Masarykova univerzita
Fakulta informatiky



Webová aplikace pro zobrazení dat z meteorologické stanice

Bakalářská práce

Tomáš Jochec

Brno, jaro 2017

Masarykova univerzita
Fakulta informatiky



Webová aplikace pro zobrazení dat z meteorologické stanice

Bakalářská práce

Tomáš Jochec

Brno, jaro 2017

Na tomto místě se v tištěné práci nachází oficiální podepsané zadání práce a prohlášení autora školního díla.

Prohlášení

Prohlašuji, že tato bakalářská práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Tomáš Jochec

Vedoucí práce: doc. PhDr. Josef Prokeš, Ph.D.

Poděkování

Rád bych zde poděkoval doc. PhDr. Josefu Prokešovi, Ph.D. za cenné rady a ochotu při vedení této práce.

Shrnutí

Práce se zabývá návrhem architektury, komunikačního rozhraní, uživatelského prostředí a samotnou implementací webové aplikace pro zobrazení dat z meteorologických stanic vlastní výroby. Součástí práce je sestavení jednoduché meteorologické stanice na platformě Arduino.

Klíčová slova

webová aplikace, meteorologická stanice, Arduino, Node.js, REST API

Obsah

1 Úvod **1**

2 Analýza **2**
2.1 *Postup analýzy* 2
2.2 *OpenWeatherMap* 2
2.3 *Weather Underground* 3
2.4 *Shrnutí* 3

3 Návrh uživatelského rozhraní **5**

4 Implementace webové aplikace **8**
4.1 *Funkční požadavky* 8
4.2 *Programovací jazyk* 9
4.3 *Klientská část* 9
4.4 *Serverové aplikační rozhraní* 11
4.5 *Návrh databáze* 11
4.5.1 PostgreSQL 11
4.5.2 Relační databázový model 12
4.6 *Autentizace klientské aplikace* 13
4.7 *Autentizace IoT zařízení* 14
4.8 *Nasazení aplikace* 17

5 Meteorologická stanice na platformě Arduino **18**
5.1 *Použité technologie* 18
5.1.1 Platforma Arduino 18
5.1.2 Arduino IDE 18
5.1.3 Implementace 19
5.2 *Hardwarová implementace* 21
5.2.1 Arduino MEGA 2560 21
5.2.2 Wi-Fi modul 23
5.2.3 Senzory 25
5.2.4 Schéma zapojení 26
5.2.5 Zprovoznění zařízení 30
5.3 *Testování* 30

6 Závěr **31**

Bibliografie **32**

A Seznam elektronických příloh **34**

B Dokumentace REST API **35**
B.1 *Měření (Measurements)* 35
B.1.1 Načtení dat od konkrétního zařízení 35
B.2 *Data* 36
B.3 *Zařízení (Devices)* 37
B.3.1 Načtení všech zařízení 37
B.3.2 Načtení jednoho zařízení 38
B.3.3 Vytvoření nového zařízení 39
B.3.4 Úprava zařízení 39
B.3.5 Smazání zařízení 41
B.3.6 Reset soukromého klíče 42
B.4 *Uživatelské účty (User Accounts)* 43
B.4.1 Registrace nového uživatele 43
B.4.2 Přihlášení uživatele 44

Seznam tabulek

5.1 Technická specifikace desky Arduino MEGA 2560[16] *22*
5.2 Seznam použitých součástek *26*

Seznam obrázků

3.1 Mockup uživatelského rozhraní aplikace *6*
3.2 Reálné uživatelské rozhraní aplikace *7*
4.1 Diagram případů užití *9*
4.2 ER diagram databáze *13*
4.3 Struktura JSON Web Tokenu *14*
4.4 Ukázka HTTP požadavku v programu Charles *16*
5.1 Výchozí verze firmwaru *24*
5.2 Schéma zapojení č. 1 *27*
5.3 Schéma zapojení č. 2 *28*
5.4 Prototyp zařízení *29*

1 Úvod

S rozmachem finančně dostupných vestavěných systémů, z nejvýznamnějších uveďme např. Arduino, Raspberry Pi či BeagleBone Black, se amatérským meteorologům dostává do rukou nástroj pro rychlou implementaci jednoúčelových zařízení – v tomto případě meteorologické stanice připojené k síti Internet. Častým řešením pro další zpracování a ukládání naměřených dat je implementace vlastní aplikace na míru danému zařízení. Tento přístup vede ke vzniku velkého množství vzájemně nekompatibilních řešení a ke znovu vytváření již existujícího.

Cílem této práce je navrhnout a implementovat jednoduchou webovou aplikaci pro vizualizaci naměřených dat z meteorologických stanic, která bude po spuštění ihned použitelná, a zároveň může sloužit jako výchozí bod pro tvorbu takovýchto aplikací. Ve své základní podobě bude aplikace nabízet možnost shromažďování a vizualizace několika druhů dat. Součástí bude zdokumentované rozhraní pro komunikaci s klientskými zařízeními a uživatel se znalostí programování bude mít možnost aplikaci dále modifikovat a rozvíjet dle svých potřeb.

Při návrhu aplikace bude brána v potaz omezená výpočetní kapacita vestavěných systémů. Zabezpečení síťové komunikace bude navrženo tak, aby bylo co nejvíce bezpečné, ale zároveň stále v možnostech slabších vestavěných systémů. Ty jsou pro tvorbu meteorologické stanice vhodné vzhledem k jejich malé energetické spotřebě, obzvláště pokud má být stanice napájena bateriemi. Pozornost bude také věnována návrhu a přístupnosti uživatelského rozhraní webové aplikace. Zdrojové kódy webové aplikace a ukázková implementace jednoduché meteorologické stanice na platformě Arduino je popsána v kapitole 5 a dostupná jako elektronická příloha této práce.

2 Analýza

2.1 Postup analýzy

V této kapitole se zaměřím na dvě oblíbené webové služby využívané amatérskými meteorology, jejichž cílem je sběr a vyhodnocování meteorologických dat, a to konkrétně na služby OpenWeatherMap a Weather Underground. Analyzuji přístupnost jejich komunikačního rozhraní pro vývojáře a možnosti, jež poskytují pro práci se shromážděnými daty.

2.2 OpenWeatherMap

OpenWeatherMap je webová služba vyvíjená společností OpenWeatherMap, Inc.. Jejím cílem je poskytovat geodata prostřednictvím jednoduché webové služby a aplikačního programovacího rozhraní (API). Služba sbírá data z tisíců senzorů rozmístěných po celém světě a využívá moderní technologie určené ke zpracování velkého objemu dat.[1]

Běžným uživatelům jsou data dostupná skrze webové rozhraní. Nabízené API pro vývojáře aplikací umožňuje jak přístup k nasbíraným datům, tak možnost odesílat naměřená data z vlastních zařízení. Požadavek na získání dat ze serveru je nutno opatřit přístupovým klíčem, který je dostupný pro registrované uživatele v administraci webové aplikace. Data lze získat ve standardizovaných formátech JavaScript Object Notation (JSON), Extensible Markup Language (XML) nebo HyperText Markup Language (HTML).

Pro odesílání vlastních dat na server je primárně určen komunikační protokol HTTP a autentizace probíhá pomocí jednoduchého schématu Basic access authentication. Tento typ autentizace má své opodstatnění vzhledem k možné omezené výpočetní kapacitě klientského zařízení, ale i přesto jej považuji za velkou bezpečnostní slabinu, protože komunikaci lze jednoduše odposlechnout a zjistit přihlašovací údaje uživatele. Služba ve své základní variantě zdarma neumožňuje přístup ke všem datům a počet požadavků na server je omezen. Ve vyšším placeném tarifu je pro komunikaci možné použití šifrovaného komunikačního protokolu HTTPS, pokud ho dané klientské zařízení podporuje. Pomineme-li možnost potenciálního zneužití nezabezpe-

čené komunikace, může být služba ve své základní podobě pro osobní použití dostačující.

2.3 Weather Underground

Weather Underground využívá data od stále se zvětšující komunity (ke dni 5. 11. 2016 více než 200 000 uživatelů) v proprietárním systému zvaném BestForecast^{TM}, určeném pro předpověď počasí na lokální úrovni.[2] Služba však podporuje pouze některé druhy komerčně dostupných meteorologických stanic. Odesílání dat z vlastních zařízení není v současné době možné.

K aplikačnímu programovacímu rozhraní služby je dostupná podrobná dokumentace a dokáže poskytovat data ve formátu XML nebo JSON. Stejně jako u OpenWeatherMap se pro komunikaci využívá komunikační protokol HTTP a požadavek na získání dat ze serveru je nutno opatřit unikátním přístupovým klíčem. K dispozici je několik plánů pro přístup k datům, lišících se množstvím poskytovaných dat a dostupným počtem požadavků na server. Základní podoba aplikace je zdarma. Webové rozhraní poskytuje uživateli přístup k předpovědi počasí a k zobrazení dat z jeho zařízení.

2.4 Shrnutí

Obě analyzované služby poskytují podobnou funkcionalitu a vzájemně si konkurují. Kvůli absenci podpory vlastnoručně sestavených klientských zařízení u služby Weather Underground se služba OpenWeatherMap pro náš případ užití jeví jako vhodnější. U této služby ovšem přetrvává problém s nezabezpečeným autentizačním protokolem pro přenos dat z klientských zařízení na server. Služba sice v placeném tarifu nabízí možnost komunikace skrze šifrovaný komunikační kanál HTTPS, který by danou bezpečnostní hrozbu vyřešil, ovšem jeho použitím bychom se připravili o podporu méně výkonných klientských zařízení, což je v případě, kdy se snažíme o univerzálnost aplikace, nežádoucí. Řešením je implementace bezpečnějšího autentizačního schématu, využívajícího pro přenos informací výpočetně nenáročný protokol HTTP, při němž se nebudou v kaž-

dém požadavku na server odesílat přihlašovací údaje uživatele. Tento autentizační přístup je podrobně popsán v kapitole 4.6.

3 Návrh uživatelského rozhraní

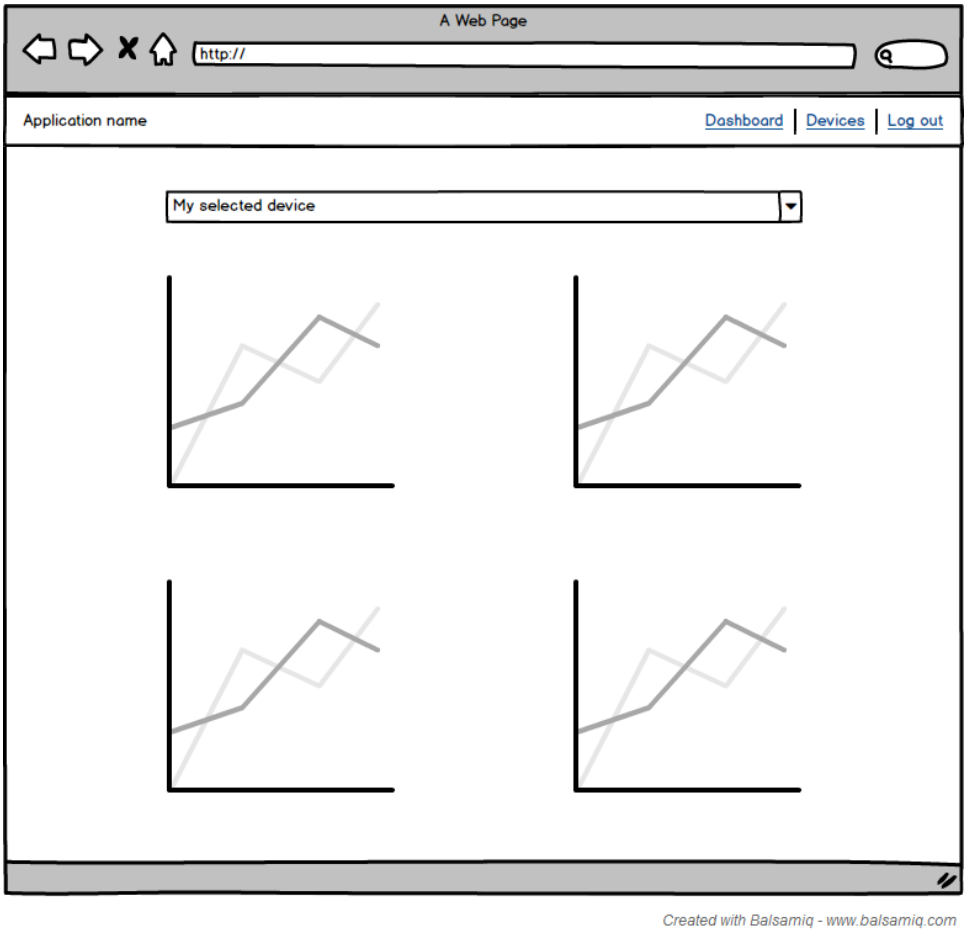
Uživatelské rozhraní jsem se rozhodl navrhnout minimalistické a snadno pochopitelné pro běžného uživatele. Pro tvorbu návrhu jsem využil nástroj Balsamiq[1], sloužící k tvorbě tzv. wireframů. Wireframe je „skica webu“ postrádající grafiku, která se soustředí výhradně na funkce, obsah a rozmístění jednotlivých komponent na webové stránce. Ukázka návrhu je na obrázku 3.1.

Uživatelské rozhraní se skládá pouze z horní lišty, jednoho hlavního sloupce s obsahem a patičky stránky. Horní lišta obsahuje odkazy v závislosti na tom, zda je uživatel autentizován či nikoliv. Pro nepřihlášeného uživatele se zobrazí odkazy pro vytvoření nového uživatelského účtu a pro přihlášení. Přihlášený uživatel zde nalezne odkaz na hlavní stránku, dále odkaz na stránku s přehledem zařízení a konečně odkaz pro odhlášení z aplikace.

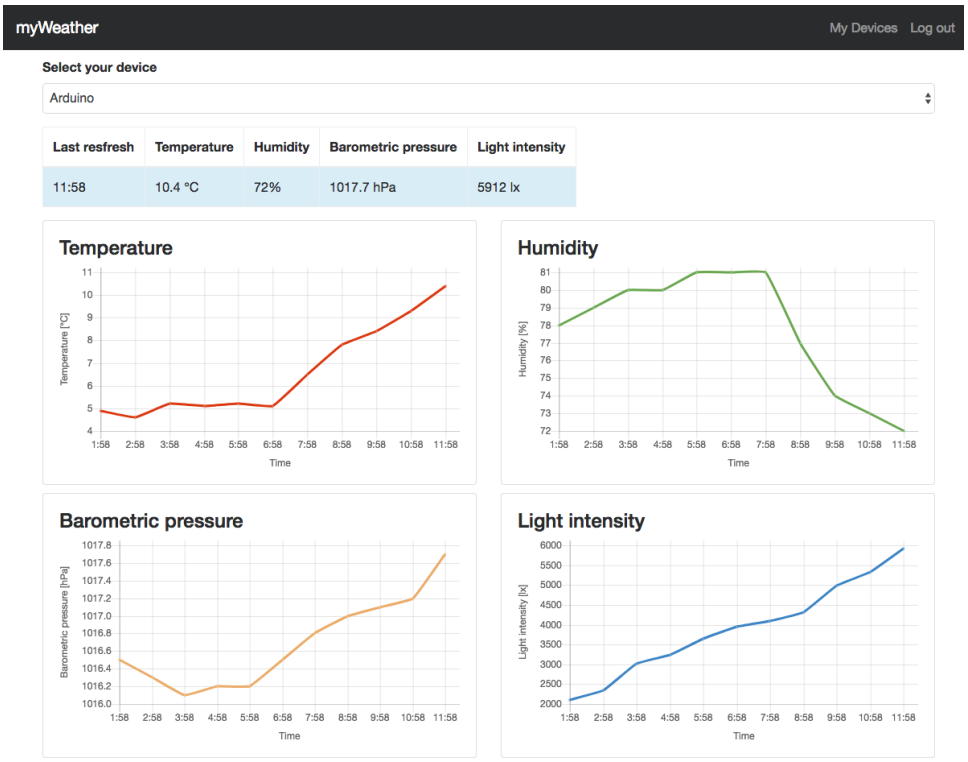
Obsah hlavního panelu je závislý na tom, ve které části aplikace se uživatel zrovna nachází. Úvodní stránka aplikace obsahuje základní informace o aplikaci a přehled všech funkcí, které aplikace nabízí. Přihlášenému uživateli je dostupná hlavní stránka, jež obsahuje vizualizaci naměřených dat. V horní části hlavního panelu si uživatel může zvolit, ze kterého zařízení chce data zobrazit.

Další stránka obsahuje přehled všech zařízení, jež si uživatel do aplikace přidal. Na této stránce lze také upravit informace o stávajících zařízeních (změnit jméno, popis nebo resetovat privátní klíč) nebo vytvořit nové zařízení. V patičce stránky se nachází odkaz na stránku s dokumentací aplikačního rozhraní. Pro tvorbu uživatelského rozhraní byl použit aplikační rámec Bootstrap[2]. Jedná se o světově nejpopulárnější HTML, CSS a JS aplikační rámec pro tvorbu responsivních webových stránek.[3] Uživatelské rozhraní je díky použití tohoto aplikačního rámce plně responsivní a snadno přístupné pro uživatele mobilních i desktopových zařízení. Výsledné rozhraní je zobrazeno na obrázku 3.2.

1. `https://balsamiq.com/`
2. `http://getbootstrap.com/`



Obrázek 3.1: Mockup uživatelského rozhraní aplikace



Obrázek 3.2: Reálné uživatelské rozhraní aplikace

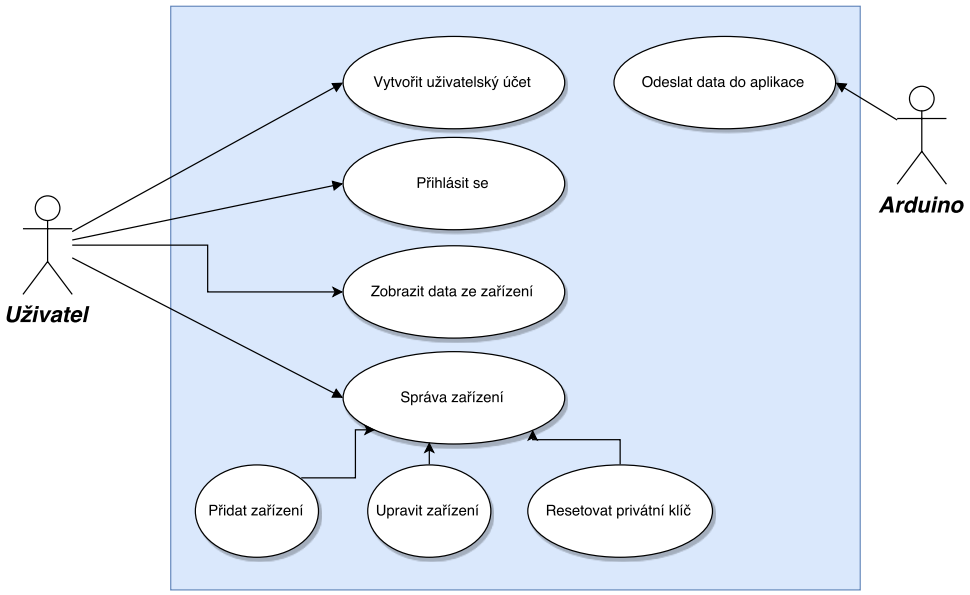
4 Implementace webové aplikace

Jednou z nejdůležitějších fází ještě před samotným začátkem vývoje je analytická fáze, při které je potřeba se podrobně zamyslet nad tím, jaké funkcionality by aplikace měla poskytovat a jak toho nejlépe docílit. Vhodný návrh v této fázi dokáže značně ulehčit celý vývoj aplikace a vyplatí se ho nezanedbat. Cílovými uživateli aplikace pro vizualizaci dat z meteorologických stanic jsou především domácí kutilové, kteří jsou schopni takové zařízení sami sestavit a naprogramovat. Můžeme proto u uživatelů očekávat jisté technické znalosti a počítat s nimi při návrhu celého systému.

4.1 Funkční požadavky

Funkční požadavky definují hlavní funkce, které by měla aplikace uživatelům poskytovat. Při vymýšlení jednotlivých případů použití jsem se zamyslel nad nejdůležitějšími funkcemi aplikace. Vzal jsem také v úvahu výsledky analýzy z kapitoly 2 a zaměřil jsem se na následující funkční požadavky:

- Kdokoliv si může vytvořit uživatelský účet.
- Zaregistrovaný uživatel si bude moci k účtu přiřadit neomezené množství zařízení.
- Každé zařízení bude mít přiřazený veřejný a soukromý klíč pro komunikaci se serverovým aplikačním rozhraním.
- Zaregistrovaný uživatel bude mít možnost upravovat název a popis jeho zařízení a resetovat jejich privátní klíč pro případ, že by byl kompromitován.
- Aplikace bude poskytovat zdokumentované serverové aplikační rozhraní.
- Aplikace bude vykreslovat data z jednotlivých zařízení za posledních 24 hodin do grafů.



Obrázek 4.1: Diagram případů užití

4.2 Programovací jazyk

Implementace webové aplikace je realizována v JavaScriptu. Konkrétně jsem využil nejnovější specifikaci ECMAScript 2015.

ECMAScript je objektově orientovaný programovací jazyk pro provádění výpočtů a manipulaci s výpočetními objekty v hostitelském prostředí.[4] Jedná se o zápis, který ještě není plně podporován, proto je pro překlad do zpětně kompatibilního kódu použita knihovna Babel[1].

4.3 Klientská část

Klientská část aplikace je založena na knihovně React. React je deklarativní, efektivní a flexibilní JavaScriptová knihovna pro tvorbu uživatelských rozhraní.[5] Aplikace postavená na této knihovně se skládá z komponent, které mají deklarovaný svůj stav, na jehož zá-

1. `https://babeljs.io/`

kladě je React vykreslí. V případě změny stavu komponenty se React automaticky postará o aktualizaci uživatelského prostředí. Pro návrh jednotlivých podstránek webové aplikace jsem využil kontejnerový návrhový vzor (container pattern). Při tomto přístupu má každá podstránka svoji tzv. kontejnerovou komponentu, která ji reprezentuje. Ta se stará o komunikaci s aplikačním rozhraním a následné vykreslení ostatních komponent, které se mají na dané podstránce zobrazit. Výhodou tohoto návrhového vzoru je skutečnost, že jednotlivé komponenty jsou malé a jejich zdrojový kód je přehledný. Oddělení logiky načítaní dat a vykreslovaní komponent má za následek také to, že jednotlivé komponenty lze navíc znovu použít a vykreslit i v jiných kontejnerových komponentách.

Architektura aplikace vychází z konceptu jednostránkové aplikace (single-page application). Při prvním požadavku se načte a lokálně uloží celý kód aplikace, která poté se serverem komunikuje prostřednictvím zdokumentovaného (příloha B) aplikačního rozhraní pouze za účelem získání nebo odeslání dat. Nedochází zde ke znovu načítání celého HTML kódu, jako v případě klasického schématu webových aplikací, a o vykreslení jednotlivých částí se stará přímo klientská aplikace. Výhodou tohoto přístupu je, že reakce aplikace působí rychle a je jasně oddělená klientská a serverová část. Mezi nevýhody patří například to, že s aplikací nelze pracovat, dokud nedojde ke kompletnímu načtení klientského kódu, což může zejména u rozsáhlejších aplikací představovat problém. Implementovaná aplikace není příliš komplexní, takže prvotní čas načtení stránky je zanedbatelný.

Tvorbu výsledného kódu aplikace zajišťuje Webpack. Jedná se o nástroj pro zapouzdření moderních JavaScriptových aplikací, který vytvoří graf všech závislostí aplikace a sestaví je do výsledného souboru.[6] Počáteční konfigurace tohoto nástroje není úplně jednoduchá, ale existuje velké množství ukázkových konfigurací, které se dají přizpůsobit pro potřeby dané aplikace. Na rozdíl od konkurenčního nástroje Browserify[2], který je schopný vyřešit závislosti pouze pro JavaScriptové knihovny a ostatní typy souborů je potřeba řešit pomocí jiných nástrojů, si Webpack se správnou konfigurací (soubor *webpack.config.js*) poradí téměř s jakýmkoliv souborem.

2. `http://browserify.org/`

4.4 Serverové aplikační rozhraní

Serverová část využívá aplikační rámec Node.js. Jde o asynchronní, událostmi řízené běhové prostředí pro JavaScript.[7] Serverové aplikační rozhraní je implementováno pomocí knihovny Express.js. Express.js je aplikační webový rámec pro Node.js, který řeší typické věci očekávané od webového serveru: směrování požadavků pomocí URL, správa cookies a relace (session), zpracování chybných požadavků atd.[8, s. 88] Aplikační rozhraní se drží principů REST (Representational state transfer) architektury a je volně přístupné pro uživatele.

Architektura REST je běžně používána pro návrh aplikačních rozhraní moderních webových služeb. Klientské aplikace využívají funkce poskytované aplikačním rozhraním pro komunikaci s webovou službou. Pro aplikační rozhraní uplatňující principy architektury REST se používá označení REST API.[9, s. 5]

Tato architektura představuje způsob, jak jednoduše číst, zapsat, editovat nebo smazat data (CRUD[3] operace) ze serveru pomocí jednoduchých HTTP požadavků. Výměna dat probíhá v jednoduchém standardizovaném formátu JavaScript Object Notation (JSON), který lze snadno a rychle interpretovat. Při jednotlivých voláních se aplikace musí vůči aplikačnímu rozhraní autentizovat. Podrobný popis implementace zabezpečení je popsán v kapitolách 4.6 a 4.7. Dokumentace rozhraní je dostupná jako součást klientské aplikace a také jako příloha této práce.

4.5 Návrh databáze

4.5.1 PostgreSQL

Pro ukládání veškerých dat aplikace využívá na serveru databázi PostgreSQL (někdy označovanou také jako Postgres). PostgreSQL je relační databáze s otevřeným zdrojovým kódem (open-source), kterou lze nainstalovat na většinu operačních systémů. Její implementace splňuje vlastnosti ACID a plně podporuje cizí klíče, spojování tabulek, vytváření pohledů, spouští (triggerů) a uložených procedur.[10] Pří-

3. CRUD – create, read, update, delete

stup k databázi je ve webové aplikaci realizován pomocí knihovny Sequelize.[4]

Jedná se o knihovnu zajišťující objektově relační mapování (ORM). Tato programovací technika se používá pro automatickou konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem.[11] Použití této techniky usnadňuje vývojáři provádění běžných databázových operací jako je čtení, zápis, úprava a mazání dat. Do velké míry také odpadá nutnost pracovat přímo s dotazovacím jazykem dané relační databáze.

4.5.2 Relační databázový model

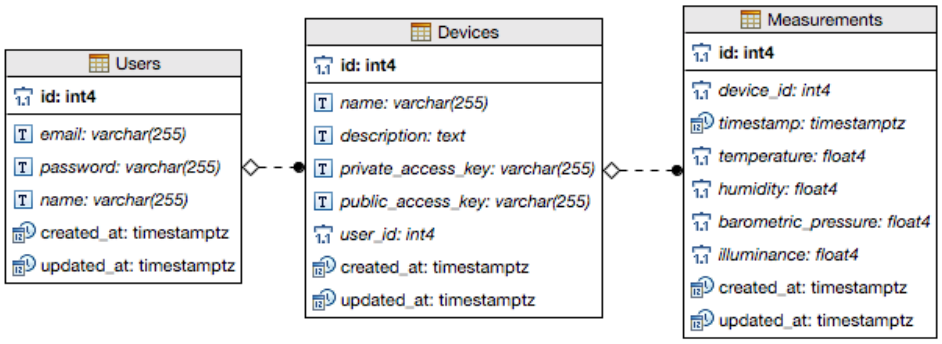
Relační databázový model používá k organizaci dat tabulky (relace). Každá tabulka má pevně stanovené sloupce (atributy), kde každý sloupec má definován svůj název, datový typ a velikost. Správně navržený databázový model by neměl obsahovat duplikátní data. Při návrhu databáze se v softwarovém inženýrství často využívá ER diagram, který je zobrazen na obrázku 4.2.

Databáze se skládá ze tří hlavních tabulek – *users*, *devices* a *measurements*.

- Tabulka *users*, obsahující atributy *email*, *password* a *name*, slouží k uchovávání informací o uživatelích.
- Tabulka *devices* obsahuje údaje o zařízeních a je ve vztahu 1:N s tabulkou *users*. Při vytváření nového záznamu se automaticky vygeneruje pár privátního a veřejného klíče a uloží se do atributů *private_access_key* a *public_access_key*.
- Tabulka *measurements* slouží k ukládání naměřených hodnot z jednotlivých zařízení. Kardinalita tabulky je 1:N vůči tabulce *devices*. Lze do ní ukládat naměřenou teplotu (*temperature*), atmosferický tlak (*barometric_pressure*), vlhkost vzduchu (*humidity*) a intenzitu světla (*illuminance*).

Každá tabulka navíc obsahuje atributy *created_at* a *updated_at*, které obsahují informaci o času vytvoření a poslední změně záznamu.

4. `http://docs.sequelizejs.com/en/v3/`



Obrázek 4.2: ER diagram databáze

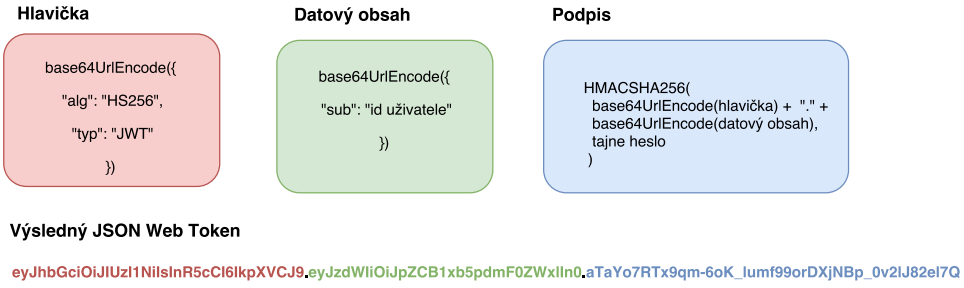
4.6 Autentizace klientské aplikace

Autentizace uživatele probíhá prostřednictvím uživatelského jména a hesla. Při registraci nového uživatele se vygeneruje a uloží do databáze otisk hesla pomocí funkce bcrypt založené na šifře Blowfish. Tato funkce je také adaptivní, což znamená, že můžeme parametrem navýšit počet iterací, čímž je zajištěna ochrana proti útokům hrubou silou i do budoucna.[12]

Samotný protokol HTTP nedokáže rozpoznat, jestli spolu dotazy na server jakkoliv souvisí, je proto nutné přenášet informace pro autentizaci uživatele s každým požadavkem na server. Není bezpečné přímo ukládat přístupové údaje uživatele, proto aplikační server při autentizaci vystaví uživateli přístupový token. Pro tvorbu a ověřování přístupových tokenů byl zvolen otevřený standard JSON Web Tokens. Jedná se o kompaktní a bezpečný způsob přenášení informací jako JSON objekt mezi komunikujícími stranami. Tyto informace mohou být ověřeny a lze jim důvěřovat, protože jsou digitálně podepsané.[13]

JSON Web Token se skládá ze tří částí: hlavičky, datového obsahu a podpisu. Hlavička obvykle obsahuje typ tokenu (JWT) a použitý šifrovací algoritmus. Datový obsah nese informace o uživateli a případně další metadata. Podpis vzniká tak, že první dvě části nezávisle na sobě převedeme do kódování Base64 a výsledné řetězce poté zašifrujeme pomocí tajného hesla definovaného na serveru a algoritmu specifikova-

ného v hlavičce tokenu. Výsledkem jsou tři řetězce ve formátu Base64 oddělené tečkou, které lze snadno přenášet pomocí HTTP požadavků.



Obrázek 4.3: Struktura JSON Web Tokenu

Aplikace se poté dále prokazuje vytvořeným tokenem v HTTP hlavičce s využitím Bearer schématu. Server tuto hlavičku dekóduje a ověří její platnost. Pokud ověření selže, odpoví stavovým kódem *401 Unauthorized*. Při odeslání přihlašovacího formuláře posílá klient na server citlivé údaje, které lze snadno odposlechnout a zneužít. Pro reálné nasazení aplikace je proto vhodné ke komunikaci použít šifrovaný protokol HTTPS.

4.7 Autentizace IoT zařízení

Metoda autentizace pomocí JSON Web Tokens popsaná v kapitole 4.6 by byla pro autentizaci IoT[5] zařízení rovněž vhodná, pokud by aplikační rozhraní bylo určené pro komunikaci s konkrétními typy zařízení. Aplikační rozhraní je však od začátku koncipováno jako univerzální, a nastává zde problém s výpočetním výkonem slabších jednoúčelových systemů. Ty nejsou schopné vést komunikaci skrze šifrovaný protokol HTTPS a mohlo by tak dojít ke zneužití přihlašovacích údajů pro přístup do webové aplikace. Je proto zapotřebí autentizaci zařízení ošetřit jiným, výpočetně méně náročným způsobem, při kterém nebudou citlivé autentizační údaje přenášeny v čitelné podobě.

Jako vhodné řešení se jeví použít HTTP schéma založené na autentizačním kódu zprávy HMAC, počítaném pomocí hašovací funkce

5. *Internet of Things* – označení pro propojení vestavěných zařízení s Internetem

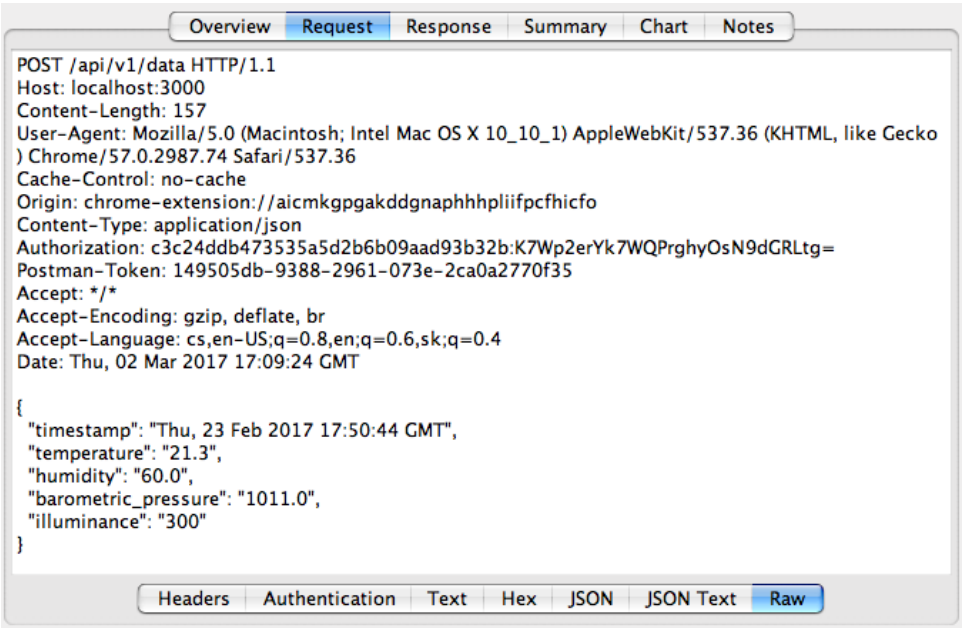
v kombinaci s tajným šifrovacím klíčem.[14] Implementovaný způsob autentizace je inspirovaný přístupem, který pro autentizaci svého programového aplikačního rozhraní využívá např. firma Amazon.com, Inc[6].

Autentizační HTTP hlavička se skládá ze dvou částí - veřejného klíče a podpisu, vzájemně oddělených dvojtečkou. Server si nejprve ověří, zda požadavek obsahuje HTTP hlavičky *X-Authorization* a *Date*, na základě které ověří, jestli byl požadavek obdržen v 15 minutovém intervalu jeho platnosti. Dále se kontroluje, zda jsou přijatá data ve formátu JSON a pokud ano, server pomocí obdrženého veřejného klíče vyhledá v databázi příslušné zařízení a zjistí si korespondující privátní klíč. Dalším krokem je ověření podpisu, který je generován následujícím algoritmem:

StringToSign = HTTP-Verb + "\n" + Content-MD5 + "\n" + Content-Type + "\n" + Date;
Signature = Base64(HMAC-SHA1(UTF-8-Encoding-Of(StringToSign), SecretAccessKey));

Pokud je přijatý podpis shodný s podpisem vygenerovaným na serveru, je požadavek přijat. V opačném případě je odmítnut s příslušným stavovým kódem a chybovou hláškou. Implementace se nachází v souboru *server* ▶ *middleware* ▶ *auth-device.js*.

6. `http://docs.aws.amazon.com/AmazonS3/latest/dev/RESTAuthentication.html`



Obrázek 4.4: Ukázka HTTP požadavku v programu Charles

4.8 Nasazení aplikace

Aplikace pro svůj běh vyžaduje nainstalovaný Node.js (vyvíjeno a testováno na verzi 6.9.2). Dále musí být na počítači běžící instance databáze PostgreSQL. Jako první je potřeba nainstalovat veškeré závislosti aplikace (nakonfigurovány v souboru *package.json*) pomocí příkazu:

```
1 $ npm install
```

Dále je potřeba vytvořit databázi příkazem:

```
1 $ createdb bc
```

Kde *bc* je jméno databáze. Výchozí adresa a port databáze jsou nastaveny na *127.0.0.1:5432*. Tyto hodnoty včetně jména databáze je možno upravit v souboru *config* ▶ *config.json*. V tomto souboru je také potřeba nastavit jméno a heslo pro přístup k databázi. V dalším kroku inicializujeme strukturu databáze pomocí příkazu:

```
1 $ sequelize db:migrate
```

Tento příkaz využívá knihovnu Sequelize a vytvoří potřebné databázové tabulky na základě modelů definovaných ve složce *server* ▶ *models*. Následně je potřeba spustit nástroj webpack příkazem:

```
1 $ npm run bundle
```

Poté už je možné aplikaci spustit příkazem:

```
1 $ npm start
```

Aplikace je lokálně přístupná na adrese `http://127.0.0.1:3000`.

5 Meteorologická stanice na platformě Arduino

5.1 Použité technologie

5.1.1 Platforma Arduino

Arduino je open-source elektronická platforma[15] osazená různými typy mikrokontrolérů ATMega vyráběnými firmou Atmel. Arduino je vhodné jako jednoduchá a levná prototypovací platforma pro tvorbu nejrůznějších elektronických zařízení. Mezi typické vybavení desek Arduino patří v závislosti na zvoleném typu několik vstupně-výstupních pinů, sériových rozhraní, USB konektor, napájecí konektor, EEPROM a flash paměť. Díky velkému množství desek Arduino dostupných na trhu, není problém vybrat typ, který vyhovuje požadavkům daného projektu.

Dále jsou k dispozici tzv. shield moduly. Jedná se o přídavné součástky, u nichž není zapotřebí žádného pájení nebo složitého sestavování a lze je jednoduše připojit k základní desce a rozšířit tak její funkce např. o podporu Wi-Fi připojení, Ethernetu, Bluetooth nebo GSM sítě.

5.1.2 Arduino IDE

Arduino IDE[1] je multiplatformní vývojové prostředí s otevřeným zdrojovým kódem dostupné pro Windows, Mac OS a Linux. Pomocí tohoto vývojového prostředí lze kód zkompilovat a naprogramovat jej do desky Arduino. Prostředí také nabízí vestavěný textový editor se základní podporou zvýrazňování syntaxe. Dostupný je rovněž Arduino Web Editor. Jedná se o webovou verzi vývojového prostředí dostupnou bez nutnosti jakékoliv instalace přímo ve webovém prohlížeči. Editor je po přihlášení dostupný na adrese `https://create.arduino.cc/editor`.

1. *Integrated Development Environment* – Vývojové prostředí

5.1.3 Implementace

K psaní programů pro Arduino se používá programovací jazyk Wiring vycházející z C++. Lze také programovat přímo v C++, čehož se často využívá např. při tvorbě nových knihoven. U Arduina užíváme dvě základní funkce – *setup()* a *loop()*.

- Funkce *setup()* se zavolá pouze jednou při spuštění Arduina. Typicky se do této funkce umisťuje základní inicializace proměnných a periferií.
- Funkce *loop()* je hlavní funkcí celého programu. Zavolá se po provedení funkce *setup()* a je opakovaně volána až do vypnutí zařízení.

Jednou z věcí, které bylo potřeba na začátku vyřešit, je generování podpisu HTTP požadavku. Jako první jsem vyzkoušel kryptografickou knihovnu Cathedrow/Cryptosuite[2], která obsahuje implementaci potřebných hašovacích funkcí. Tato knihovna je ovšem 7 let stará a nelze ji v nejnovější verzi vývojového prostředí zkompilovat. Vyzkoušel jsem tedy knihovnu spaniakos/Cryptosuite[3], která je aktualizovanou verzí předešlé knihovny a opravuje problémy s kompilací. Její součástí je navíc podrobná dokumentace s ukázkami použití jednotlivých kryptografických algoritmů. Narazil jsem u ní ovšem na problém, že hašovací funkce HMAC-SHA-1 vracela jiný výstup, než jsem očekával. Nejprve jsem si myslel, že špatně používám funkce knihovny a chyba je na mé straně, ale poté jsem zjistil, že nejsem jediný[4], kdo má tento problém, a jedná se o chybu přímo v knihovně.

Nakonec jsem tedy skončil u knihovny adafruit/Cryptosuite[5], která je odlehčenou verzí obou předešlých a obsahuje implementaci pouze hašovacích funkcí SHA-1 a HMAC-SHA-1, které pro vygenerování podpisu potřebuji. S touto knihovnou jsem již žádné problémy neměl a výstup hašovacích funkcí byl správný.

Aby bylo možné vygenerovat ověřovací podpis dle algoritmu popsaného v kapitole 4.7, je zapotřebí ještě hašovací funkce MD5, funkce

2. `https://github.com/Cathedrow/Cryptosuite`
3. `https://github.com/spaniakos/Cryptosuite`
4. `https://github.com/spaniakos/Cryptosuite/issues/7`
5. `https://github.com/adafruit/Cryptosuite`

pro převod do URL kódování a implementace kódování Base64. Pro tvorbu MD5 hašů jsem využil knihovnu ArduinoMD5[6]. Pro převod do URL kódovaní není potřeba celé knihovny a funkce je následující:

```
1  String URLEncode(const char* msg)
2  {
3   const char *hex = "0123456789ABCDEF";
4   String encodedMsg = "";
5
6   while (*msg!='\0'){
7    if( ('a' <= *msg && *msg <= 'z')
8     || ('A' <= *msg && *msg <= 'Z')
9     || ('0' <= *msg && *msg <= '9') ) {
10     encodedMsg += *msg;
11   } else {
12    encodedMsg += '%';
13    encodedMsg += hex[*msg >> 4];
14    encodedMsg += hex[*msg & 15];
15   }
16   msg++;
17  }
18  return encodedMsg;
19 }
```

Poslední částí je aplikace funkce Base64. Využil jsem zjednodušené implementace[7] pro řetězce stejné délky (20 znaků), které jsou výstupem hašovací funkce HMAC-SHA1.

Dále bylo potřeba vyřešit práci s reálným časem a aktuálním datem, pro vytvoření časové značky (timestamp), která je nezbytnou součástí vysílaného HTTP požadavku. Častým řešením tohoto problému je použití přídavného obvodu reálného času (RTC – Real Time Clock). Tyto moduly bývají energeticky nenáročné a udržují relativně přesný čas. Vlivem okolních jevů (např. teploty) může dojít k ovlivnění frekvence oscilátoru, což může postupem času způsobit odchylky až v řádech minut. Vzhledem k tomu, že měření hodnot probíhá každou hodinu a nepotřebuji časový údaj úplně přesně, odchylka by šla tolerovat. Problém by mohl nastat v případě, že by se čas zařízení rozešel s časem serveru o více než 15 minut. Poté by server začal příchozí požadavky odmítat.

---

6. https://github.com/tzikis/ArduinoMD5
7. https://github.com/adafruit/Adafruit-Tweet-Receipt/blob/master/Gutenbird.ino

Rozhodnul jsem se tedy obvod reálného času nepoužít, a využil jsem toho, že zařízení je připojeno k internetu. Aktuální datum a čas zjišťuji vysláním GET požadavku na server. Odpověď serveru obsahuje HTTP hlavičku *Date*, ze které načítám a ukládám aktuální časovou značku. Tento požadavek je vysílán bezprostředně před tím, než se na server odešle POST požadavek obsahující aktuální naměřená data. Časová odchylka je v řádech vteřin.

5.2 Hardwarová implementace

Jako základní deska celého zařízení se používá Arduino MEGA 2560 (dále jen „Arduino“). Celý prototyp jednoduché meteorologické stanice je realizován na nepájivém poli. K Arduinu je připojený Wi-Fi modul, teplotní a vlhkostní čidlo, senzor atmosferického tlaku a fotorezistor pro měření intenzity světla. Jednotlivé součástky jsou podrobněji popsány v následujících kapitolách.

5.2.1 Arduino MEGA 2560

Arduino MEGA 2560 slouží jako základní výpočetní prvek celého zařízení. Tato deska za cenu 35€ (ke dni 6. 3. 2017) poskytuje uspokojivý výpočetní výkon a velké množství digitálních I/O pinů pro komunikaci se senzory.

Mikrokontrolér	ATmega2560
Pracovní napětí	5V
Vstupní napětí	7-12V
Max. vstupní napětí	6-20V
Digitální I/O piny	54
PWM I/O piny	14
Analogové vstupy	16
DC proud na pin	40 mA
Flash paměť	256 KB (8 KB využito pro bootloader)
SRAM	8 KB
EEPROM	4 KB
Krystal	16 MHz

Tabulka 5.1: Technická specifikace desky Arduino MEGA 2560[16]

5.2.2 Wi-Fi modul

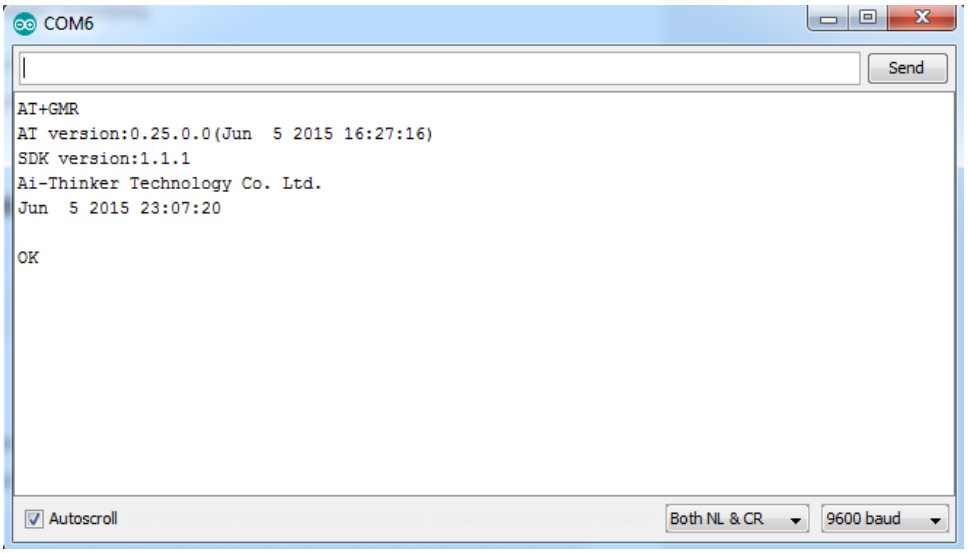
Pro připojení zařízení k bezdrátové síti jsem využil modul *ESP8266*. Jedná se o malý (15 x 25 mm) Wi-Fi modul, který zvládá komunikaci v normách 802.11 b/g/n, a je schopný pracovat i jako samostatná výpočetní jednotka. Napájecí napětí tohoto modulu je 3.3V a je vhodné použít externí zdroj, protože maximální odběr proudu může být až 250 mA, přičemž integrovaný 3.3V zdroj na desce Arduina tyto hodnoty nedokáže pokrýt.

Modul ESP8266 je na trhu v několika variantách a nelze přesně určit, jaký firmware a výchozí konfiguraci bude součástka obsahovat. Komunikace s modulem se provádí sériovou komunikací pomocí tzv. AT příkazů. Po propojení modulu s Arduinem je potřeba pomocí vývojového prostředí nahrát do Arduina jednoduchý program, který nám umožní sériovou komunikaci.

```
1  #include <SoftwareSerial.h>
2
3  void setup() {
4    Serial2.begin(115200); //esp8266
5    Serial.begin(115200);
6  }
7
8  void loop() {
9    while (Serial2.available())Serial.write(Serial2.read
         ());
10   while (Serial.available())Serial2.write(Serial.read
         ());
11 }
```

Poté lze navázat komunikaci pomocí funkce Serial Monitor integrované přímo ve vývojovém prostředí. Je potřeba správně nastavit modulační rychlost (baud rate) a znaky pro zalomení konců řádků. Tyto hodnoty se odvíjí od verze firmwaru a standardně je modulační rychlost nastavená na hodnotu 115200 Bd nebo 9600 Bd. Znaky pro zalomení konců řádků mohou být carriage return („návrat vozíku“), new line (nový řádek) nebo kombinace obou předchozích a je potřeba vyzkoušet různé varianty, dokud modul nevrátí smysluplnou odpověď.

Pokud je vše nastaveno správně, po odeslání příkazu *AT* modul odpoví hláškou *OK*. Pomocí příkazu *AT+GMR* lze vypsat firmware zařízení.



Obrázek 5.1: Výchozí verze firmwaru

Výchozí hodnota modulační rychlosti byla u mého modulu z výroby nastavena na 115200 Bd a bylo potřeba ji změnit na 9600 Bd, protože při původní rychlosti dochází při asynchronní sériové komunikaci k velkému množství chyb v přenášených datech. Tuto změnu lze provést pomocí příkazu:

```
1 AT+UART_DEF=9600,8,1,0,0
```

Kde jednotlivé parametry postupně za sebou představují:

- modulační rychlost v baudech (baud rate)
- datové bity (data bits)
- stop bity (stop bits)
- parita (parity)
- řízení toku dat (flow control)

Dalším nastavením, které je vhodné změnit, je provozní mód modulu. Dostupné provozní módy jsou:

- 1 = Klient (Station mode)

- 2 = Přístupový bod (Access Point)
- 3 = Klient + Přístupový bod (Station mode)

Výchozí hodnota je číslo tři. Změnu módu pouze na klienta vykoná příkaz:

```
1 AT+CWMODE=1
```

Po prvotním nastavení jsem dále pro jednodušší komunikaci s modulem využil knihovnu WiFiEsp[8], která je inspirovaná knihovnou pro oficiální WiFi shield a mnohá volání funkcí jsou zde totožná. Knihovna obsahuje podrobnou dokumentaci včetně ukázkových kódů pro nejčastější případy použití. Výhodou této knihovny je také to, že využívá výše zmíněných AT příkazů a není proto potřeba do WiFi modulu nahrávat žádný speciální firmware.

5.2.3 Senzory

Teploměr a vlhkoměr DHT11

Digitální čidlo vlhkosti vzduchu a teploty DHT11 dokáže měřit teplotu v rozsahu 0 - 50 °C s přesností ±1 °C. Vlhkost měří v rozmezí 20 - 90% s přesností ±4%. Načítání dat z tohoto senzoru zajišťuje knihovna DHT-sensor-library[9].

Senzor barometrického tlaku BMP180

Senzor barometrického tlaku BMP180 je schopný kromě měření tlaku i měření teploty. Dále pak lze získat také informace o aktuální nadmořské výšce, kterou ovšem nevyužívám. Přesnost měření tlaku je dle informací výrobce[10] 300 - 1100 hPa, což odpovídá nadmořské výšce +9000 m až -500 m. Přesnost měření barometrického tlaku je -4 až +2 hPa. Okolní teplotu je možno měřit v rozsahu 0 až +65 °C s přesností ±2 °C. Obsluhu senzoru realizuji pomocí knihovny Adafruit-BMP085-Library[11].

---

8. https://github.com/willdurand/EspWiFi
9. https://github.com/adafruit/DHT-sensor-library
10. https://cdn-shop.adafruit.com/datasheets/BST-BMP180-DS000-09.pdf
11. https://github.com/adafruit/Adafruit-BMP085-Library

Snímač intenzity osvětlení BH1750

Prvotní nápad byl použít pro měření intenzity osvětlení fotorezistor. Po hlubším prozkoumání této problematiky jsem však zjistil, že pro převod výstupního napětí z fotorezistoru na jednotky luxů je potřeba mít referenční zařízení, podle kterého by se převod na luxy dal zkalibrovat. I tak by však výstup z fotorezistoru nebyl příliš přesný.

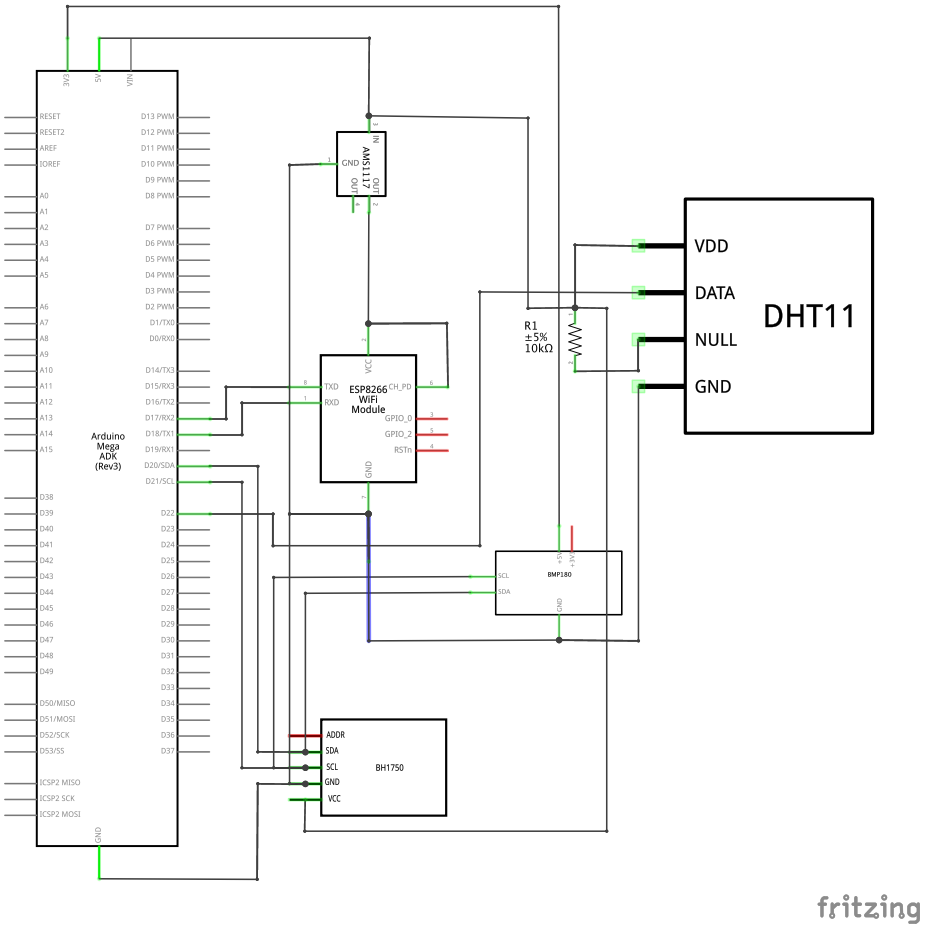
Rozhodl jsem se tedy použít snímač intenzity osvětlení BH1750, jehož výstupní data jsou přímo v jednotkách luxů. Při programování jsem pro získání dat využil knihovnu BH1750[12].

5.2.4 Schéma zapojení

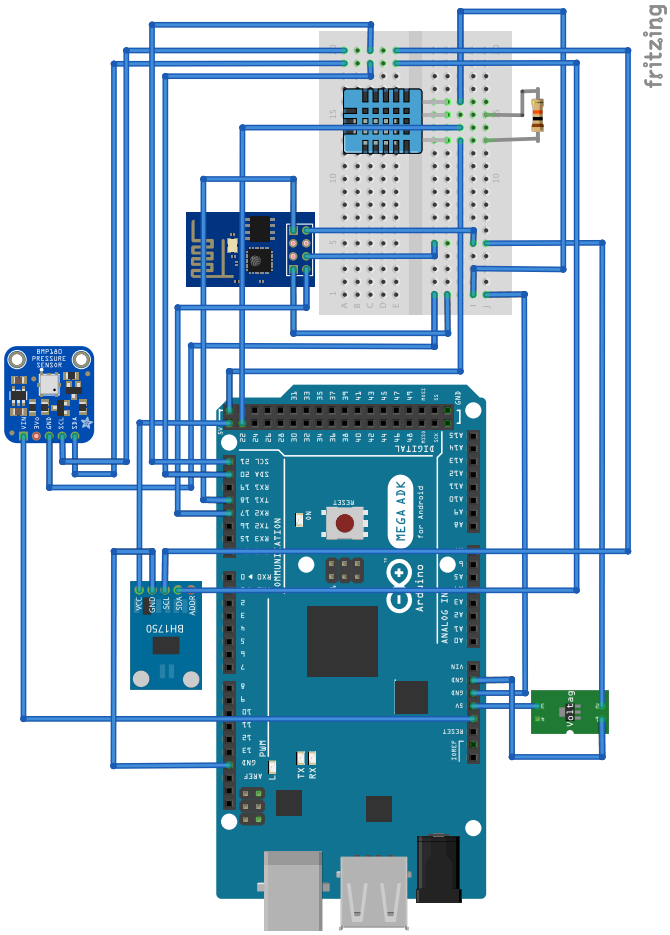
Součástka	Počet kusů
Arduino Mega 2560	1
Teploměr a vlhkoměr DHT11	1
Wi-Fi modul ESP8266	1
Senzor barometrického tlaku BMP180	1
Stabilizátor napětí 4,5V..7V na 3,3V/0,5A	1
Rezistor 10 kΩ	1
Snímač intenzity osvětlení BH1750	1

Tabulka 5.2: Seznam použitých součástek

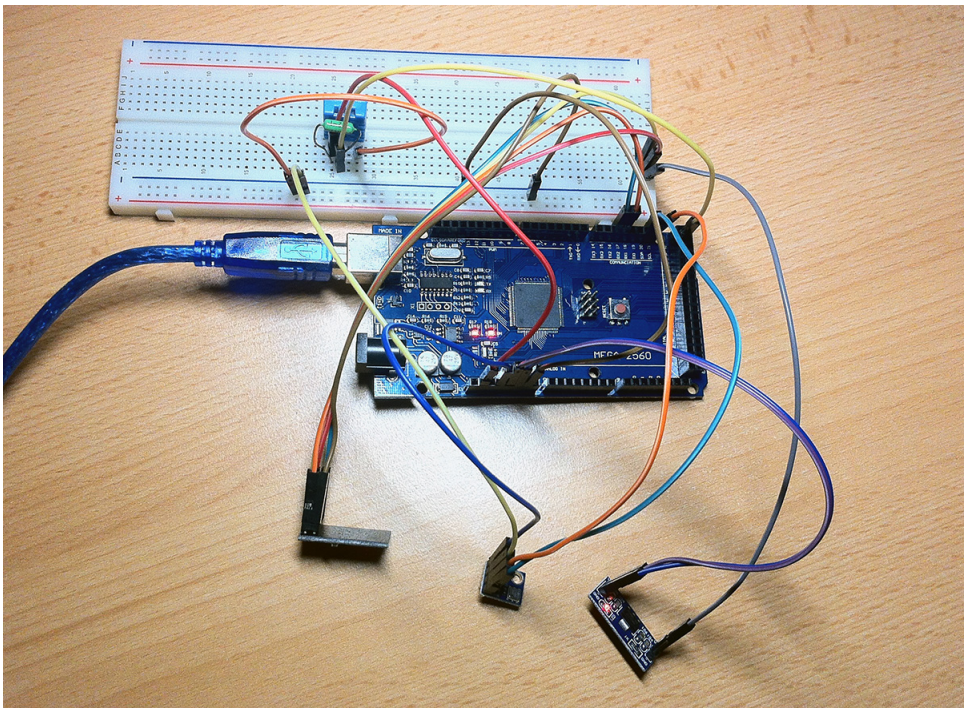
12. `https://github.com/claws/BH1750`



Obrázek 5.2: Schéma zapojení č. 1



Obrázek 5.3: Schéma zapojení č. 2



Obrázek 5.4: Prototyp zařízení

5.2.5 Zprovoznění zařízení

Pro nahrání řídicího programu do Arduina doporučuji využít oficiální vývojové prostředí Arduino IDE. V případě, že se zapojení nedrží přesně schématu, je zapotřebí v kódu programu upravit čísla příslušných pinů.

Pro správnou funkčnost je potřeba jako první nakopírovat příslušné knihovny, které program využívá. Knihovny se nacházejí ve složce Libraries v příloze **source_arduino.zip**. Umístění knihoven se liší v závislosti na operačním systému. V operačních systémech Windows a Mac OS jsou knihovny umístěny ve složce „Arduino“, která se nachází v uživatelské složce „Dokumenty“. Na Linuxu jsou knihovny typicky umístěny ve složce /home/<username>/Sketchbook.

Po nakopírování knihoven je potřeba restartovat vývojové prostředí.

5.3 Testování

Výsledné zařízení jsem testoval s lokálně spuštěnou webovou aplikací. Měření a odesílání dat na aplikační rozhraní probíhalo v hodinovém intervalu po dobu dvou týdnů. V průběhu testování nedošlo k žádné chybě na straně zařízení, ani webové aplikace.

I přesto, že celý průběh testování proběhl hladce, nemusel tento způsob odhalit všechny potenciální chyby. Do budoucna zde vidím prostor ke zlepšení např. formou automatických testů.

6 Závěr

Při tvorbě této práce jsem se seznámil s aplikačním rámcem Node.js a implementoval webovou aplikaci pro zobrazení dat z meteorologické stanice. Dle zadání jsem navrhl a s pomocí moderních technologií vytvořil jednoduché uživatelské rozhraní aplikace, které je přístupné pro uživatele desktopových i mobilních zařízení. Dále jsem se zaobíral problematikou serverových aplikačních rozhraní od návrhu až po samotnou implementaci a tvorbu dokumentace.

Druhou částí práce bylo navrhnout, sestavit a naprogramovat prototyp meteorologické stanice na platformě Arduino, umožňující sběr několika druhů dat, a demonstrovat použití aplikačního rozhraní jejím napojením na webovou aplikaci. Všechny cíle vytyčené v zadání práce se mi podařilo splnit a během testování jak webové aplikace, tak i sestaveného zařízení jsem nenarazil na žádný závažný problém.

Webová aplikace nabízí prostor pro další vývoj, a chtěl bych ji tudíž dále rozšiřovat nad rámec bakalářské práce. Do budoucna mám v úmyslu aplikaci rozšířit např. o možnost zobrazení historických dat v různých časových úsecích, přidat podporu pro sběr dalších druhů dat a pokrýt aplikaci automatickými testy.

Bibliografie

1. *About us. VANE Geospatial Platform* [online] [cit. 2016-12-14]. Dostupné z: http://owm.io/about.
2. *About Our Data | Weather Underground* [online] [cit. 2016-12-14]. Dostupné z: https://www.wunderground.com/about/data.
3. *Bootstrap The most popular HTML, CSS, and JS framework in the world.* [online] [cit. 2017-03-01]. Dostupné z: https://v4-alpha.getbootstrap.com/.
4. *ECMAScript® 2015 Language Specification – ECMA-262 6th Edition* [online] [cit. 2017-02-08]. Dostupné z: http://www.ecma-international.org/ecma-262/6.0/#sec-scop.
5. *Tutorial: Intro To React* [online] [cit. 2017-02-08]. Dostupné z: https://facebook.github.io/react/tutorial/tutorial.html#what-is-react.
6. *Concepts* [online] [cit. 2017-02-08]. Dostupné z: https://webpack.js.org/concepts/.
7. *About Node.js®* [online] [cit. 2017-02-07]. Dostupné z: https://nodejs.org/en/about/.
8. WILSON, Jim R. *Node.js the Right Way Practical, Server-Side javascript That Scales*. Pragmatic Bookshelf, 2013. ISBN 1937785734.
9. MASSE, Mark. *REST API Design Rulebook*. O'Reilly Media, 2011. ISBN 1449310508,9781449310509.
10. *PostgreSQL: About* [online] [cit. 2017-03-06]. Dostupné z: https://www.postgresql.org/about/.
11. *Objektově relační mapování* [online] [cit. 2017-03-14]. Dostupné z: https://cs.wikipedia.org/wiki/Objektov%C4%9B_rela%C4%8Dn%C3%AD_mapov%C3%A1n%C3%AD.
12. PROVOS, Neils; MAZIERES, David. *A Future-Adaptable Password Scheme* [online] [cit. 2017-01-28]. Dostupné z: http://www.openbsd.org/papers/bcrypt-paper.pdf.
13. JONES, Michael; BRADLEY, John; SAKIMURA, Nat. *JSON Web Token (JWT)* [RFC 7519]. RFC Editor, 2015. Request for Comments, č. 7519. Dostupné z DOI: 10.17487/rfc7519.

14. KRAWCZYK, Dr. Hugo; BELLARE, Mihir; CANETTI, Ran. *HMAC: Keyed-Hashing for Message Authentication* [RFC 2104]. RFC Editor, 1997. Request for Comments, č. 2104. Dostupné z DOI: 10.17487/rfc2104.

15. *What is Arduino?* [online] [cit. 2017-03-06]. Dostupné z: https://www.arduino.cc/en/Guide/Introduction.

16. *Arduino Mega 2560 Rev3* [online] [cit. 2017-03-06]. Dostupné z: https://store.arduino.cc/product/A000067.

A Seznam elektronických příloh

- **source_web.zip**: archiv obsahující zdrojové kódy webové aplikace
- **source_arduino.zip**: archiv obsahující zdrojové kódy pro zařízení Arduino
- **scheme.zip**: archiv obsahující schéma zapojení z programu Fritzing

B Dokumentace REST API

Všechny požadavky na aplikační rozhraní kromě adresy */api/v1/data* vyžadují **platný JSON Web Token** pro autentizaci.

B.1 Měření (Measurements)

B.1.1 Načtení dat od konkrétního zařízení

Načte data za posledních 24 hodin od zařízení s konkrétním *id*.
Metoda: GET

```
1 /api/v1/measurements/:device_id
```

Parametr	Typ	Popis
device_id	Number	ID zařízení, kterému data patří.

Kódy odpovědí

- **Success 200**

  Vrací pole objektů obsahující naměřená data.

  Ukázka odpovědi:


```
[{
 "id":1,
 "device_id":2,
 "timestamp":"2017-03-18T14:41:09.000Z",
 "temperature":10.5,
 "humidity":59,
 "barometric_pressure":1004,
 "illuminance":800,
 "created_at":"2017-03-18T14:41:09.000Z",
 "updated_at":"2017-03-18T14:41:09.000Z"
}]
```


- **Error 401**

  Neoprávněný přístup. Ověření JWT selhalo.

B.2 Data

Přijme data (měření) od konkrétního zařízení.
Metoda: POST

```
1 /api/v1/data
```

Požadavek musí obsahovat HTTP hlavičku *Date* pro ověření, jestli je požadavek přijat v **15 minutovém intervalu jeho platnosti**. Hlavička *X-Authorization* je požadována pro ověření totožnosti zařízení. Formát autorizační hlavičky je: *verejny_klic_zarizeni:podpis*

Podpis je generován následujícím algoritmem:

```
1 StringToSign = HTTP-Verb + "\n" + Content-MD5 + "\n"
      + Content-Type + "\n" + Date;
2 Podpis = Base64( HMAC-SHA1( UTF-8-Encoding-Of(
      StringToSign ), SecretAccessKey ) );
```

Akceptovaný formát dat je JSON.

Datové parametry

Parametr	Požadováno	Popis
timestamp	**Ano**	Časová značka měření dat.
temperature	Ne	Teplota ve stupních Celsia.
humidity	Ne	Vlhkost v %.
barometric_pressure	Ne	Atmosferický tlak v hPa.
illuminance	Ne	Intenzita osvětlení v luxech.

Kódy odpovědí

- **Success 200**

```
1 {success: 'Measurement was received successfully.';}
```

- **Error 401**

```
1 {message: 'Bad signature'}
```

- **Error 500**

Parametr	Popis
message	Timestamp is required
message	Request is older than 15 minutes
message	Content-Type is not application/json
message	Device was not found

B.3 Zařízení (Devices)

B.3.1 Načtení všech zařízení

Načte všechna zařízení patřící autentizovanému uživateli.
Metoda: GET

```
1 /api/v1/devices
```

Kódy odpovědí

- **Success 200**
  Vrací pole objektů obsahující zařízení, která patří autentizovanému uživateli.

Parametr	Popis
id	ID zařízení.
name	Jméno zařízení.
description	Popis zařízení.
private_access_key	Soukromý klíč zařízení.
public_access_key	Veřejný klíč zařízení.
user_id	ID uživatele, kterému zařízení patří.
created_at	Časová známka vytvoření záznamu v databázi.
updated_at	Časová známka úpravy záznamu v databázi.

- **Error 401**

  Neoprávněný přístup. Ověření JWT selhalo.

B.3.2 Načtení jednoho zařízení

Metoda: GET

```
1 /api/v1/devices/:id
```

Parametr	Typ	Popis
id	Number	ID zařízení.

Kódy odpovědí

- **Success 200**
  Objekt obsahující informace o konkrétním zařízení.

Parametr	Popis
id	ID zařízení.
name	Jméno zařízení.
description	Popis zařízení.
private_access_key	Soukromý klíč zařízení.
public_access_key	Veřejný klíč zařízení.
user_id	ID uživatele, kterému zařízení patří.
created_at	Časová známka vytvoření záznamu v databázi.
updated_at	Časová známka úpravy záznamu v databázi.

- **Error 401**

  Neoprávněný přístup. Ověření JWT selhalo.

B.3.3 Vytvoření nového zařízení

Metoda: POST

```
1 /api/v1/devices/:id
```

Datové parametry

Parametr	Požadováno	Popis
name	**Ano**	Jméno zařízení.
description	Ne	Popis zařízení.

Kódy odpovědí

- **Success 200**

Parametr	Hodnota
success	true
message	Device was created successfully!
device	Objekt s parametry zařízení.

- **Error 401**

  Neoprávněný přístup. Ověření JWT selhalo.

B.3.4 Úprava zařízení

Metoda: PUT

```
1 /api/v1/devices/:id
```

Parametr	Typ	Popis
id	Number	ID zařízení k úpravě.

Datové parametry

Parametr	Požadováno	Popis
name	**Ano**	Jméno zařízení.
description	Ne	Popis zařízení.

Kódy odpovědí

- **Success 200**

Parametr	Hodnota
success	true
message	Device was updated successfully!
device	Objekt s upravenými parametry zařízení.

- **Error 400**
  Jméno zařízení nebylo poskytnuto. Ověření selhalo.

Parametr	Hodnota
success	false
message	Check the form for errors.
errors	Please provide name of your device.

- **Error 401**

  Neoprávněný přístup. Ověření JWT selhalo.

B.3.5 Smazání zařízení

Metoda: DELETE

```
1 /api/v1/devices/:id
```

Parametr	Typ	Popis
id	Number	ID zařízení pro smazání.

Kódy odpovědí

- **Success 200**

Parametr	Hodnota
success	true
message	Device was deleted successfully!
device	Objekt s parametry smazaného zařízení.

- **Error 401**

  Neoprávněný přístup. Ověření JWT selhalo.

B.3.6 Reset soukromého klíče

Resetuje soukromý klíč daného zařízení.
Metoda: PUT

```
1 /api/v1/devices/reset/:id
```

Parametr	Typ	Popis
id	Number	ID zařízení.

Kódy odpovědí

- **Success 200**

Parametr	Hodnota
success	true
message	Private key was reseted successfully!
privateAccessKey	Soukromý klíč.

- **Error 401**

  Neoprávněný přístup. Ověření JWT selhalo.

B.4 Uživatelské účty (User Accounts)

B.4.1 Registrace nového uživatele

Metoda: POST

```
1 /api/v1/auth/signup
```

Datové parametry

Parametr	Požadováno	Popis
name	**Ano**	Jméno uživatele.
email	**Ano**	E-mail uživatele.
password	**Ano**	Heslo.

Kódy odpovědí

- **Success 200**

Parametr	Hodnota
success	true
message	You have successfully signed up! Now you should be able to log in.

- **Error 400**

  Některé informace nebyly poskytnuty. Ověření selhalo.

Parametr	Hodnota
success	false
message	Check the form for errors.
errors	Pole s možnými chybami (jméno, email nebo heslo nebyly vyplněny)

- **Error 409**

  E-mail je již obsazen.

Parametr	Hodnota
success	false
message	Check the form for errors.
errors	This email is already taken.

B.4.2 Přihlášení uživatele

Metoda: POST

```
1 /api/v1/auth/login
```

Datové parametry

Parametr	Požadováno	Popis
name	**Ano**	Jméno uživatele.
email	**Ano**	E-mail uživatele.

Kódy odpovědí

- **Success 200**

Parametr	Hodnota
success	true
message	You have successfully logged in!
token	JSON Web Token
user	Parametry přihlášeného uživatele.

- **Error 400**

  Některé informace nebyly poskytnuty. Ověření selhalo.

Parametr	Hodnota
success	false
message	Check the form for errors.
errors	Pole s možnými chybami (jméno nebo heslo nebylo vyplněno).