zařízení se nastavuje pomocí přepínačů na desce. To spoří místo, jelikož jsou oba moduly na jedné desce a minimalizuje problémy se spojením mezi moduly. První tři režimy jsou realizovány pomocí dvou přepínačů, vždy jeden pro TX a druhý pro RX. Deska může pracovat v několika režimech:

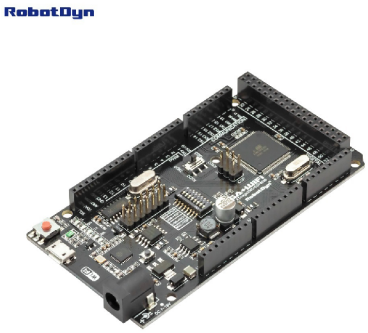
- Arduino->ESP: Přímé spojení ATmega2560 a ESP8266 pomocí UART
- USB->Arduino: Spojení USB s ATmega2560 pro sériovou komunikaci a programování
- USB->ESP: Spojení USB s ESP8266 pro sériovou komunikaci
- GND-GPIO0: Samostatný přepínač, který v kombinaci s předchozím režimem umožňuje programovat ESP8266

ATmega2560 obsahuje čtyři UART kanály a ESP8266 jen jeden. Na desce je přepínač, který umožňuje vybrat, který UART kanál má použít ATmega2560 pro komunikaci s ESP8266. Na výběr je D0 a D3.

[1] https://cdn.sparkfun.com/datasheets/Dev/Arduino/Other/CH340DS1.PDF

Desku lze napájet pomocí USB2.0 konektoru typu Micro B napětím 5V nebo pomocí externího napájení 7-16V DC. Zařízení pracuje s 5V logikou.

ESP je velice levný WiFi čip, který pracuje s TCP/IP protokolem a Wi-Fi standardy 802.11 b/g/n 2.4 GHz. K desce lze připojit externí anténu pro lepší signál.




Obrázek 2.1: Robotdyn Arduino MEGA+WIFI (převzato z [8])

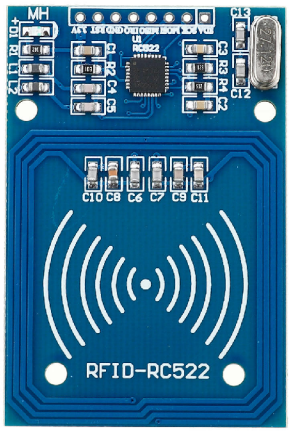
### Nextion panel

Jako displej je použit dotykový HMI panel Nextion NX4832T035 [9], který má 3.5"velkou obrazovku. Panel má vlastní mikroprocesor a dokáže fungovat zcela samostatně. Zařízení se napájí pomocí napětí 5V. S ostatními komponenty komunikuje pomocí UART rozhraní. Společnost Nextion dodává i software pro návrh GUI pro panel a programování chování. Pracuje na klasickém principu GUI, tedy prvky jsou objekty a mají události a parametry. Na tyto události pak může reagovat program, který pošle zprávu pomocí UART rozhraní. Podoba HMI panelu je na obrázku 2.2.

Obrázek 2.2: HMI panel Nextion (převzato z [9])

### RFID čtečka

Pro RFID čtečku se používá modul RC522 2.3, který se připojuje k Arduinu pomocí SPI 2.2.1 sběrnice a dokáže číst čipy o frekvenci 13.56 MHz. Tato frekvence odpovídá technologii NFC[2], která se používá např.: v mobilních telefonech. Ve většině případů ale nelze telefon použít, jelikož z bezpečnostní důvodů nové telefony generují při každém použití nový náhodný identifikátor NFC čipu.[3]




Obrázek 2.3: RFID čtečka RC522[4]

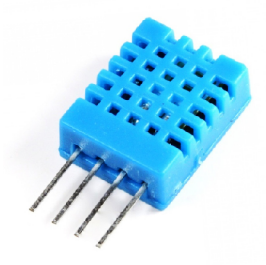
[2] https://www.svetandroida.cz/bezdratove-technologie-nfc/

[3] https://stackoverflow.com/questions/27239473/get-static-nfc-tag-id-with-hce-mode

[4] Převzato z https://www.banggood.com/cs/RFID-RC522-RF-IC-Card-Reader-Sensor-Module-with-S50-Blank-Card-and-Key-Ring-for-Arduino-Raspberry-Pi-40pin-Male-to-Female-Jumper-Wires-RFID-Tag-p-1606037.html

### Čidlo teploty a vlhkosti DHT11

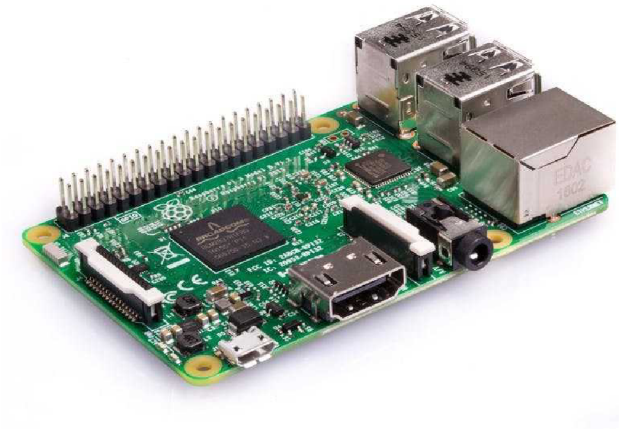
DHT11 2.4 je levné čidlo teploty a vlhkosti. Teplotu měří v rozsahu `0-50°C`;. s přesností `±2°C`. Vlhkost měří v rozsahu `20-90%RH` s přesností `±5%RH`.



Obrázek 2.4: Čidlo teploty a vlhkosti DHT11[5]

### 2.1.2 Centrální jednotka

Centrální jednotka je tvořena počítačem RaspberryPi 2.5. Ten obsahuje 64-bitový procesor Broadcom BCM2837 architektury ARM. Dále obsahuje 1GB paměti ram, 4x USB port, 1x 100 Mbit/s Ethernet a WiFi, HDMI. Operační systém a data jsou uložena na microSD kartě. Napájený je pomocí USB.



Obrázek 2.5: Počítač RaspberryPi (převzato z [6])

## 2.2 Software

### 2.2.1 Přístupová stanice

Přístupová stanice nemá žádný konkrétní software. Pro programování Arduina a ESP se používá Arduino IDE a pro HMI panel Nextion Editor.

[5]Převzato z https://www.kondik.cz/senzor-dht11

### Programování

Programování Arduina a ESP je založeno na syntaxi jazyka C++, ale používá knihovny C i C++.
HMI panel se z grafického hlediska tvoří jednoduše pomocí již zmíněného editoru metodou drag-and-drop. Grafické rozhraní používá události a metody. Samotné programování probíhá pomocí speciální instrukční sady.

### SPI

Serial Peripheral Interface (SPI) je synchronní sériová komunikace určená pro krátkou komunikaci mezi vestavěnými systémy. Rozhraní bylo vyvinuto společností Motorola v průběhu 80. let 20. století.
SPI používá full-duplex, což znamená, že komunikace může v jeden čas probíhat v obou směrech, a master-slave architekturu. Jako master je označené vždy jedno zařízení, které je pomocí vodičů připojeno k více slave zařízením. Master zařízení poté určuje s kterým zařízením bude komunikovat pomocí tzv. Slave Select (SS). Na obrázku 2.6 je příklad zapojení.
SPI používá běžně 4 vodiče:

- SCLK: Hodinový signál z Master
- MOSI: Master Output Slave Input
- MISO: Master Input Slave Output
- SS: Slave Select


Obrázek 2.6: Příklad zapojení dvou zařízení pomocí SPI[6]

### UART

Universal asynchronous receiver-transmitter (UART)[7] je prostředek pro asynchronní sériovou komunikaci, kde je možné konfigurovat formát dat a jejich rychlost.
UART používá běžně pouze dva vodiče, každý pro jeden směr komunikace. Ty jsou označené TX (odesílání) a RX (příjem). Rychlost přenosu a formát dat musí být předem nastaven na zařízeních na stejné hodnoty.

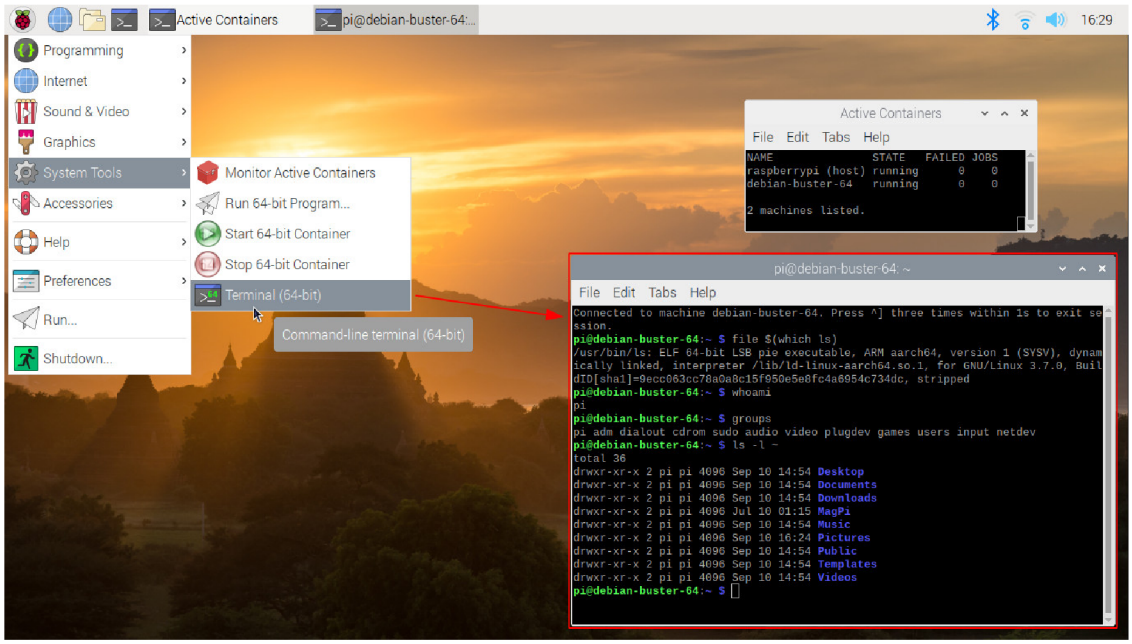
## 2.2.2 Centrální jednotka

RaspberryPi používá operační systém Raspbian [6].

[6] Převzato z https://cs.wikipedia.org/wiki/Serial_Peripheral_Interface
[7] https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter

### Raspbian

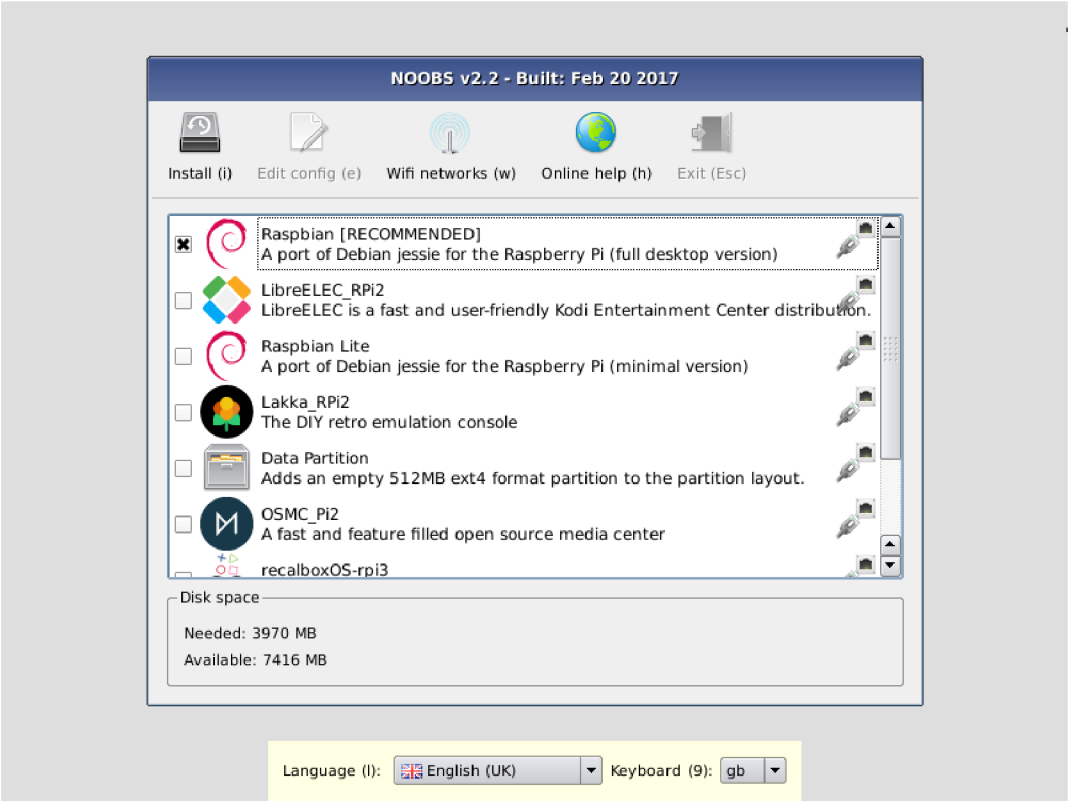
Raspbian je open source operační systém založený na operačním systému Debian GNU/Linux. Od roku 2015 je společností Raspberry Pi Foundation dodáván jako oficiální operační systém pro RaspberryPi. Hlavním cílem systému je vysoká optimalizace pro nízko-výkonové procesory založené na architektuře ARM. Systém používá jako uživatelské rozhraní PIXEL, na obrázku 2.7. PIXEL je lehké a výkonné rozhraní založené na LXDE a správci oken Openbox. Systém je dodáván s několika výchozími aplikacemi, jako například: Mathematica a Chromium, což je open source verze prohlížeče Google Chrome. Ačkoliv nemá RaspberryPi vysoký výkon, je použitelné pro běžnou kancelářskou práci, různá vestavěná řešení nebo multimediální stanice.
Raspbian existuje i ve verzi bez uživatelského rozhraní, kdy je k dispozici pouze terminál. Tato verze je určená pro serverové použití, nebo jakékoliv jiné použití, kde není potřeba grafické uživatelské rozhraní.




Obrázek 2.7: Operační systém Raspbian (převzato z [6])

Systém používá balíčkovací systém dpkg[8] a systém APT[9], pomocí kterých je možné jednoduše nainstalovat, a poté spravovat, nepřeberné množství aplikací.
Pro RaspberryPi existují i jiné operační systémy. Většina je založená na bázi Linux jako např.: openSUSE, CentOS a Fedora. Exisují ale i alternativy, které nejsou založené na bázi Linux. Mezi ně patří např.: FreeBSD, RISC OS Pi a Windows 10 IoT Core.
Pro instalaci systémů se používá New Out Of Box Software (NOOBS)[10], na obrázku 2.8. Jedná se o jednoduchý manažer, který umožňuje pomocí GUI stáhnout a nainstalovat vybraný operační systém.

[8] https://man7.org/linux/man-pages/man1/dpkg.1.html
[9] http://manpages.ubuntu.com/manpages/focal/man8/apt.8.html
[10] https://www.raspberrypi.org/documentation/installation/noobs.md




Obrázek 2.8: Grafický manažer operačních systémů na RPi (převzato z [6])

### Apache

Apache[11] je populární konfigurovatelný open-source HTTP server vyvinutý společností Apache Software Foundation v roce 1995. Apache je dostupný mimo Linux i pro celou řadu dalších operačních systémů.
Apache používá systém modulů, díky kterým lze rozšířit možnosti serveru. Mezi nejpopulárnější patří moduly pro podporu programovacích jazyků (PHP, Perl, Python a další), moduly pro autentizaci (`mod_access`, `mod_auth` a další), modul pro zabezpečení pomocí SSL/TLS[12] (`mod_ssl`), moduly pro nastavení proxy, rewrite engine pro přepis URL adresy(`mod_rewrite`) a mnoho dalších. Apache umožňuje kompresi dat pro zvýšení výkonu serveru.
Server pracuje s konkrétními servery jako samostatnými moduly. To znamená, že je možné jednoduše nakonfigurovat několik nezávislých služeb. Např.: jeden server HTTP na portu 80 a druhý nezávislý HTTPS server na portu 443.
Z hlediska výkonu se Apache řadí mezi jedno z nejvýkonnější dostupných řešení v oblasti webových serverů.

[11] https://httpd.apache.org/
[12] https://www.ictblog.cz/sifrovani-a-autentizace-co-je-to-ssl-tls-a-https/

### PHP

PHP[13] je skriptovací programovací jazyk určený primárně pro programování webových stránek. PHP se používá na straně serveru pro zpracování a poté vytvoření odpovědi. Používá se pro spojení s databází.

### MariaDB Server

MariaDB[14] je relační databáze založená na MySQL[15] vytvářená komunitou a šířená jako opensource. Hlavním cílem je vysoká kompatibilita s MySQL.

### Webový server

### HTML+CSS

HTML je značkovací jazyk určený pro definování logické struktury dokumentu. CSS je jazyk pro popis vzhledu dokumentu. Spolu s HTML tvoří základ vytváření webové stránky.

### JavaScript

JavaScript je multiplatformní skriptovací jazyk primárně určený pro vývoj webové stránky na straně klienta. Pro samotnou funkčnost není nutný, ale obecně zlepšuje zkušenost uživatelů při používání.

### Bootstrap

Bootstrap [2] je open source nástroj pro zjednodušení práce s HTML, CSS a JS. Poskytuje mnoho nástrojů a funkcí, které zjednodušují vývoj webových aplikací, jako např.: funkce pro tvorbu responzivního rozhraní.

### jQuery

jQuery[16] je malá a rychlá knihovna pro JavaScript. Obsahuje funkce, které zjednoduší práci s HTML dokumentem, jako např.: manipulaci s prvky v HTML, animace, AJAX a zpracování událostí.

### AJAX

AJAX[17] je prostředek pro neblokující asynchronní komunikaci mezi klientem a serverem. Implementuje se pomocí jazyka JavaScript. Umožňuje dynamickou změnu obsahu bez nutnosti obnovení celé stránky, čímž snižuje nároky na přenos dat a zvyšuje rychlost.

### Nette

Nette [5] je open source PHP framework, který ulehčuje vývoj webových aplikací. Jeho cílem je zajištění bezpečnosti a vytváření moderních aplikací za použití všech dostupných

[13] https://www.php.net/
[14] https://mariadb.org/
[15] https://www.mysql.com/
[16] https://jquery.com/
[17] https://www.w3schools.com/js/js_ajax_intro.asp

technologií jako např.: HTML5, AJAX a další. Framework využívá návrhový vzor MVC, což znamená, že grafické rozhraní, aplikační logika a kód pro zobrazení dat jsou oddělené.

### Latte

Je šablonový systém pro PHP, který je navržený pro práci s Nette, a poskytuje rychlé a bezpečné prostředí pro psaní šablon pro webovou stránku.

### Composer

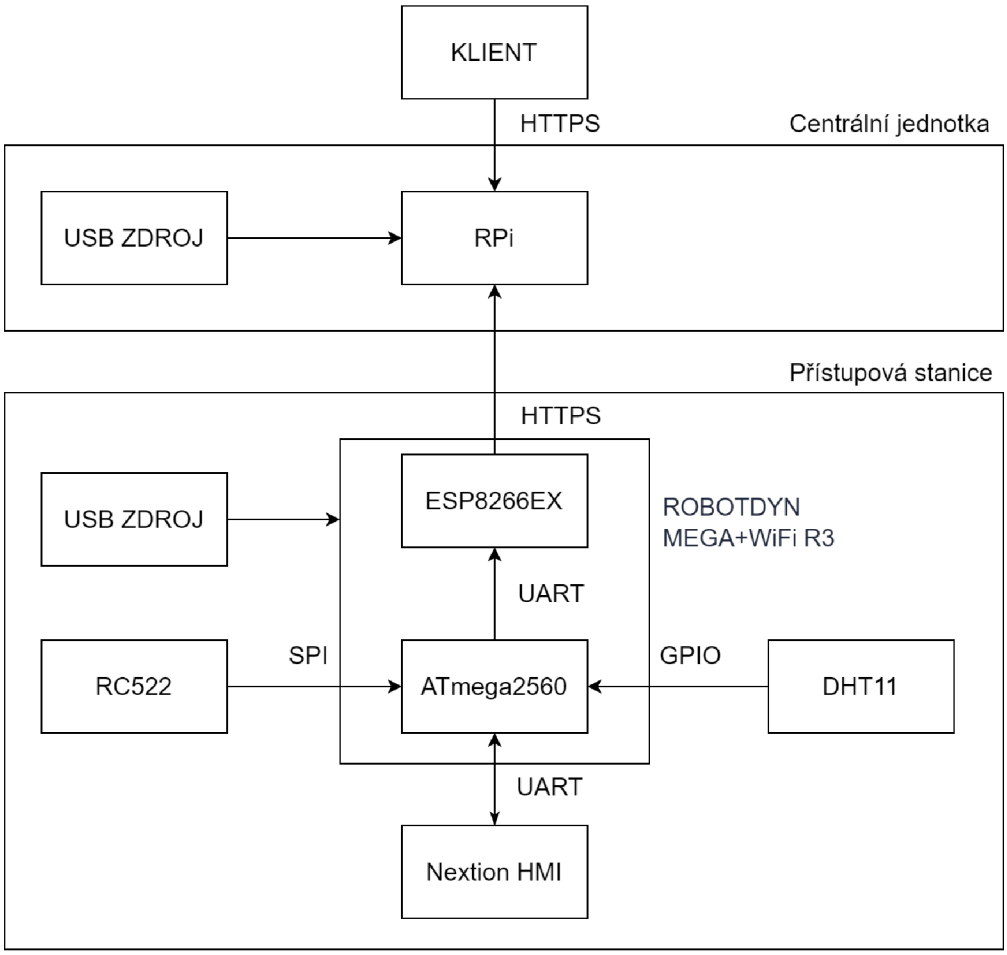
`Composer`[18] je správce závislostí pro `PHP`. Jeho hlavním úkolem je spravovat doplňky, které vyžaduje `PHP`. Díky tomu lze jednoduše přidat nebo aktualizovat jakýkoliv doplněk.

[18] https://getcomposer.org/

# Kapitola 3

# Návrh řešení a implementace

Tato kapitola popisuje návrh a provedení řešení celé práce.
Na obrázku 3.1 je finální podoba návrhu řešení. Schéma zobrazuje příklad zapojení jedné přístupové stanice. V systému jich ale může být neomezené množství a být nezávisle připojeny k centrální jednotce.




Obrázek 3.1: Blokové schéma systému

## 3.1 Přístupová stanice

Pro přístupovou stanici 3.2 jsem zvolil variantu s deskou Arduino a čipem ESP8266, které jsou sice fyzicky na jedné desce, ale logicky jsou oddělené. Toto řešení je výhodné z hlediska oddělení logiky, kdy Arduino kontroluje celou logiku aplikace a ESP má na starosti jen komunikaci se serverem. Další výhodou je velice nízká cena, kdy tato deska od výrobce Robotdyn stojí v přepočtu zhruba 310 Kč. Spolu s dalšími součástkami a krabičkou nepřesáhne v součtu 500 Kč, což je velice nízká cena, ačkoliv přichází s několika omezeními. Další variantou je použít jen vývojovou desku ESP32. Ta disponuje větším výkonem i pamětí než Arduino a ESP8266, ale má omezené množství vstupů a výstupů a cena této desky sama o sobě převyšuje 550 Kč.


Obrázek 3.2: Stanice

Dalším podstatným faktorem je spotřeba připojených periferií:

- ATmega2560 max. 200mA[1]
- ESP8266EX max. 170mA [4]
- RC522 max. 26mA[2]
- DHT11 max. 2.5mA[3]
- Nextion HMI: 145mA[4]

To je v součtu **543.5mA**. Deska Robotdyn může být napájena pomocí USB při 5V, kdy je maximální vstupní proud **500mA**. Při napájení pomocí DC konektoru je to až 800mA při 9V. Vypočtená hodnota je maximální možná hodnota dle dokumentací. Jedná se o maximální možnou hodnotu, která může nastat jen v extrémních případech a tedy není při napájení pomocí USB žádný problém. V tomto konkrétním případě je použita deska, která integruje Arduino a ESP dohromady jako jeden celek. Lze ale použít i obyčejnou samostatnou desku Arduino a tu připojit k ESP čipu v některé z jeho variant(např.: ESP-01 nebo pokročilejší ESP-12E). Ty lze poté propojit přes UART, ale kvůli rozdílnému napájení, kdy Arduino pracuje s 5V logikou a ESP s 3,3V logikou je nutné použít převodník logických úrovní. Poté lze dosáhnout stejné funkcionality jako s deskou Robotdyn. Další variantou je použít jen ESP-32, které disponuje dostatečným výkonem

### 3.1.1 Program čipu ESP8266

Úkolem ESP8266 je přijaté požadavky pomocí UART přeposlat pomocí HTTPS na předem určený server a odpověď poslat zpět pomocí UART. Prvním problémem je, že vstupní buffer UART na čipu Arduino MEGA2560 má pouze 64 bytů, takže nastal problém při přenášení delších zpráv. Ten je vyřešený jednoduše tím, že ESP vždy pošle pouze 60 bytů a ukončovací znak `\r` a poté čeká. Až Arduino zpracuje všechny znaky, pošle zpět opět znak `\r` a ESP opět pošle další data. Tím je zajištěno, že se žádná data neztratí v případě, kdy by nestihla být zpracována.
Pro připojení k WiFi je použita knihovna `ESP8266WiFiMulti` , která umožněju specifikovat více WiFi sítí a ESP si poté při zapnutí vybere tu s nejlepším signálem. Pro samotné HTTPS požadavky je použita knihovna `ESP8266HTTPClient` ve spojení s `WiFiClientSecureBearSSL` pro použití SSL. Jelikož server v rámci testování používá neověřený certifikát, tak je nastaveno, aby při komunikaci neprobíhala kontrola certifikátů.
Po připojení k WiFi tedy aktivně čeká na příjem dat pomocí UART, po přijetí dat nejprve ověří, jestli se nejedná o kontrolní znak `C`, pokud ano, tak jen odpoví kódem `OK`. Pokud se nejedná o kontrolní znak, vytvoří HTTPS požadavek. Ten vždy používá metodu GET, což nesplňuje architekturu REST[5], ale ESP nijak nepracuje s požadavky, jeho úloha je jen tzv. pošťák, takže pro jednoduchost a rychlost využívá jen tuto metodu. Po přijetí odpovědi, která je vždy typu JSON[6], ji, jak již bylo zmíněno, začne postupně posílat zpět.

[1] https://diyi0t.com/arduino-mega-tutorial/
[2] https://www.hotmcu.com/mifare-1356mhz-rc522-rfid-card-reader-module-p-84.html
[3] https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf
[4] https://robu.in/product/nextion-nx4832t035-3-5-hmi-kernel-tft-lcd-touch-display-module/
[5] https://cs.wikipedia.org/wiki/Representational_State_Transfer
[6] https://www.json.org/

### 3.1.2 Program čipu Arduino

Tento program používá řadu knihoven, jejich seznam naleznete v příloze.
Program vyžaduje mít nastavený token pro přístup k API, který se používá pro ověření při komunikaci se serverem a samotnou IP adresu a protokol serveru. API token se nastavuje v konfigurační části programu 3.3 stejně jako IP adresa a při změně je nutné znovu zkompilovat program.

```
// USER SETTINGS
#define STATION_TOKEN "h17222VeMwugFvl7"
#define MAX_USERS 40
#define IP "https://192.168.1.103"
```

Obrázek 3.3: Ukázka konfigurace programu MEGA2560 na stanici

Další nastavení je omezení na počet uživatelů pro danou stanici, to je nastaveno pro MEGA2560 na 40. Je to způsobeno tím, že deska má jen 8 kB SRAM[7].
Pro uložení dat o uživatelích se používá pole struktury, která obsahuje RFID uživatele jako pole typu `char` o velikosti 9 bajtů, PIN uživatele jako `integer` a oprávnění jako typ `byte`. Pro 40 uživatelů má tedy pole 800 bajtů. Data, která se přijmou jako odpověď od serveru jsou typu JSON, ta se pomocí knihovny[8] analyzuje a uloží do objektu `StaticJsonDocument`, který má pevnou velikost 2100 bajtů.
Globální proměnné zabírají v projektu `56%` paměti a podle analýzy IDE zbývá 3559 bajtů paměti pro další použití. Dále musíme počítat s tím, že data, která přijmeme od serveru se nejdříve uloží do objektu typu `String`, pro který tedy potřebujeme také 2100 bajtů, až poté se převedou do objektu JSON. Zbývá nám tedy 1459 bajtů a ty jsou nutné pro bezproblémovou práci lokálních proměnných. O formátu JSON dat, která generuje API, se více dočtete v kapitole 3.5.
Částečným řešením by mohlo být využití EEPROM[9] paměti. Velikost EEPROM paměti čipu MEGA2560 je 4kB. EEPROM je paměť napěťově nezávislá (nevolatilní). Její nevýhodou je ale životnost. V případě MEGA2560 se udává maximální počet přepisů 100 tisíc. Pokud by se paměť využívala pro uložení dat, která se často mění, tak se může zkrátit životnost zařízení. Paměť lze tedy použít jen pro finální pole uživatelů, které má ale nejmenší velikost, takže při použití EEPROM nedojde k žádnému markantnímu zlepšení. V případě ESP8266 není žádný problém, jelikož má SRAM o velikosti 160 kB.

Program začíná tím, že inicializuje vstupy a výstupy, restartuje HMI panel a poté pošle kontrolní znak pomocí UART čipu ESP. Pokud zaznamená kladnou odpověď, tak pokračuje dále, pokud ne, tak vyčkává a opakuje posílání zprávy každých 20 sekund, dokud neobdrží kladnou odpověď. Pokud nedojde k úspěšnému spojení s ESP, tak program nepokračuje dále.
Pro počítání času, i když není k dispozici modul reálného času, se používá funkce `millis()`, tato funkce vrací čas v milisekundách od začátku běhu programu. Algoritmus funguje tak, že když dojde k bloku, kde je třeba měřit čas, tak si uloží aktuální hodnotu a v dalších prů-

[7] https://www.arduino.cc/en/tutorial/memory
[8] https://arduinojson.org/
[9] https://www.arduino.cc/en/Reference/EEPROM

chodech programu ověřuje, jestli rozdíl aktuální hodnoty od té uložené je větší než zadaná hodnota.

Komunikace s ESP, pomocí UART, je zajišťovaná systémem fronty typu FIFO (First In First Out). Maximální velikost fronty je 10 položek. Pokud se pokusíte přidat další položku, jestliže je fronta plná, tak se jedná o kritickou chybu. Taková situace by neměla nastat, jelikož zpracování požadavků na server je vykonáváno mnohem rychleji, než uživatel dokáže akcemi generovat nové požadavky. Pokud i přesto nastane, tak to pravděpodobně znamená, že nelze nějaký požadavek vykonat. V takovém případě bude celá fronta smazána.
Pokud Arduino potřebuje vytvořit požadavek na server, použije funkci
`boolean ESPQueue_Add(String command, int type)`. Parametr `command` je URL adresa cesty požadavku, např.: `/api/get-users?` a `type` nastavuje, jak se má reagovat na odpověď. Funkce přidá požadavek na konec fronty.


```
{
  "m": {
    "s": "ok",
    "u": [
      [
        "e953345b",
        3,
        "1234"
      ],
      [
        "3aec7415",
        3,
        "1234"
      ]
    ],
    "c": "2"
  },
  "h": "fa95b3a0583a4e4763f7b81b4ad43481"
}
```


Obrázek 3.4: Příklad odpovědi serveru stanici s výpisem uživatelů

Dalším krokem k ověření integrity dat je výpočet kontrolního součtu zprávy. Kontrolní výpočet využívá algoritmus MD5[10] a vytváří se z serializovaného prvku odpovědi `"m"`. Arduino při přijetí zprávy vypočítá nový kontrolní součet a ověří s přijatým. Tím je zajištěno, že zpráva je kompletní a správná, tedy zpráva nebyla po cestě změněna např.: útočníkem. Jedná se o velice efektivní metodu pro zajištění integrity dat při přenosu.
Po úspěšné inicializaci se program dostane do hlavní nekonečné smyčky.
Při každém průchodu se zpracovává několik částí:

- Obsluha ESP fronty
- Aktualizace dat
- Kontrola přiložení RFID

[10] https://cs.wikipedia.org/wiki/Message-Digest_algorithm

### Obsluha fronty

Pokud je ve frontě nějaká položka k odeslání, tak se k GET požadavku připojí parametr tokenu stanice pro ověření a pošle pomocí UART na ESP. Zde problém s délkou vstupního bufferu čipu ESP není, jelikož má délku 256 bajtů a to je pro požadavky plně dostačující. Pokud je odeslán požadavek, tak se v každém dalším průchodu hlavní smyčky programu kontroluje, jestli nepřišla odpověď. Vždy je poslán jen jeden požadavek a dokud není přijata jakákoliv odpověď, tak se čeká na odpověď. Čekání není blokující, během odeslání požadavku a čekání na odpověď může uživatel interagovat se stanicí a mohou se do fronty přidávat další požadavky. Pokud je přijata odpověď, tak se provede syntaktická kontrola, jestli jsou data ve správném JSON formátu. Poté jsou převedena na JSON objekt a je ověřeno, jestli návratový kód je `ok` Arduino neověřuje k jaké chybě došlo, předpokládá, že API je funkční a na požadavky vždy odpoví, jak se očekává. Pokud by nepřišla žádná odpověď do 10 sekund, je požadavek z fronty odstraněn a ignorován.

### Aktualizace dat

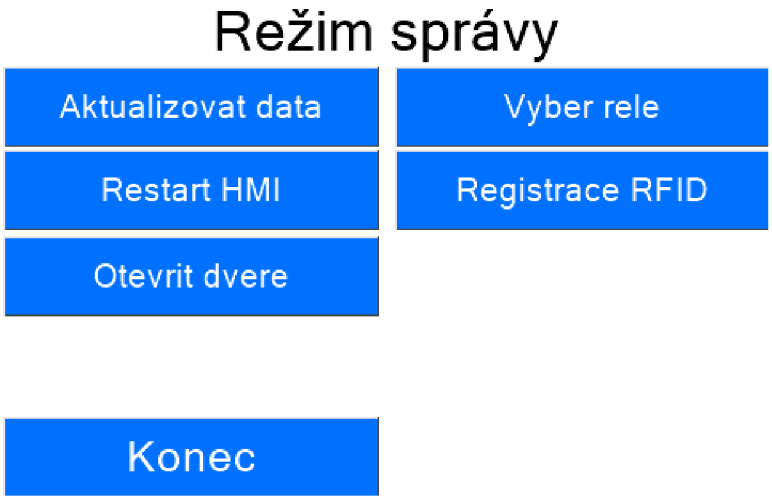
Aktualizace dat, tedy uživatelů pro tuto stanici, probíhá jednou za 10 minut. Při přijetí dat jsou provedeny standardní kontroly, jak již bylo zmíněno. Důležité je, že stará data jsou přepsána až poté co jsou úspěšně dokončeny kontroly. To znamená, že pokud by se aktualizace jakkoliv nezdařila, tak to neovlivní funkčnost a budou použita předchozí data. Jednou za hodinu také probíhá odeslání aktuální teploty z teplotního čidla do databáze.

### Kontrola přiložení RFID

Pokud je přiložen nový RFID čip ke čtečce, začne se ověřovat, zda má být umožněn přístup. Nejdříve se ověří, zda je RFID v seznamu a pokud ano, tak se ověří úroveň oprávnění. Při standardní úrovni je přímo umožněn přístup, při dvoufázovém ověření je třeba zadat čtyřmístný PIN kód na klávesnici, která se zobrazí na displeji. Pokud je PIN správný, je umožněn přístup. Poslední úrovní je správce. V tomto režimu je také vyžadováno dvoufázové ověření, ale po úspěšném ověření je spuštěn režim správce.

### Režim správce

Režim správce 3.5 umožňuje různá nastavení stanice.




Obrázek 3.5: Obrazovka režimu správy na HMI panelu

Správce může ručně aktualizovat data, tedy obnovit seznam uživatelů a jejich oprávnění, restartovat HMI panel a otevřít dveře.
Výběr relé zobrazí další obrazovku 3.6, která umožňuje nastavit jednotlivé chování každého relé. Stanice obsahuje 4 samostatné relé.

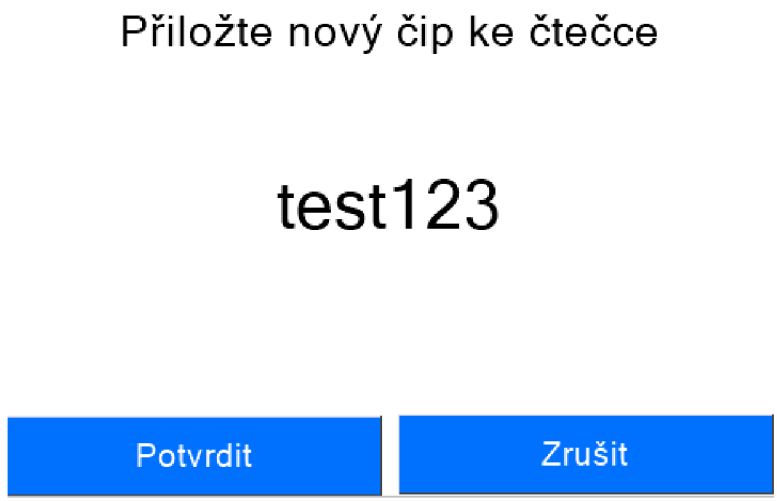

Obrázek 3.6: Obrazovka nastavení relé na stanici

Kliknutím na jednotlivá tlačítka lze relé přepínat mezi režimy, ty jsou tři.
V prvním režimu označeném - nemá relé žádnou funkci a je deaktivované.
V režimu `ZVONEK` se relé spíná při stisknutí tlačítka zvonku na úvodní obrazovce. Tlačítko zvonku se zobrazuje jen v případě, že alespoň jedno relé je v režimu zvonku. Zvonek se spíná po dobu 1000 ms.
V posledním režimu `DVERE`, funguje relé jako spínač dveří. Pokud je uživateli povolen přístup, jsou sepnuta všechna relé v tomto režimu po dobu 1000 ms.

Veškeré nastavení stanice je specifické jen pro danou stanici a není synchronizováno se

serverem. Nastavení je uloženo v EEPROM paměti čipu Arduino MEGA.
Účelem relé spínačů je jen vytvořit impulz pro další akce, resp.: spuštění zvonku nebo otevření dveří. Cílem není řešit logiku dalšího zpracování např.: jaký bude u dveří použit zámek, jak dlouho bude zvonit zvonek, atd.

Poslední položkou režimu správy je `Registrace RFID`. Tato položka zobrazí novou obrazovku 3.7 s výzvou k přiložení nového čipu ke čtečce. Pokud je načtení úspěšné, zobrazí se na displeji identifikátor nového čipu.
Po stisknutí tlačítka `Potvrdit` se pošle požadavek na uložení nového RFID do databáze.




Obrázek 3.7: Obrazovka uložení nového RFID na stanici

### 3.1.3 HMI Panel

Některé důležité instrukce:

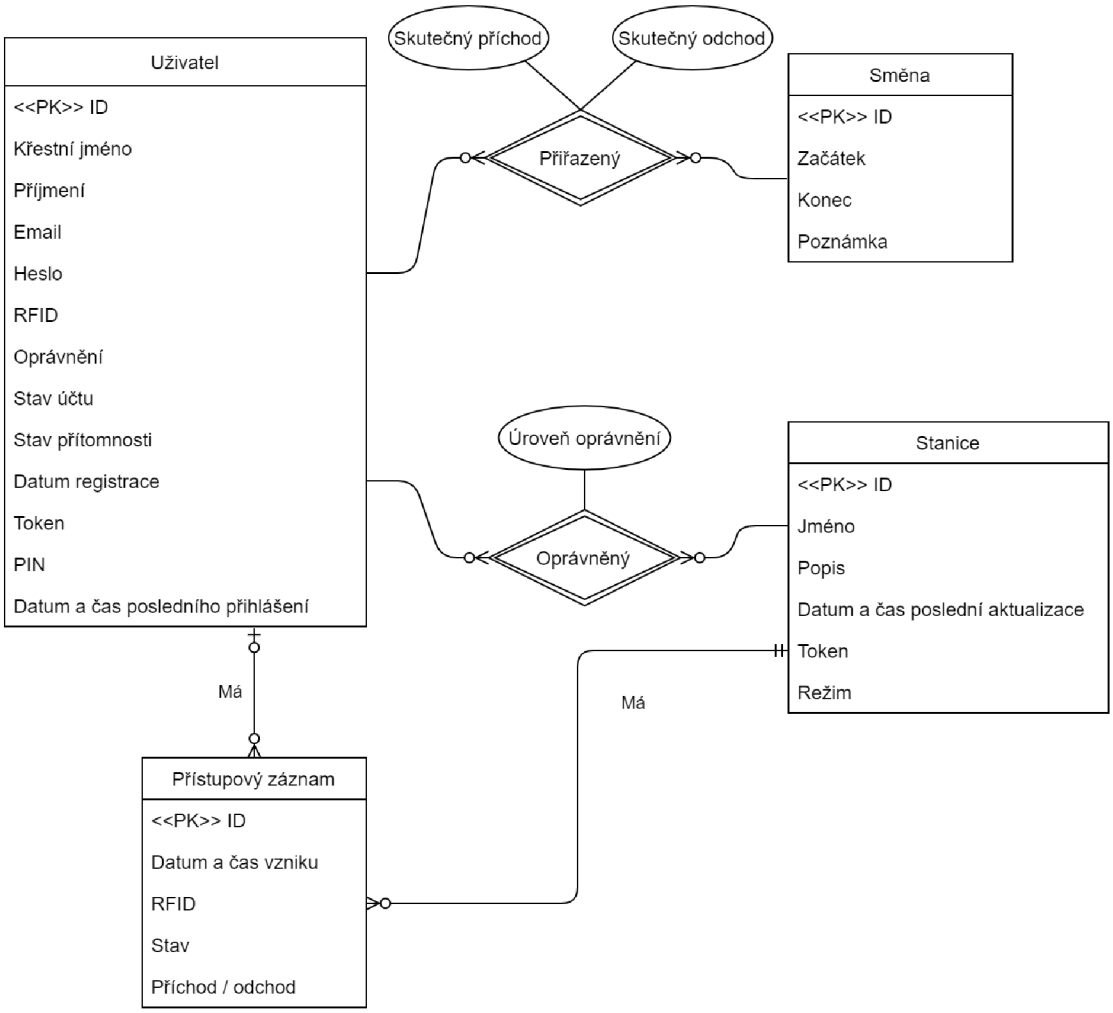
- **get <atribut>** Zašle zadanou hodnotu pomocí rozhraní UART
- **page <pid>** Změní stránku na zadanou
- **vis <komponenta><stav>** Změní komponentu na viditelnou/neviditelnou

Instrukční sada obsahuje standardní konstrukce jako např.: if, while, atd.
Tlačítka, textové elementy a další jsou komponenty. Pokud chceme např.: změnit text textové komponentě, pošleme příkaz `b1.txt=\"-\"`.
Rozdíl v chování je v případě zobrazení resp. nezobrazení tlačítka zvonku na úvodní obrazovce. V tomto případě Arduino neposílá přímo příkaz k zobrazení nebo skrytí tlačítka v závislosti na nastavení, jelikož kdykoliv by se změnila stránka na displeji, tak se toto nastavení ztratí. Pro vyřešení tohoto problému se používá na displeji globální proměnná, kterou lze nastavit kdykoli. Arduino tedy pošle příkaz ke změnění hodnoty této proměnné. Vždy při přechodu na úvodní obrazovku se HMI panel na základě hodnoty této proměnné rozhoduje, zda tlačítko se zvonkem zobrazit. Stejný princip se používá i pro zobrazení aktuální teploty. Při samotném spuštění stanice se zobrazuje stránka načítání. Arduino se pokouší

inicializovat stále dokola, i když dochází k chybě, ale tuto chybu nijak neindikuje na displeji. Proto je na stránce načítání časovač, který po uplynutí 20 sekund zobrazí stránku s chybou, aby upozornil na to, že inicializace neprobíhá správně.

## 3.2 Databáze

Následující obrázek 3.8 zobrazuje zkrácený návrh datového modelu systému. V obrázku nejsou vyobrazeny entity bez vazby.




Obrázek 3.8: Datový model

## 3.3 Zabezpečení

Zabezpečení komunikace ESP a RaspberryPi je možné provést dvěma způsoby, kdy ideální by byla kombinace obou, ale to nemusí být v praxi vždy možné.

- **Použití izolované zabezpečené WiFi sítě**
  Při tomto řešení se použije izolovaná síť s dostatečným zabezpečením, ke které se mohou připojit jen zařízení patřící do systému a ne žádní jiní klienti, kteří by mohli odposlouchávat síť a případně provádět útoky. Poté je možné přenášet data jen pomocí protokolu HTTP bez zabezpečení. Nevýhodou je větší náročnost na infrastrukturu, což nemusí být v vždy vhodné.
- **Šifrování dat**
  ESP dokáže pracovat s protokolem HTTPS s šifrováním SSL/TLS. Nejjednodušší je využít knihovnu `BearSSL`. Ta umožňuje šifrované spojení pomocí protokolu HTTPS s využitím SSL. Toto připojení zajišťuje dostatečné zabezpečení a je použito v této implementaci.

### Hesla

V databázi na serveru RaspberryPi se ukládají veškeré údaje o uživatelích a je nutné zajistit jejich bezpečnost. Nejdůležitější je zabezpečit přístupové údaje uživatelů, které tvoří kombinace emailu a hesla. Vhodné a dostatečné zabezpečení je použití nativní funkce v PHP s použitím algoritmu `BCRYPT` pro vytvoření kontrolního součtu. Tato funkce vytvoří kontrolní součet ze zadaného hesla, uloženého v databázi. Tento kontrolní součet nelze dešifrovat. Při přihlášení je vytvořen ze zadaného hesla nový kontrolní součet a porovnán s tím uloženým v databázi. V případě shody se jedná o stejné heslo a je umožněn přístup.

## 3.4 Informační systém

Infromační systém je postavený na open-source PHP frameworku Nette. Nette využívá návrhový vzor Model-view-controller (MVC), který odděluje datový model aplikace, řídíci logiku a uživatelské rozhraní do tří nezávislých modulů. O vykreslování se stará šablonovací systém Latte. Mezivrstva, tedy Controller v kontextu MVC, je tzv. presenter, který má na starosti řízení a předávání dat mezi datovou vrstvou a uživatelským rozhraním, tedy Latte. Výhodou toho řešení je zjednodušení práce, jelikož framework se postará o mnoho záležitostí. Má pokročilé ladící nástroje, výbornou bezpečnost, širokou kolekci doplňků a další funkce.

### 3.4.1 Struktura projektu

- **app** Obsahuje samotnou aplikaci.
- **log** Logy systému.
- **temp** Dočasné soubory, cache.
- **vendor** Knihovny a doplňky.
- **www** Veřejný adresář obsahující grafické rozhraní a vstupní soubor index.php.

Složka `app`:

- **Api** Apitte aplikace zajišťující API.
- **config** Konfigurační soubory NEON.
- **Controls** Obsahuje vlastní komponenty a šablony pro DataGrid.
- **lang** Obsahuje NEON soubory s překlady.
- **MainModule** Obsahuje šablony a presentery hlavní aplikace.
- **Models** Obsahuje soubory pro ORM, služby a další pomocné třídy.
- **Presenters** Obsahuje šablony a presentery pro obsluhu chyb.
- **Router** Obsahuje třídu zajišťující zpracování URL.
- **Security** Obsahuje třídu zajišťující přihlášení uživatelů.
- **VisitorModule** Modul zajištující přihlášení, registraci a obnovu hesla.

### 3.4.2 Konfigurace

Konfigurační soubory typu NEON obsahují nastavení doplňků, připojení na databázi, cache, session, překladů a další.
Soubor Bootstrap.php obsahuje nastavení produkčního režimu, ladění pomocí Tracy, časového pásma a další.

### 3.4.3 Struktura aplikace

Celý server se dělí na dvě hlavní části. Pokud je požadován informační systém, tak se použije `Nette` aplikace. Druhá varianta je API (aplikační rozhraní). V tomto případě se použije `Apitte` aplikace.
Základní struktura aplikace využívá PHP dědičnost pro vytvoření jednoduché struktury aplikace. Nicméně existuje několik dalších tříd (služby), které aplikace využívá. Služby jsou ke třídám "připojeny" pomocí `Dependency Injection (DI)` 3.9. Podstatou Dependency Injection (DI) je odebrat třídám zodpovědnost za získávání objektů, které potřebují ke své činnosti (tzv. služeb) a místo toho jim služby předávat při vytváření. Služby mohou být předány např.: pomocí konstruktoru. Služba DI se postará o předání instance dané třídy.

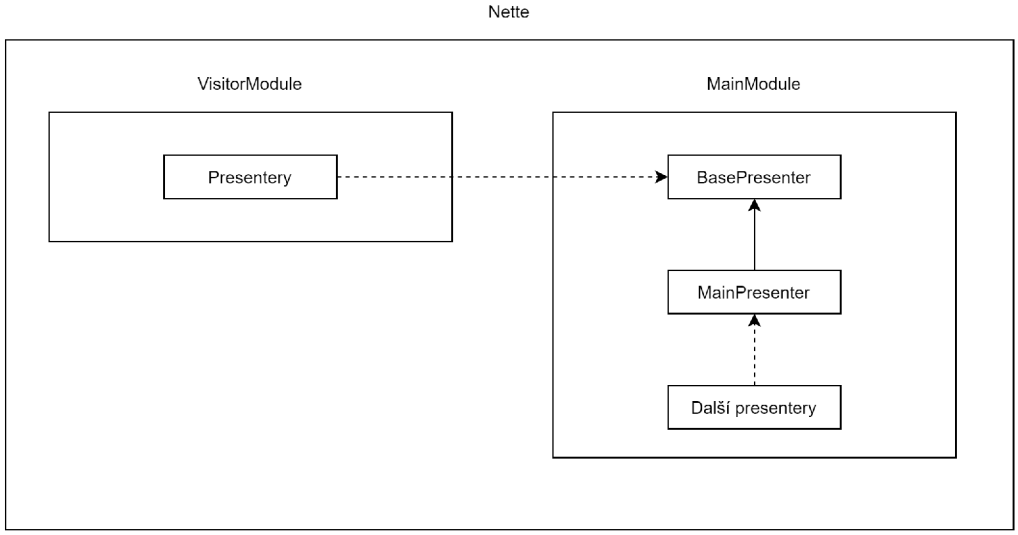
```
/** @var Orm  */
private $orm;

public function __construct(Orm $orm)
{
    $this->orm = $orm;
}
```

Obrázek 3.9: Ukázka předání služby třídě pomocí DI

Hlavní Nette aplikace se dělí na dva moduly.

- **VisitorModule**
  Tento modul má na starosti přihlášení, registraci a obsluhu zapomenutého hesla.
- **MainModule**
  Hlavní modul má na starosti všechny operace pro přihlášené uživatele.

Obrázek 3.10 ukazuje propojení jednotlivých presenterů v aplikaci.




Obrázek 3.10: Struktura informačního systému

`BasePresenter` má na starosti obsluhu lokalizace. Poskytuje funkce pro další presentery pro překlad a funkci pro změnu jazyka. Obsahuje také funkce pro zobrazení notifikací, které využívají knihovnu `iziToast`. Knihovna `iziToast` poskytuje mnoho možností nastavení. Je možné notifikaci zobrazit po načtení stránky a nebo ihned pomocí AJAX. V tomto případě je nutné ručně překreslit snippet obsahující skript notifikace. Snippet je prvek Latte šablony, který může být pomocí AJAX požadavku překreslen. Tento presenter využívají všechny další presentery.
`MainPresenter` využívají všechny presentery v `MainModule`. Tento presenter poskytuje funkce pro kontrolu oprávnění a obsluhuje zobrazení notifikací pro manažery a správce.

### 3.4.4 Nittro

Celá Nette aplikace využívá doplněk `Nittro`. `Nittro` je klientský framework vytvořen pomocí jazyku Javascript. Je určen pro práci s Nette. Hlavním účelem je aplikovat na celou aplikaci AJAX. Díky Nittro je jakýkoliv odkaz zpracováván jako AJAX požadavek a poté překreslena jen určitá část stránky. AJAX je asynchronní požadavek, který si od serveru, v kombinaci s Nittro, vyžádá část stránky určenou k obnovení. Server provede zpracování požadavku a vrátí jen tu část, kterou klient požadoval. Výsledkem je úspora při přenosu dat a pro uživatele lepší UX (User Experience). Nittro má další funkce jako např.: animace při změně stránky, validace formulářů, Flash zprávy a další. Díky tomu není nutné při přechodu mezi stránkami přenášet CSS a JS knihovny a soubory znovu, ale jen nový obsah.

### 3.4.5 Nextras ORM

Informační systém pracuje primárně s databází. Aplikace ve většině případů nepracuje s databází na úrovni psaní SQL dotazů, ale využívá princip ORM [7]. Ten spočívá v tom, že relační databázi mapuje na entity. Entita 3.11 je třída, která má mapované atributy vůči sloupcům tabulky v databázi. Tyto atributy se nastavují pomocí komentáře `@property`. Nastavuje se jméno a typ, který je poté ověřován. Lze nastavit i výchozí hodnoty, klíče, výčtový typ, vazby a další. Podporované jsou všechny vazby jako např.: 1:M, M:N, a další.

```
/**
 * @property-read int           $id {primary}
 * @property DateTimeImmutable  $datetime {default NOW}
 * @property string             $logRfid
 * @property int                $status {default 0} {enum self::ACCESS_*}
 * @property Station            $idStation  {m:1 Station, oneSided=true}
 * @property User               $idUser  {default NULL} {m:1 User, oneSided=true}
 * @property int                $arrival {default NULL} {enum self::ARRIVAL_*}
 */
class AccessLog extends Entity
{
    const ACCESS_DENIED = 0;
    const ACCESS_GRANTED = 1;

    const ARRIVAL_FALSE = 0;
    const ARRIVAL_TRUE = 1;
}
```

Obrázek 3.11: Ukázka definice entity

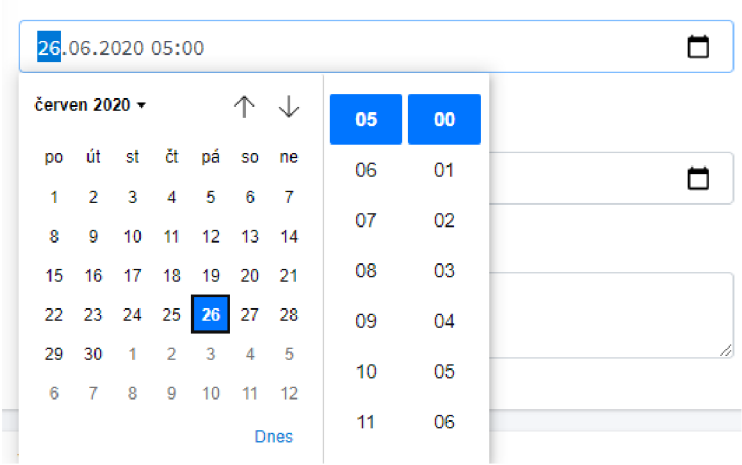
Pokud chceme s ORM pracovat, stačí pomocí DI vyžádat model ORM. Tento model poskytuje přístup k repositářům jednotlivých entit. Repositář je sada funkcí pro práci s danou entitou (tabulkou). Pokud je v tabulce nastaven cizí klíč jako "odkaz" na jiný záznam a je takový atribut entity nastaven jako vazba na jinou entitu i v ORM, tak se automaticky postará, aby byl poskytnut skutečný objekt, na který cizí klíč "odkazuje". Pokud by jsme takovou vazbu chtěli změnit, můžeme použít klíč i objekt. ORM se postará o správné zpracování v obou případech.
Repositář používá standardní funkce jako např.: getBy, findAll a další pro hledání a výběry z tabulky. Pokud chceme získat všechny záznamy z tabulky stačí použít příkaz
`$data=$this->orm->users->findAll()->fetchAll()`
Tímto řádkem získáme pole objektů entity `Uživatel`. Pokud chceme jakýkoliv atribut upravit stačí ho přímo v objektu změnit a poté provést uložení pomocí
`$this->orm->users->presistAndFlush($entita)`
Problémem při filtrování je funkce LIKE, kterou využívá komponenta DataGrid. Výchozí metoda findBy nepodporuje přímé použití LIKE filtrování. Bylo tedy nutné vytvořit `LikeFilterFunction`. Tato funkce zajišťuje správné použití LIKE filtrování. Příklad použití:
`findBy([[LikeFilterFunction::class, $klíč, $hodnota]])`, kde klíč je název sloupce v tabulce a hodnota je text pro hledání.
I když poskytuje ORM komplexní práci s databází, existují situaci, kdy použití ORM není vhodné. Jedná se především o situace, kdy v komponentě DataGrid je třeba zobrazit různé sloupce na základě cizích klíčů. ORM sice umožňuje k těmto datům přistoupit, ale z pohledu DataGrid je lepší mít k dispozici jednoduché dvourozměrné pole hodnot. V tomto případě se používá `Nette Database Explorer`. Jedná se o integrovanou součást Nette, která umožňuje jednoduchou práci s databází obdobně jako ORM, ale na výstupu nejsou objekty reprezentující entity.

### 3.4.6 Nextras DataGrid

Nextras DataGrid [7] je komponenta pro zobrazení dat. Jedná se o komplexní komponentu poskytující vše co potřebujete jako např. filtrování, řazení, stránkování, akce pro řádky, inline editace a další. Alternativou je Ublaboo/DataGrid. Ten je sice jednoduší pro práci při vývoji, ale nenabízí složitější možnosti přizpůsobení.
Výhodou, možná i nevýhodou, je že nestačí poskytnout jako zdroj dat tabulku nebo repositář. Je třeba ručně definovat funkci, která má k dispozici objekt filtru, řazení a stránkování. Jelikož získání dat se zpracováním všech zmíněných objektů se opakuje napříč celou aplikací, tak byla vytvořena služba DataGridFactory. Tato služba obsahuje metody pro jednoduché vytváření DataGrid komponent. Nejdůležitější funkce:

- **createDataGrid()**
  Vytvoří DataGrid objekt a nastaví šablonu, stránkování a překladač.
- **createDataSource()**
  Funkce pro vytváření dat na základě jména repositáře. Funkce dokáže brát v potaz filtrování, řazení a stránkování.
- **createDataSourceNotORM()**
  Podobná funkce jako createDataSource(), ale nepoužívá ORM. Umožňuje výběr dat z více tabulek.

Všechny DataGrid komponenty využívají stejnou Bootstrap šablonu. Ta byla lehce upravena, aby vyhovovala vzhledu ostatních komponent. Dále využívá každá svou vlastní specifickou šablonu, která definuje vzhled sloupců, vlastní vykreslování buněk, akce pro řádky a další.
Pro filtrování a inline editaci se používá `Nette Forms`. Jedná se o komponentu usnadňující práci s formuláři. Obsahuje mnoho prvků jako např.: tlačítka, textová pole, výběrové pole a další. Neobsahuje ale pole pro práci s daty a časem. Pro toto byl použit doplněk `voda/date-input` 3.12. Ten poskytuje komponentu pro výběr data a času ve stylu Bootstrap.




Obrázek 3.12: Výběr data a času

Dále bylo třeba vytvořit komponentu, která dovolí výběr intervalu. Pro toto jsem vytvořil

vlastní `ExtendedForm` a `ExtendedFormContainer` (pro DataGrid). Ten obsahuje metodu pro vytvoření komponenty skládající se ze dvou vyběrových polí pro výběr intervalu data a času.

### 3.4.7 Lokalizace

Informační systém je lokalizován ve dvou jazycích. K dispozici je čeština a angličtina. Lokalizaci zajišťuje doplněk `Kdyby/Translation`. Pokud není vybrán jazyk, používá se čeština. Tento doplněk má na starosti vybrat na základě URL jazyk překladu. Příklad URL pro angličtinu:
`https://.../en/main/homepage/my-shifts`. Samotné překladové fráze jsou uloženy v souborech typu NEON 3.13 ve formátu klíč, hodnota.

```
name: "Docházkový systém"
signin: "Přihlásit se"
```

Obrázek 3.13: Příklad souboru NEON s překlady

Použití 3.14 v šabloně probíhá pomocí globálního makra a v PHP jsou definovány funkce v `BasePresenter`.

```
$this->translate("yesB"); // PHP

{_messages.main.profile.settings} // Latte
```

Obrázek 3.14: Příklad použití překladů

### 3.4.8 Úrovně uživatelů

Systém pracuje se čtyřmi úrovněmi uživatelů. Úrovně jsou hierarchické, tedy vyšší úroveň má vždy všechny oprávnění nižší úrovně.

- **Neregistrovaný**
  Neregistrovaný uživatel nemá v systému žádné oprávnění kromě registrace.
- **Registrovaný**
  Registrovaný a schválený uživatel má základní přístup k systému. V kontextu malé společnosti se jedná o zaměstnance. Může zobrazit své údaje, upravovat je, zobrazit své oprávnění na stanicích a směny.
- **Manažer**
  Manažer má pravomoc spravovat uživatele nižších úrovní. V kontextu malé společnosti se jedná o vedoucího. Může plánovat směny a nastavovat oprávnění na stanicích.
- **Správce** Správce má nejvyšší úroveň v systému. Může upravovat všechny uživatele včetně ostatních správců. Jedná se o administrátora systému, který má neomezené oprávnění.

Následující obrázek 3.15 zobrazuje případy užití všech rolí v systému.


Obrázek 3.15: Diagram případů užití jednotlivých úrovní uživatelů

### 3.4.9 Přihlašování

Pro použití informačního systému je nutné se přihlásit. Pro přihlášení je potřeba kombinace emailu a hesla. Aby se uživatel mohl přihlásit, je nutné, aby byl jeho účet schválený manažerem nebo správcem. Pokud uživatel zapomene heslo, může zadáním emailu, heslo obnovit. Po potvrzení přijde uživateli email s novým dočasným heslem. Je k dispozici možnost trvalého přihlášení trvající 14 dní. Standardní přihlášení vyprší po 30 minutách bez aktivity uživatele.

### 3.4.10 Proces registrace

Proces registrace začíná tím, že uživatel vyplní registrační formulář. Tím se vytvoří v systému nový účet, který ale není aktivní a nemůže být použit pro přihlášení. Manažer nebo správce poté schválí účet a poté se uživatel může přihlásit.

### 3.4.11 Proces přiřazení RFID čipu

Aby mohl být RFID čip přiřazen uživateli je nutné čip v systému registrovat. To může provést pouze správce fyzicky pomocí stanice. Na stanici v režimu správce zvolí možnost `Registrace RFID`. Přiloží nový čip a stiskne tlačítko `Potvrdit`. Pokud již RFID v systému existuje, nebude čip přidán. V opačném případě je přidán a připraven k přiřazení. Samotné přiřazení může provést i manažer.
To probíhá pomocí možnosti v informačním systému: Modul manažera -> Správce uživatelů. Zde se vybere uživatel a zvolí možnost Přiřadit RFID. Poté stačí vybrat z tabulky RFID a potvrdit. Tím má uživatel přiřazený osobní RFID čip.

### 3.4.12 Poštovní služba

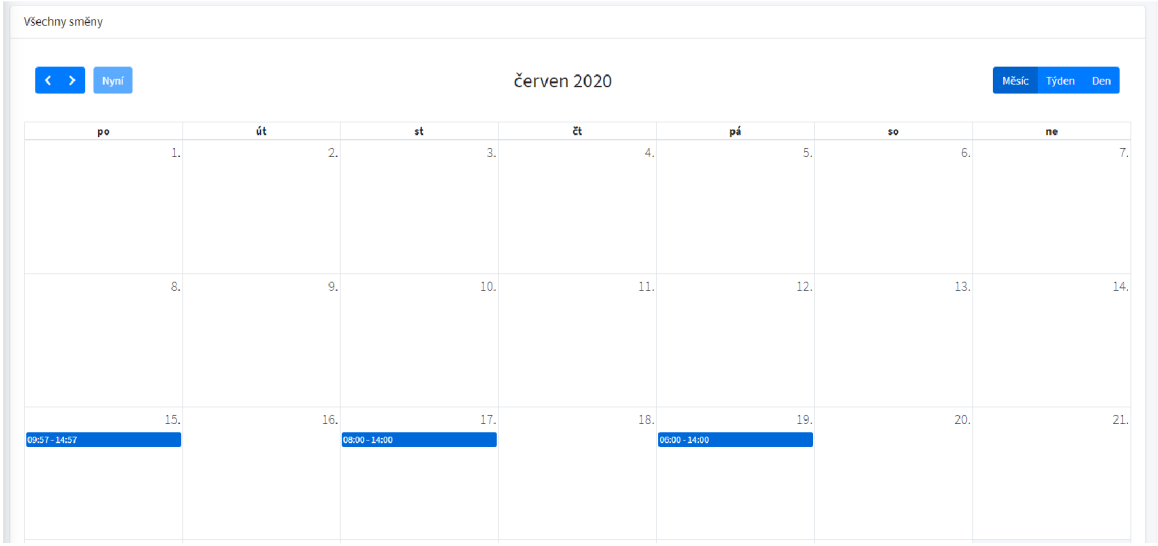
Systém dokáže odesílat emaily pomocí serverů společnosti Google. Využívá pro to linuxovou utilitu `msmtp`. Pro odesílání využívá protokol SMTP. Odesílání neprobíhá přímo v místě, kde je potřebné, ale pomocí fronty, aby uživatel nebyl zdržován čekáním na odeslání emailu. Jako fronta slouží tabulka v databázi. Obsluhu této fronty provádí crontab úloha každou minutu. Obsluhu fronty zajišťuje API. Pokud se nezdaří email odeslat po pěti pokusech, tak je z fronty smazán.

### 3.4.13 Systém kontroly uživatelů

U uživatelů se sleduje zda jsou fyzicky přítomní v budově, tedy poslední záznam je příchod. Potom lze jednoduše zobrazit aktuálně přítomné uživatele, stejně jako historii příchodů a odchodů.

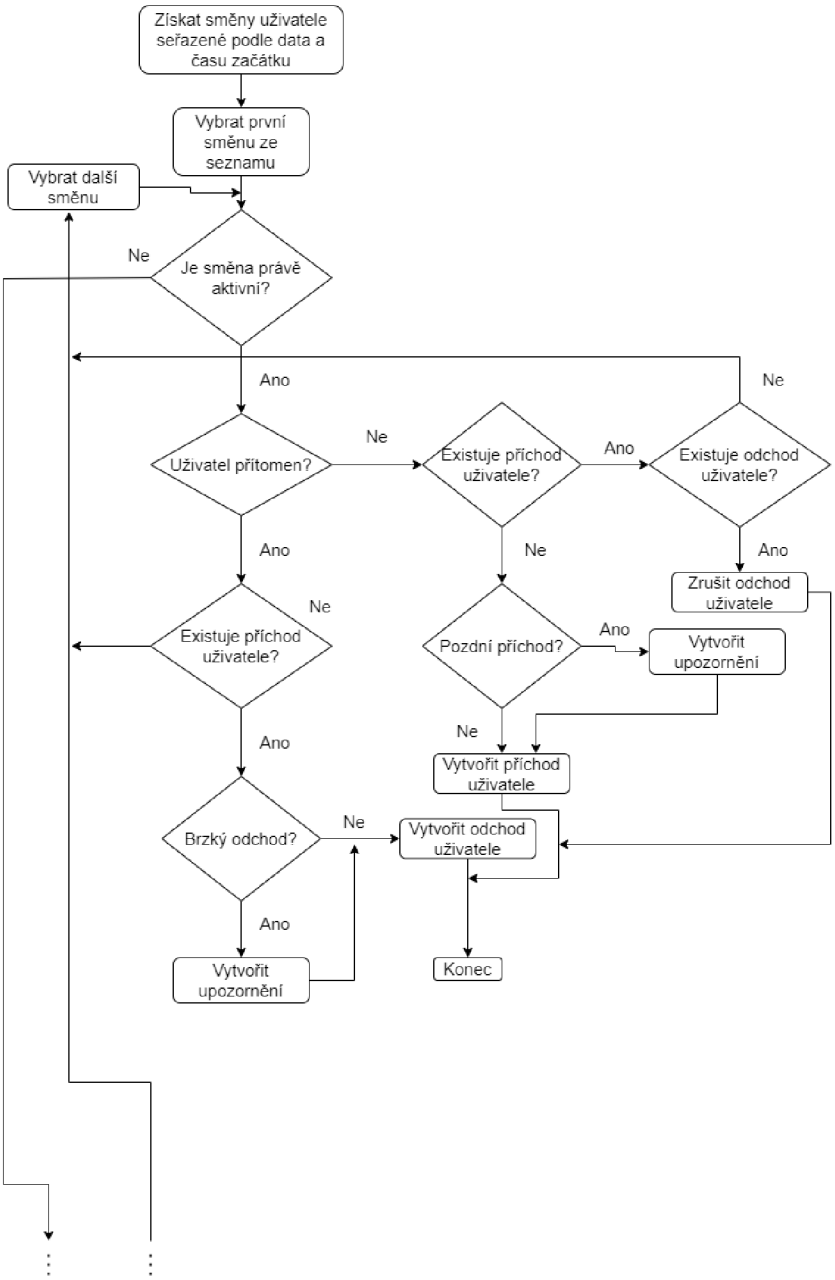
### 3.4.14 Systém směn

Informační systém umožňuje manažerům a správcům plánovat směny pro zaměstnance. Směny jsou zobrazeny jako kalendář 3.16 pomocí doplňku FullCalendar pro přehledné zobrazení a také detailně pomocí DataGrid.

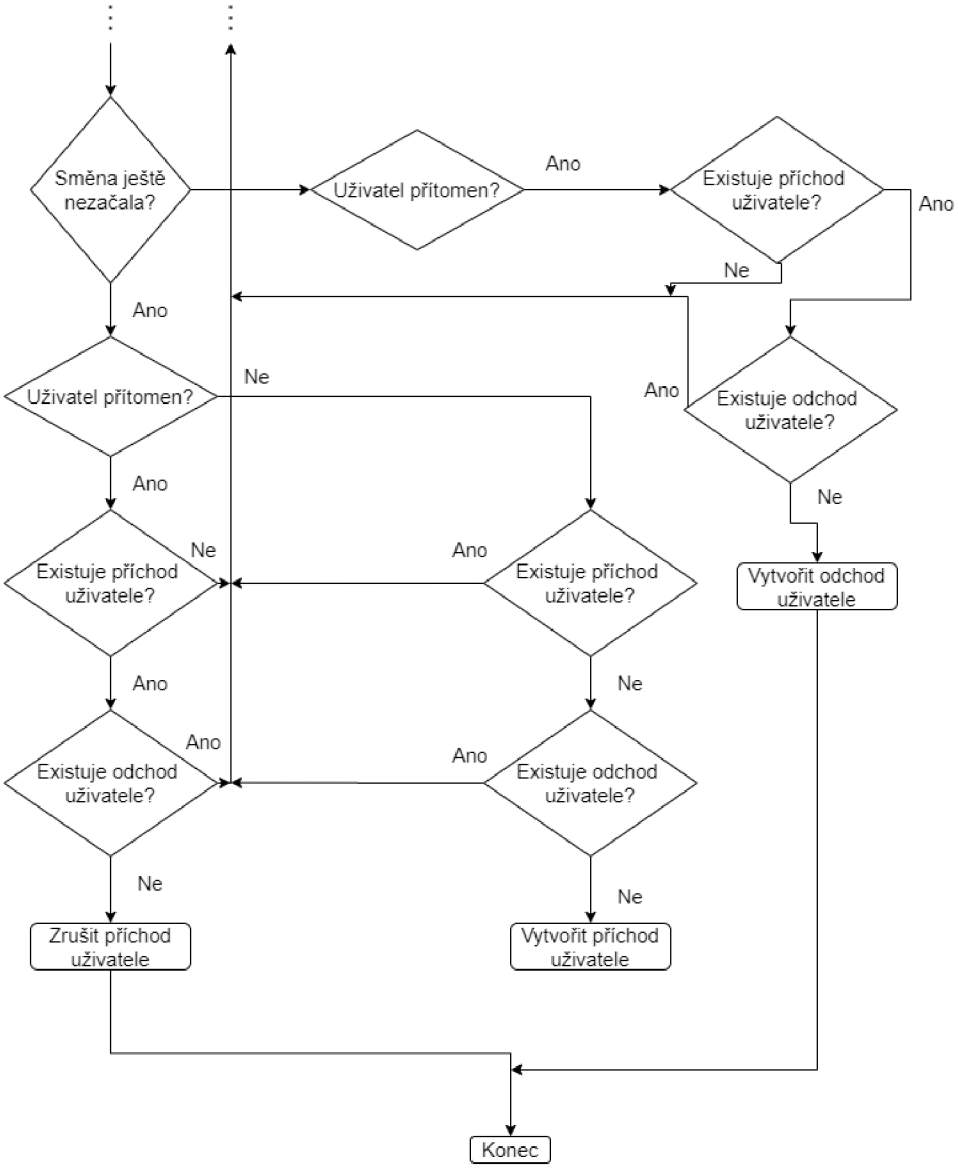



Obrázek 3.16: Kalendář s výpisem směn

Při vytváření směny je nutné vyplnit začátek a konec, popř. doplnit poznámku pro zaměstnance a přiřadit je ke směně. Směny se mohou překrývat. Pokud má uživatel přiřazenou směnu, tak se při příchodu a odchodu zaznamenává čas vůči směně a tím je možné kontrolovat, zda někdo nepřišel pozdě, či neodešel příliš brzy. Vývojový diagram, na obrázcích 3.17 a 3.18, zobrazuje, jak je zpracován přístup uživatele na stanici, která je v režimu docházky, v souvislosti se směnami.



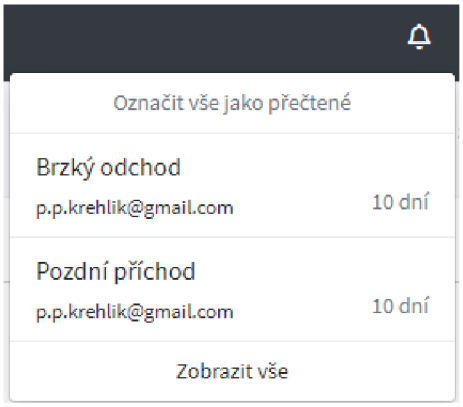

Obrázek 3.17: Vývojový diagram směn, část 1




Obrázek 3.18: Vývojový diagram směn, část 2

### 3.4.15 Notifikace

Systém notifikací je dostupný jen pro manažery a správce. Jak již bylo zmíněno, systém kontroluje příchody a odchody uživatelů na směny. V nastavení je možné nastavit maximální odchylku od začátku a konce směny. Např.: uživatel má začátek směny v 8 hodin a odchylka je nastavena na 30 minut. Pokud uživatel přijde v 8:31, vytvoří se upozornění o pozdním příchodu a je zobrazeno všem manažerům a správcům.
Notifikace 3.19 se zobrazují bez rozdílu stejně pro všechny uživatele, kteří mají již zmíněná oprávnění. Notifikace mají dva stavy a to: přečteno a nepřečteno. Tento stav je globální, takže pokud jeden uživatel označí notifikaci jako přečtenou, zobrazuje se jako přečtená resp. vyřízená pro všechny uživatele.




Obrázek 3.19: Notifikace

### 3.4.16 Zabezpečení

Zabezpečení spojení mezi klientem a webovým serverem, je realizováno pomocí HTTPS protokolu. Přístup pomocí nezabezpečeného protokolu HTTP není dostupný.

#### Oprávnění

Informační systém obsahuje několik rolí uživatelů s různými oprávněními. Při provádění operací uživatelem je nutné kontrolovat, jestli má dostatečné oprávnění.
První krok je zajištění, aby uživateli bylo zobrazeno jen to, k čemu má oprávnění. O oprávnění se stará `MainPresenter`. Tento presenter při startu uloží do šablony informace o aktuálně přihlášeném uživateli. Při vykreslování se pak k těmto informacím přistupuje a zobrazují jen ty položky, ke kterým má uživatel oprávnění.
Mohlo by se ale stát, že se útočník pokusí ručně zadat URL a provést dotaz, k němuž nemá oprávnění. Proto `MainPresenter` obsahuje funkci `isAllowed(...)`. Jakékoliv další presentery rozšiřující tento presenter, mohou poté kdykoliv pomocí této funkce zkontrolovat, zda má právě přihlášený uživatel dané oprávnění. Takže každý presenter může při startu, nebo jakékoliv jiné operaci, zkontrolovat, že má aktuálně přihlášený uživatel dostatečné oprávnění.

### 3.4.17 Grafické rozhraní

Jako grafické rozhraní slouží Bootstrap šablona AdminLTE [3]. Šablona je určena pro vývoj informačních systémů a poskytuje jednoduchý a moderní vzhled.
Celý informační systém je responzivní, tedy přizpůsobuje se velikosti okna, a tím je použitelný na skoro všech zařízeních.

#### Latte

Latte umožňuje jednoduše zobrazovat stránky bez nutnosti duplicitního kódu. Základem je hlavní šablona. Hlavní šablona obsahuje záhlaví a zápatí. Jako obsah se vloží další šablona na základě právě zobrazené stránky.
Předávání dat mezi Nette a Latte funguje jednoduše vložením do šablony např.: pomocí PHP kódu v presenteru: `$this->template->test=1;`. V Latte 3.20 je možné psát běžné PHP konstrukce jako např.: if, while, switch a další. Tím je možné přenést jakoukoliv logiku přímo do HTML kódu.

```
<h3 class="profile-username text-center">{$userName}</h3>

<p class="text-muted text-center">
    {switch $permission}
        {case 1}{_messages.main.roles.registered}
        {case 2}{_messages.main.roles.manager}
        {case 3}{_messages.main.roles.admin}
    {/switch}
</p>
```

Obrázek 3.20: Ukázka integrace Latte v HTML kódu

### 3.4.18 Vývoj

`Nette` obsahuje nástroj `Tracy`. Ten poskytuje komplexní nástroje pro vývoj a ladění PHP aplikace. `Tracy` v případě chyby vypisuje informace o souboru, ve kterém chyba vznikla, o funkci, hodnoty proměnných a další informace. Také měří délku zpracování požadavku, čímž lze zlepšit rychlost aplikace.

### 3.4.19 Známé chyby

Doplněk `Nittro` zajišťuje načítání stránek pomocí AJAX. Ve velice ojedinělých případech nejsou odkazy funkční a je nutné celou stránku obnovit. Jedná se o interní chybu doplňku `Nittro` v klientské části, kterou nedokážu vyřešit.

## 3.5 Aplikační rozhraní a integrace do jiného systému

Aplikační rozhraní je realizováno pomocí frameworku `Apitte`. Kdykoliv je zadán požadavek v podobě: `https://.../api/...`, tak není spuštěna aplikace `Nette`, ale `Apitte`.
Ačkoliv se jedná o samostatný framework, je plně kompatibilní s `Nette`.

### 3.5.1 Architektura

`Apitte` využivá tzv. `Controllery`, které slouží jako koncové body. Ty reprezentují unikátní URL jako např. `/api/users` a operace (HTTP metody). `BaseController` je základní controller a specifikuje výchozí URL pro API.
API poskytuje kompletní kontrolu nad daty v databázi pro integraci do jiného systému. Veškeré odpovědi jsou typu JSON.

```
/**
 * @GroupPath("/api")
 */
class BaseController implements IController
{

}
```

Obrázek 3.21: BaseController

Rozšiřováním třídy, na obrázku 3.21, se buduje posloupnost URL adresy. Mezi koncovými body a `BaseController` je `MainController`. Ten pomocí DI zprostředkovává závislosti a poskytuje pomocné funkce pro zpracování požadavků těm koncovým bodům, které to vyžadují.

### 3.5.2 Koncové body

Koncové body obsluhují samotné požadavky. Metody 3.22 jsou definovány jako funkce, kdy v rámci komentáře funkce, je definováno na jaký požadavek tato funkce reaguje. U metody je nutné specifikovat typ HTTP požadavku např.: GET, POST a další, cestu, parametry a jejich typy a umístění (přímo v URL nebo jako parametr uvnitř požadavku) a návratové kódy.

```
/**
 * Delete station.
 * Admin user token required.
 * @Path("/{email}")
 * @Method("DELETE")
 * @RequestParameters({
 *          @RequestParameter(name="userToken", type="string", description="User token", in="query",
 *      @RequestParameter(name="email", type="string", description="Email of user to delete"),
 * })
 * @Responses({
 *     @Response(code="200", description="Success"),
 *     @Response(code="400", description="Bad request"),
 *     @Response(code="403", description="Forbidden")
 * })
 * @param ApiRequest $request
 * @param ApiResponse $response
 * @return ApiResponse
 */
public function delete(ApiRequest $request, ApiResponse $response): ApiResponse
{
    ...
}
```

Obrázek 3.22: Ukázka metody koncového bodu

### 3.5.3 REST

API dodržuje skoro ve všech případech principy REST. To znamená, že pro získání dat se používá `GET` požadavek, pro vytvoření záznamu `POST`, atd. Jedinou výjimkou je `LegacyController`, který se používá při komunikaci přístupových stanic a serveru. Je to kvůli tomu, že požadavky mezi Arduinem a ESP jsou předávány jako URL. ESP by poté muselo zjišťovat o jaký požadavek se jedná a poté provést, jenže tím by vznikla další nežádoucí logika. Arduino proto využívá jen `GET` požadavky.

### 3.5.4 Zabezpečení

API poskytuje funkce pro práci se všemi daty v systému. Pro ověření se používá token, který mají uživatelé a stanice. REST API využívá pro ověření token uživatele. Je ale nutné, aby byl uživatel správce, jelikož API nedodržuje klasická oprávnění uživatelů, jak bylo zmiňováno v dřívějších kapitolách, a poskytuje kompletní kontrolu nad daty, která by mohla způsobit bezpečnostní riziko. `LegacyController`, který se používá pro komunikaci se stanicemi, využívá pro ověření token stanice.

### 3.5.5 OpenAPI

Pro dokumentaci se využívá standard OpenAPI verze 3. OpenAPI schéma lze získat požadavkem `/api/schema/`. Schéma obsahuje kompletní dokumentaci požadavků, parametrů, a všeho, co je potřebné pro práci s API.

### 3.5.6 Integrace do jiného systému

API umožňuje přístup ke všem datům k databázi, čímž je možné jednoduše napojit jiný systém a získat pomocí něj jakákoliv data.

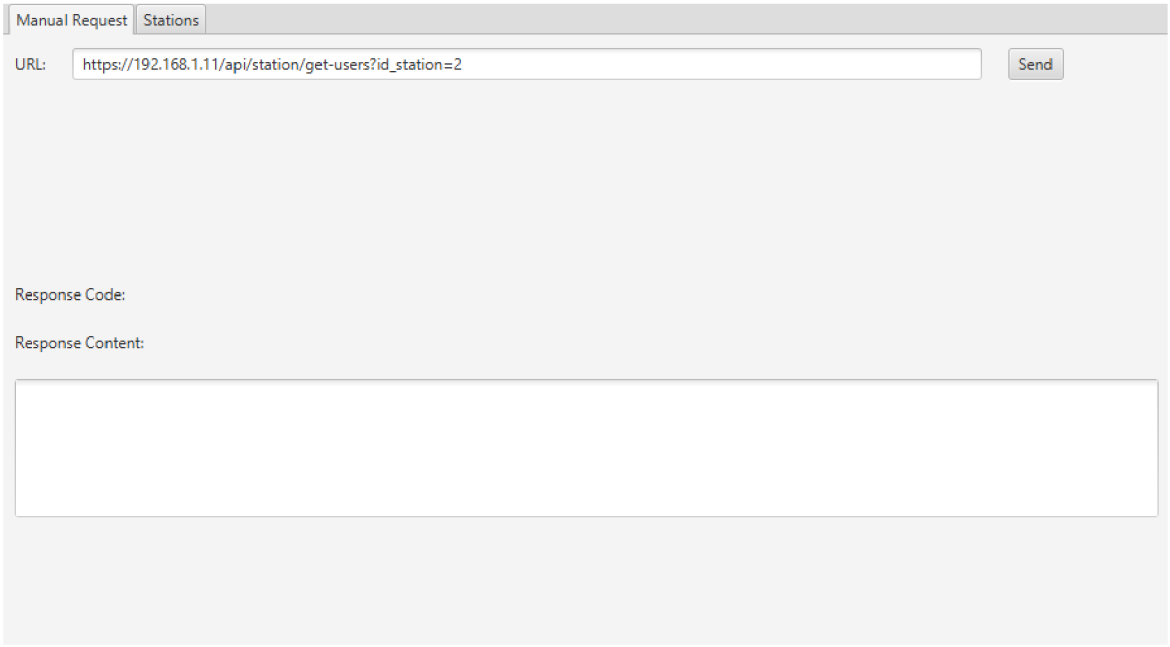
# Kapitola 4

# Testování

Testování systému takovéhoto rozsahu je značně komplikované a nelze rozumně automatizovat. Proto bylo zvoleno několik jiných metod pro otestování systému.

## 4.1 Simulace přístupových stanic

Jelikož vytvořit přístupovou stanici je náročné, tak byl za účelem demonstrace řešení vyroben jen jeden demonstrační kus. Pro simulaci více stanic v systému byla vytvořena aplikace v programovacím jazyce Java verze 8.
Jedná se o jednoduchou aplikaci, která má za úkol simulovat reálné stanice. Aplikace disponuje grafickým uživatelským rozhraním vytvořeným pomocí JavaFX.



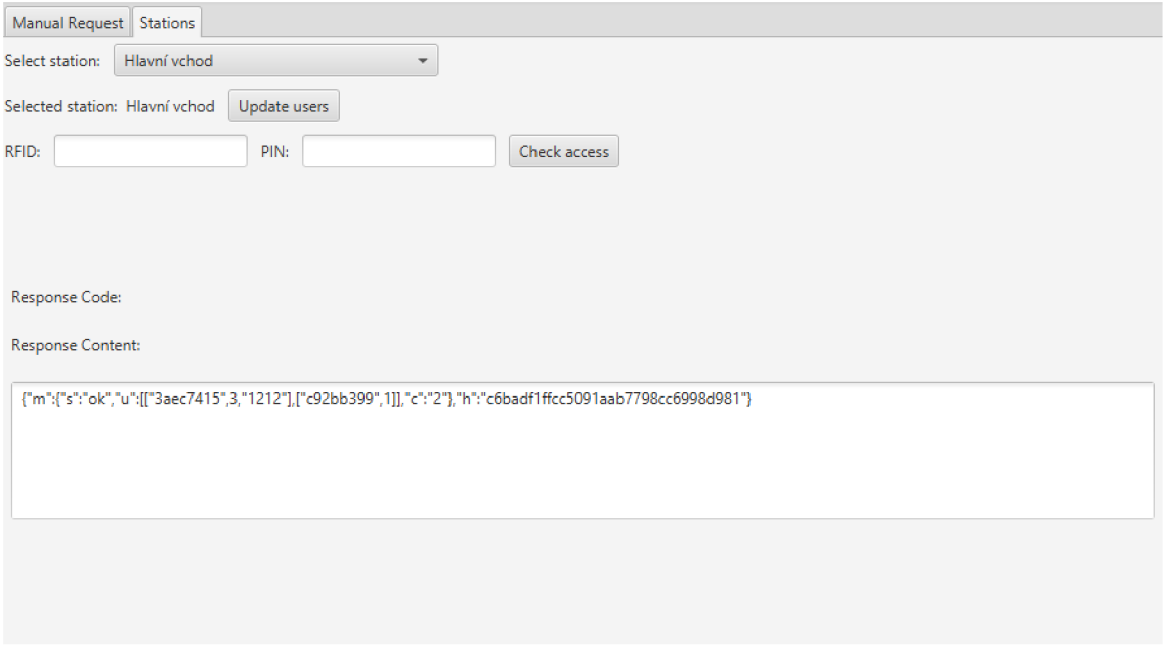

Obrázek 4.1: Ukázka úvodní obrazovky aplikace

### 4.1.1 Manuální požadavky

Úvodní obrazovka, na obrázku 4.1, aplikace umožňuje ručně vytvářet a posílat požadavky. Odpověď serveru je zobrazena v neformátované podobě.

### 4.1.2 Simulace stanic

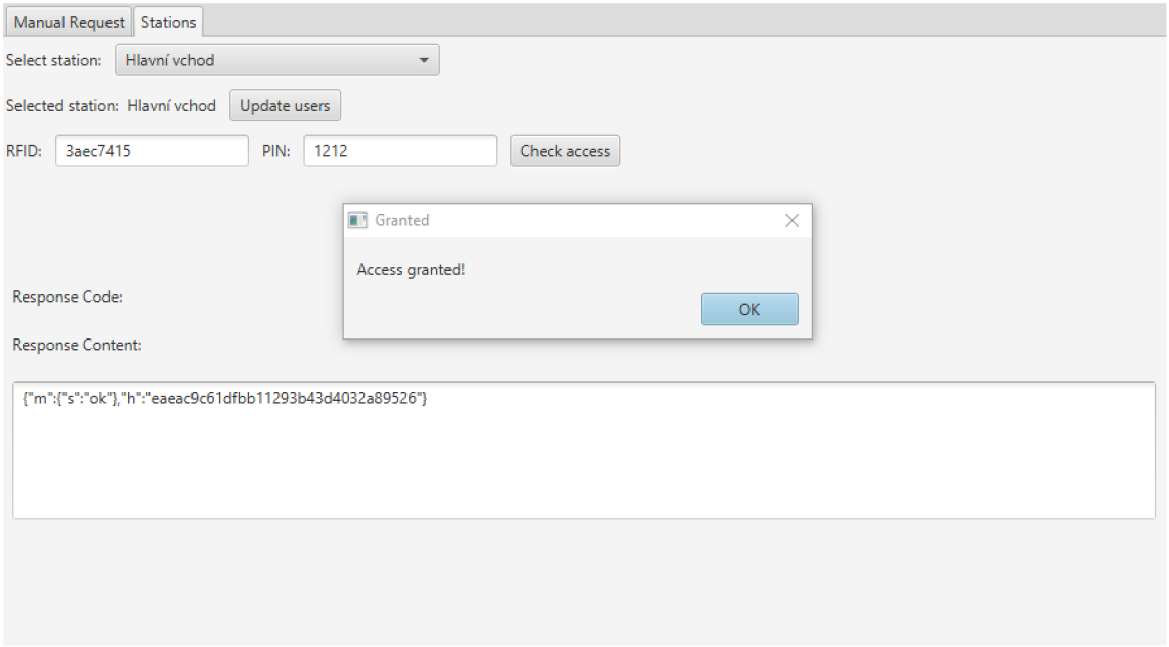
Při zapnutí aplikace se pomocí API ze serveru získá seznam všech stanic v systému a seznam jejich uživatelů. Ty jsou pak na záložce `Stations`, obrázek 4.2, zobrazeny v seznamu. Stačí vybrat konkrétní stanici, v případě potřeby znovu aktualizovat seznam uživatelů, zadat RFID popř. PIN a ověřit přístup. Aplikace provede stejné operace jako reálná stanice. Tímto simulátorem bylo testováno API.




Obrázek 4.2: Ukázka simulace stanice v aplikaci (aktualizace seznamu uživatelů)

### 4.1.3 Příklad testování

Na obrázku 4.3 je příklad testování přístupu a odpovědi serveru.
Jak je patrné na obrázku, systém se chová dle očekávání. Je zobrazeno okno s povolením přístupu a odpověď serveru signalizuje stav `"OK"`.




Obrázek 4.3: Přístup uživatele s PIN kódem

### 4.1.4 Výsledky

Testování probíhalo po skončení vývoje a byly testovány široké možnosti vstupů a poté očekávané výstupy. Pro testování jsem použil aplikaci `Postman`. Jedná se o aplikaci pro vývoj a testování API.
Testování ukázalo, že je API plně funkční.

## 4.2 Testování informačního systému

### 4.2.1 Návrh

Pro otestování použitelnosti informačního systému jsem zvolil metodu, kdy uživatelům zadám sadu úkolů a během jejich práce je budu přímo pozorovat. Pro testování jsem zvolil 3 uživatele.

- Žena, 45 let
- Dítě, 14 let
- Muž, 50 let

Testování předpokládá, že budou mít všichni testovaní uživatelé oprávnění správce. Uživatelům bude poskytnuta úvodní přihlašovací stránka informačního systému a poté již nebudou nijak vedeni k dalším krokům. Testování nepokrývá všechny možnosti informačního systému, ale jen ty nejdůležitější. Cílem testování je zjistit, jak se bude uživatelům s informačním systémem pracovat a případně, na základě zpětné vazby, provést patřičné úpravy.

- Zaregistrujte se a přihlaste
- Zobrazte svůj profil a změňte svoje jméno a PIN
- Změňte jazyk na angličtinu a zpět
- Zobrazte své směny a stanice
- Vytvořte novou směnu a přiřaďte alespoň 2 uživatele
- Odeberte nějakému uživateli RFID, zaregistrujte nové pomocí stanice a přiřaďte ho uživateli
- Smažte stanici číslo 1 všechna oprávnění a vytvořte nová
- Zobrazte Výpis přístupů a filtrujte jen povolené příchody vzestupně podle data a času
- Vygenerujte nějakému uživateli, kromě sebe, nový API klíč
- Odhlaste se, obnovte heslo pomocí emailu a znovu přihlaste novým heslem

### 4.2.2 Výsledky

- **Žena, 45 let**
  Základní operace jako přihlášení, správa profilu a zobrazení vlastních směn, atd. je bez problémů.
  Komplexnější operace jsou problematičtější. Některé operace uživatel bez pomoci vůbec neprovedl a nechápal princip toho, co má dělat. Na základě zpětné vazby bylo vylepšeno několik překladů a vzhled tlačítek. Z hlediska informačního systému se uživatel orientoval dobře, ale nechápal příliš co s čím souvisí z hlediska celého systému, např.: vazba uživatele a RFID.

- **Dítě, 14 let**
  Základní operace bez problému, komplexní operace opět problematické. Uživatel byl docela "zběsilý", čímž přehlédl několik prvků a klikal na jiné bez rozmyslu. Uživatel pochopil systém jako celek. Při práci s informačním systémem byl ale zmatený a dělal mu veliký problém rozdělení menu na část pro manažera a správce. Vytýkal také plánování směn, které by mělo být více provázané s grafickým kalendářem.

- **Muž, 50 let**
  Uživateli trvalo delší dobu se zorientovat a zjistit, kde jaké položky najít. Ve výsledku byly stejné problémy jako u předchozích uživatelů a ničím zvláštním se neodlišoval.

Z testování vyplynulo, že grafické rozhraní není příliš přehledné a na základě zpětné vazby bylo provedeno několik úprav pro zlepšení přehlednosti. Uživatele, ačkoliv to nebylo v rámci testování v plánu, jsem musel částečně navést, aby byli schopni najít všechny funkce.

## 4.3 Testování systému jako celek

Systém jsem po dokončení vývoje testoval ručně, jelikož takto komplexní systém nelze testovat automatizovaně a proto jsem se snažil o co nejdůkladnější testování.
Testování hardwarové části nebylo složité. Zkoušel jsem různé vstupy, přikládat RFID čipy, testovat, zda displej správně reaguje atd. Během testování jsem nenarazil na žádný problém a systém fungoval.
Další částí je informační systém. Ten byl testován opět ručně zkoušením co nejvíce variant vstupů, které mohou nastat. Během tohoto testování bylo odhaleno mnoho chyb. Tyto chyby byly opraveny.

# Kapitola 5

# Závěr

Cílem práce bylo vytvořit fyzicky přístupovou stanici a propojit ji se serverem. Dále byl vytvořen informační systém pro kompletní správu a API pro propojení stanic a integrování do jiných systémů. Během vytváření přístupové stanice jsem zjistil, že do budoucna by bylo dobré nahradit kombinaci desek Arduino a ESP8266 jen jednou deskou ESP32. Výhody této desky jsou hlavně ve větším výkonu, paměti a také zjednodušení komunikace, jelikož by bylo vše řízeno z jedné desky. Dalším nedostatkem je HMI panel. Ten během práce přestal pracovat, resp.: dotyková vrstva přestala reagovat na dotyky. Není to první případ, kdy tento problém s HMI panelem od společnosti Nextion nastal a do budoucna bych ho vyměnil za jiný model. Celkově ale musím hodnotit hardware za spolehlivý vůči své ceně. Zajímavým rozšířením by mohlo být použití pohybového senzoru, kdy displej by byl zapnutý jen při detekci pohybu.
Informační systém je také možné rozšířit a to např.: komplexnější správou směn, podrobnějších výpisů a mnoho dalších rozšíření. Pokud bych tuto práci vytvářel znovu, tak bych rozhodně navrhl uživatelské rozhraní informačního systému jinak, jelikož během testování jsem zjistil, že pro uživatele není přehledný a příliš přívětivý.

V práci je nejprve představeno, proč toto řešení vzniká a poté pomocí jakých prostředků bude realizováno. V dalších kapitolách je popisováno, jak byly dané technologie použity a řešeny různé problémy, které se během vývoje vyskytly. Na závěr je popsáno, jak bylo provedeno testování.

Důležité ale je, že se všechny technologie podařilo úspěšně propojit, jelikož práce využívá velké množství různých technologií, a výsledkem je funkční produkt. Získal jsem díky němu nové znalosti z nejrůznějších oblastí.

# Literatura


[1] Arduino. *Arduino* [online]. 2020 [cit. 2020-07-09]. Dostupné z: https://www.arduino.cc/.

[2] Bootstrap. *Bootstrap documenation* [online]. 2020 [cit. 2020-07-12]. Dostupné z: https://getbootstrap.com/docs/4.5/getting-started/introduction/.

[3] Colorlib. *AdminLTE* [online]. 2020 [cit. 2020-07-09]. Dostupné z: https://adminlte.io/.

[4] ESPRESSIF. *ESP8266EX Datasheet* [online]. 2020 [cit. 2020-07-09]. Dostupné z: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf.

[5] Foundation, N. *Nette Documentation* [online]. 2020 [cit. 2020-07-09]. Dostupné z: https://doc.nette.org/en/3.0/.

[6] Foundation, R. P. *Raspberry Pi* [online]. 2020 [cit. 2020-07-09]. Dostupné z: https://www.raspberrypi.org/.

[7] Nextras. *Nextras Components* [online]. 2020 [cit. 2020-07-09]. Dostupné z: https://nextras.org/.

[8] RobotDyn. *MEGA+WiFi R3 ATmega2560+ESP8266, flash 32MB, USB-TTL CH340G, Micro-USB* [online]. 2020 [cit. 2020-07-09]. Dostupné z: https://robotdyn.com/mega-wifi-r3-atmega2560-esp8266-flash-32mb-usb-ttl-ch340g-micro-usb.html.

[9] STUDIO, I. *Nextion* [online]. 2020 [cit. 2020-07-09]. Dostupné z: https://nextion.tech/.

# Příloha A

# Manuál

## A.1 Přístupová stanice

### A.1.1 Přístup

Použití přístupové stanice z hlediska uživatele je velice jednoduché. Pro přístup stačí přiložit RFID čip do prostoru pod displejem a pokud je to vyžadováno, tak zadat PIN.

### A.1.2 Režim správce

#### Aktualizovat data

Aktualizuje seznam uživatelů ze serveru. Aktualizace probíhá na pozadí.

#### Restart HMI

Restartuje displej, čímž ukončí režim správce.

#### Výběr relé

Zobrazí další obrazovku. Kliknutí na tlačítko přepíná režim příslušného relé.

#### Otevřít dveře

Spustí relé, která jsou v režimu `DVERE`.

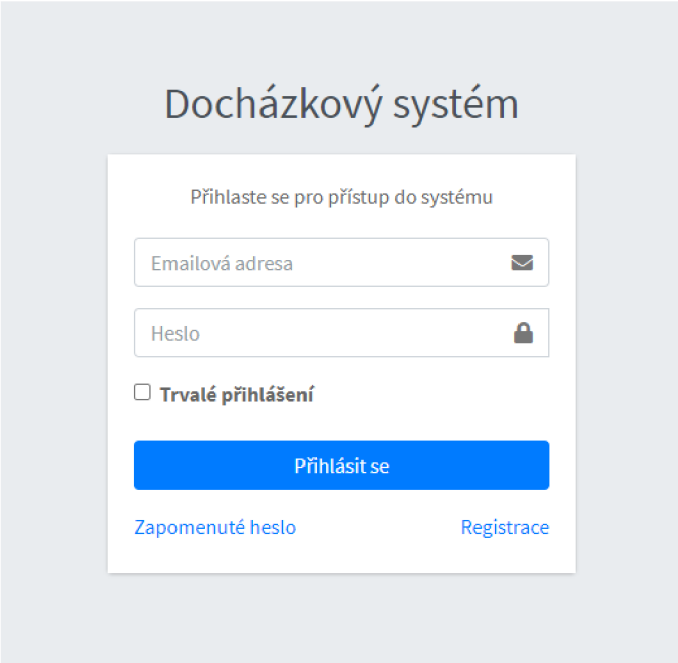
#### Registrace RFID

Po výzvě přiložte čip a klikněte na tlačítko `Potvrdit`. RFID se, pokud již není v systému ,uloží do databáze.

## A.2 Informační systém

### A.2.1 Přihlášeni, registrace, obnovení hesla

Pro přihlášení, na obrázku A.1, je nutné zadat email a heslo. Je možné trvalé přihlášení po dobu 14 dní, v opačném případě vyprší přihlášení po 30 minutách.
Při registraci je nutné vyplnit potřebné údaje a poté vyčkat na potvrzení účtu správcem.
Pro obnovení hesla je nutné zadat email. Na ten je poté zasláno nové dočasné heslo.




Obrázek A.1: Stránka s přihlášením

### A.2.2 Profil

Stránka profilu A.2 umožňuje změnu hesla, jména, osobního PIN kódu a vygenerování nového API klíče.

### A.2.3 Nástěnka

Nástěnka A.3 zobrazuje odpracované hodiny tento a minulý týden spolu s nadcházejícími směnami. Pro manažery a správce je zobrazen počet aktuálně přítomných uživatelů.

### A.2.4 Jazyk

Jazyk probíhá pomocí "vlajky" v pravém horním rohu. Na výběr je angličtina a čeština.

### A.2.5 Základní možnosti

Každý uživatel si může zobrazit své směny a oprávnění na stanicích.

Obrázek A.2: Profil

Obrázek A.3: Úvodní okno informačního systému (nástěnka)

### A.2.6 Modul manažera

#### Základní možnosti

Jako manažer máte možnost zobrazit v tabulkách přítomné uživatele, stanice a oprávnění uživatelů na stanicích.

#### Správa směn

Směny se zobrazují v kalendáři a tabulce seřazené dle data a času začátku. V tabulce s výpisem směn je možné provádět přímo editaci směny nebo ji smazat. Lze také pomocí zaškrtávacích políček mazat směny hromadně.
Po kliknutí na směnu v tabulce nebo kalendáři se zobrazí tabulka s výpisem přiřazených uživatelů. Ty lze odebrat.
Při vytváření nové směny je nutné zadat začátek a konec, kdy konec musí být později než začátek. Poznámka směny je volitelná. Uživatelé se vybírají ze seznamu. Pro výběr více uživatelů je možné použít klávesu `CTRL` pro jednotlivý výběr nebo `SHIFT` pro hromadný výběr.

#### Správa uživatelů

Zobrazí se tabulka, na obrázku A.4, se všemi uživateli. Jako manažer lze pouze pracovat s účty uživatelů nižší úrovně. Výjimkou je plánování směn a přiřazování uživatelů na směny, které je možné provádět jako manažer bez omezení.
Dále lze příslušnými tlačítky přiřadit nebo odebrat RFID uživateli, nastavovat oprávnění na stanicích a plánovat uživateli směny.

### A.2.7 Modul správce

Jako správce máte neomezené oprávnění nad všemi uživateli a prostředky systému.

#### Základní možnosti

Správce má stejná oprávnění jako manažer a navíc možnost zobrazit kompletní výpis přístupů ze všech stanic.

#### Správa stanic

Umožňuje kompletní správu stanic. Lze vytvářet nové, upravovat a mazat je.

#### Správa uživatelů

Správa uživatelů poskytuje jiné možnosti než stejná stránka v Modulu manažera.
Zde je možné upravit veškeré údaje jakéhokoliv uživatele, včetně ruční změny RFID. Lze také vygenerovat komukoliv nový API klíč a smazat účet.

#### Správa nových RFID

Správa RFID, která nejsou přiřazena žádnému uživateli a čekají na přiřazení. Je možné tyto záznamy mazat.

### Nastavení

Umožňuje nastavení odchylky od začátku a konce směny. Nastavení je pouze v angličtině.

## A.3 Proces registrace

Po registraci je nutné vyčkat na schválení účtu správcem. Poté je teprve možné se přihlásit do informačního systému.

## A.4 Proces přiřazení RFID uživateli

Přiřazení RFID může provádět manažer a správce, nicméně přidat RFID do systému na terminálu může pouze správce.
Přidání nového RFID do systému pomocí terminálu:

- Přiložte čip správce k terminálu a zadejte osobní PIN kód.
- Klikněte na tlačítko "Registrace RFID".
- Přilože čip ke čtečce a vyčkejte na zobrazení identifikátoru.
- Klikněte na tlačítko "Potvrdit". Popřípadě můžete operaci zrušit tlačítkem "Zrušit".
- Čip je nyní přidán do systému a může být přiřazen k uživateli. Pokud se čip do systému nepřidal, tak již v systému je registrovaný.

Přiřazení RFID uživateli:

- Přejděte na stránku Modul manažera -> Správa uživatelů.
- Klikněte na tlačítko "Přiřadit RFID"u daného uživatele.
- Vyberte RFID ze seznamu kliknutím na tlačítko "Přiřadit"u daného řádku.
- Čip je nyní přiřazen k uživateli.

Poznámka: Pro přístup uživatele do systému nestačí, aby měl přiřazené RFID, jeho účet musí být také schválený.

| | ID | Křestní jméno | Příjmení | Emailová adresa | Osobní RFID | Schválený účet? | Datum/Čas registrace | Poslední přihlášení | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Vše | dd.mm.rrrr - dd.mm.rrrr | dd.mm.rrrr - dd.mm.rrrr | Filtrovat data Zrušit |
| ☐ | 1 | Honza | Palát | p.p.krehlik@gmail.com | 3aec7415 | ANO | 30.11.-0001 00:00:00 | 12.07.2020 15:05:54 | Upravit RFID Opravnění Směny |
| ☐ | 4 | Lenka | Křehlíková | lkrehlikova@centrum.cz | e953345b | ANO | 28.08.2017 00:00:00 | 30.06.2020 16:14:08 | Upravit RFID Opravnění Směny |
| ☐ | 5 | Vladimír | Drápela | vdrap@seznam.cz | | ANO | 28.08.2017 00:00:00 | | Upravit RFID Opravnění Směny |
| ☐ | 6 | Josef a Helena | Křehlíkovi | email@email.com | | ANO | 28.08.2017 00:00:00 | | Upravit RFID Opravnění Směny |
| ☐ | 7 | Petr | Křehlík | pekrehlik@seznam.cz | c92bb399 | ANO | 21.09.2017 18:52:41 | 01.07.2020 10:51:16 | Upravit RFID Opravnění Směny |

Obrázek A.4: Správa uživatelů v režimu manažera

# Příloha B

# Seznam použitých knihoven a doplňků

## B.1 Přístupová stanice

Arduino core for ESP8266 WiFi chip
ArduinoJson
ArduinoMD5
DHT sensor library
EEPROM Library
MFRC522
SPI Library

## B.2 Informační systém

Apitte
Bootstrap
Chart.js
DateInput
flag-icon-css
Font Awesome
FullCalendar
iziToast
jQuery
Kdyby/Translation
Moment.js
Nextras DataGrid
Nextras ORM
Nittro