Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Návrh zavlažovacího systému řízeného pomocí Raspberry Pi

Libor Selecký

Bakalářská práce

2019

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2018/2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Libor Selecký**
Osobní číslo:	**I16343**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Návrh zavlažovacího systému řízeného pomocí Raspberry Pi**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem práce je návrh a realizace automatického zavlažovacího systému pomocí Raspberry Pi. Teoretická část bakalářské práce bude věnována technologiím pro chytré domy se zaměřením na jejích řízení. Praktická část práce bude zaměřena na vlastní návrh automatického zavlažovacího systému pomocí Raspberry Pi. Systém bude řízen jednak na základně časového plánu, ale také na základně dat ze senzorů. Návrh předpokládá i využití takových komponent, které je možné začlenit do hlasových asistentek jako je Google Home či Amazon Echo. Pro základní vzdálené řízení bude dále nutné navrhnout vhodné aplikační rozhraní a webovou aplikaci.

Rozsah pracovní zprávy: **min 30 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

CHIN, Stephen a James L WEAVER Raspberry Pi with Java: programming the internet of things (IoT) New York: McGraw-Hill Eucation, [2016]. ISBN 978-0071842013
SCHWARTZ, Marco Building Smart Homes with Raspberry Pi Zero New York: Packt Publishing, [2016]. ISBN 978-1786466952

Vedoucí bakalářské práce: **Ing. Jan Fikejz, Ph.D.**
Katedra softwarových technologií

Datum zadání bakalářské práce: **31. října 2018**
Termín odevzdání bakalářské práce: **12. května 2019**

Univerzita Pardubice
Česká republika
L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Ing. Lukáš Čegan, Ph.D.
pověřený vedením katedry

V Pardubicích dne 14. prosince 2018

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 29. 11. 2019

Libor Selecký

PODĚKOVÁNÍ

Tímto bych rád poděkoval vedoucímu práce Ing. Janu Fikejzovi, Ph.D. za velmi cenné rady a připomínky v průběhu zpracování bakalářské práce. Dále také své rodině za neutuchající podporu a trpělivost během mého studia.

ANOTACE

Bakalářská práce se zabývá návrhem a realizací systému automatického zavlažování rostlin s využitím mini počítače Raspberry Pi. V teoretické části práce jsou krátce představeny technologie pro chytré domy a jejich řízení.

Praktická část se věnuje vývoji backend aplikace poskytující RESTové rozhraní pro správu senzorů, výstupů a plánování úloh. Automatické provádění úloh může být řízeno jednak časovým plánem, ale také na základě dat získaných ze senzorů. Další částí je vývoj webové aplikace pro vzdálené ovládání celého systému.

KLÍČOVÁ SLOVA

IoT, Raspberry Pi, zavlažovací systém, Java, REST, Spring Framework, Angular 8

TITLE

Design of irrigation system controlled by Raspberry Pi

ANNOTATION

The bachelor thesis deals with design and realization of automatic irrigation system with usage of minicomputer Raspberry Pi. The theoretical part shortly introduces technologies for smart homes and its control.

The practical part is dedicated to development of backend application providing a RESTful interface for management of sensors, outputs and tasks scheduling. Execution of automatic tasks can be controlled by time schedule or conditions dependent on data from sensors. The next part deals with development of web application for easy remote control of the whole system.

KEYWORDS

IoT, Raspberry Pi, irrigation system, Java, REST, Spring Framework, Angular 8

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

API	Application Programming Interface
CSS	Cascading Style Sheets
GPIO	General Purpose Input/Output
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IoT	Internet of Things
JPA	Java Persistence API
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MVC	Model View Controller
ORM	Object Relational Mapping
REST	Representational State Transfer
SoC	System on Chip
SPI	Serial Peripheral Interface
TCP/IP	Transmission Control Protocol/Internet Protocol
URL	Uniform Resource Locator
USB	Universal Serial Bus
WiFi	Wireless Fidelity

ÚVOD

Dnešní doba je plná moderních technologií a lidé jsou v neustálém spojení, jak mezi sebou, tak s děním okolo. Moderní člověk chce žít jednoduše pohodlný život, být informován a mít věci pod kontrolou.

Na tento trend reaguje tzv. čtvrtá průmyslová revoluce, která si mimo jiné klade za cíl digitalizaci a automatizaci činností vykonávaných lidmi. Nedílnou součástí této revoluce je také pojem IoT (internet věcí), díky jemuž se lze čím dál častěji setkávat s různými realizacemi těchto cílů. Příkladem může být chytré město, které automaticky řídí provoz na křižovatkách. Zároveň to může být také celý chytrý dům, který reguluje teplotu v domě, při setmění zatahuje rolety, rozsvěcí světla nebo při odchodu vypíná spotřebiče.

Hlavním cílem této práce je rozšíření schopností chytrého domu na jeho zahradu. Především návrhem a implementací systému pro zavlažování rostlin. Systém umožňuje zaznamenávání dat ze senzorů, ovládání aktivních prvků (ventilů, čerpadel) a spouštění zavlažování dle časového plánu či stanovených podmínek. Součástí systému je zároveň aplikační rozhraní pro navazující klientské aplikace.

Tvorbě webové aplikace, jakožto konzumenta zmíněného aplikačního rozhraní, je věnována samostatná kapitola. Cílem webové aplikace je poskytnutí uživatelského rozhraní pro vzdálené ovládání systému. Ke správě systému tak lze použít kterékoliv zařízení s moderním webovým prohlížečem. Tato aplikace zároveň slouží k přehlednému zobrazení historie měření senzorů.

Součástí práce je také představení existujících řešení pro řízení chytrých domů a krátké seznámení s technologiemi použitými u obou aplikací. Práce rovněž zahrnuje část dokumentace aplikačního rozhraní.

1 INTERNET VĚCÍ

Pojem internet věcí je seskupením chytrých zařízení, které jsou vzájemně propojené v síti. Jak už ze samotného názvu vyplývá, tak se zpravidla jedná o celosvětovou síť internetu, ale není to podmínkou. Komunikace zařízení v rámci IoT může být omezená i na lokální síť, ať už s využitím protokolu TCP/IP, nebo jiného způsobu komunikace například Bluetooth, RFID a podobných [1]. K propojení lze tedy využít téměř jakýkoliv způsob komunikace, který umožní jednoznačně odlišit dvě různá zařízení.

Tato zařízení se označují jako chytrá především díky svým vlastnostem. Hlavním důvodem k tomuto označení je, že jsou schopna poskytovat nějaká data či měnit svůj stav. Aby dané zařízení mohlo poskytovat data, tak musí obsahovat elektroniku, software nebo senzory ke snímání různých fyzikálních veličin [1]. Data jsou sdílená s ostatními chytrými zařízeními a ty pak na základě analýzy těchto dat konají další akce. Oproti běžnému používání počítače nejsou tyto informace tvořeny člověkem.

1.1 Historie

Přestože IoT zažívá svůj největší rozmach převážně v posledních deseti letech, tak lze nalézt jisté náznaky tohoto nastupujícího trendu již před více než půl stoletím. Za první záblesk IoT lze považovat dálnopis z roku 1964, který umožňoval lidem se sluchovým postižením používat telefon převedením zvukových tónů na elektrické signály, které byly na straně příjemce převedeny a vytištěny na papír. O něco sofistikovanějším byl způsob, kterým v roce 1982 výzkumníci z Univerzity Carnegieho-Mellonových zjišťovali stav nápojového automatu připojeného k internetu. O 8 let později na to bylo navázáno při konferenci Interopu vzdáleným ovládáním toasteru připojeného k internetu pomocí protokolu TCP/IP [2].

Důležitým milníkem pro IoT byl především rok 1999, kdy Kevin Ashton během své prezentace poprvé vyslovil slovní spojení: *„Internet of things“* [1]. Tento výraz se zalíbil i na univerzitě MIT, která tento pojem zaznamenala a přiřadila ho k technologii pro elektronické značkování věcí při distribuci produktů. V roce 2008 mezinárodní organizace vytvořily detailní specifikaci pojmu IoT. Zároveň v tomto roce odstartoval raketový nárůst zařízení připojených k internetu, přičemž velká část tohoto přírůstku byla tvořena právě chytrými zařízeními [2]. To bylo umožněno především rozšiřováním dostupnosti sítí, ale také pokračující miniaturizací procesorů, senzorů a dalších elektronických součástek.

1.2 Průmysl 4.0

Průmysl 4.0 je především iniciativou velkých technologických společností a státních institucí o rozšíření poptávky po moderních technologiích a jejich využívání. To vše nahrazením manuální lidské práce roboty, přidělením strojově čitelných identifikátorů pro materiály, automatickým řízením logistiky a automatickým předáváním dat mezi databázemi. Spolu s tím také souvisejícím efektivním využíváním materiálů a snižováním vlivů člověka na životní prostředí [3].

U výrobních společností by měla být hnacím motorem převážně vidina vyšších zisků, se zrychlením, zefektivněním a zkvalitněním výroby. V případě státních institucí se jedná o snižování nákladů, administrativní zátěže a obecné zkvalitnění občanského života. Vize čtvrté průmyslové revoluce ovšem nekončí pouze u výrobních společností a veřejného sektoru, ale snaží se zacílit i na jednotlivce. V praxi by se tak mělo být možné setkávat nejen s realizacemi plně automatizovaných továren, digitalizovaných zdravotních zařízení, chytrých měst, ale také chytrých domácností či vzájemně komunikujících dopravních prostředků. Propojení těchto částí znázorňuje Obrázek 1.

Obrázek 1 – Ilustrace Průmyslu 4.0 [3]

1.3 Chytré domy

Zvláštní kategorií spotřebitelského internetu věcí jsou bezpochyby chytré domácnosti. Ty mohou mít pod kontrolou hned několik dalších subsystémů. Může to být například chytrý termostat, který získává data ze senzorů teploty pro účely vytápění či ventilaci budovy.

Dále také systém propojených elektrospotřebičů jako jsou lednice, televize, pračky a další [3]. Dalším příkladem může být bezpečnostní systém zahrnující kamery, elektronické zámky, čidla pohybu nebo dokonce protipožární opatření.

Propojením všech těchto subsystému vzniká plnohodnotný chytrý dům, který mohou jeho obyvatelé ovládat například pomocí svého mobilu, ať už přímo v domě nebo z druhého konce světa. Nicméně možnost manuálního ovládání těchto subsystémů je spíše doplňkovou funkcí. Primárním důvodem, proč si člověk takový dům pořídí, je především jeho automatizovanost. Takový dům je sám schopen reagovat na změny v prostředí a učit se zvykům jeho obyvatel [3]. Příkladem může být výpadek elektřiny a automatické přepnutí na záložní zdroj s omezením energeticky náročných spotřebičů. Nebo také zapnutí vytápění těsně před příchodem obyvatel.

1.3.1 Apple HomeKit

Technologie od společnosti Apple, která umožňuje ovládání chytrých zařízení, jako jsou termostaty, světla, zámky a další. Jelikož jsou tato zařízení zpravidla vybavena bezdrátovou technologií Bluetooth nebo WiFi, tak není zapotřebí použití žádných dalších zařízení, ale postačí, že jsou zapojená do stejné lokální sítě. Ovládat je pak lze pomocí aplikace v zařízení se systémem iOS a to i pomocí hlasové asistentky Siri. Pro vzdálené ovládání je zapotřebí centrální bod, který ostatní chytrá zařízení zpřístupní z veřejné sítě. Centrálním bodem může být televize Apple TV, iPad nebo chytré reproduktory HomePod [4].

Nevýhodou tohoto řešení od společnosti Apple je velice limitovaný výběr kompatibilních chytrých zařízení, absence podpory češtiny a také vysoká pořizovací cena ve srovnání s konkurencí.

1.3.2 Google Smart Home

Své řešení pro ovládání chytrých domácností nabízí také společnost Google. Obdobně jako u Apple HomeKit i zde k ovládání domácnosti může sloužit chytrý reproduktor Google Home. Další možností je centrální bod Google Home Hub, který navíc disponuje dotykovou obrazovkou s uživatelským rozhraním. Obě zařízení existují v několika verzích a lze se k nim připojit bezdrátově pomocí WiFi. K nastavení zařízení slouží aplikace Home dostupná pro Android a iOS. Samozřejmostí je i podpora hlasového asistenta Google Assistant s využitím zabudovaného mikrofonu. Ten umožní ovládat chytrá zařízení v domě pomocí hlasových povelů [5].

Vybírat lze ze široké škály kompatibilních zařízení. U verze Home Mini je navíc velkou výhodou nízká pořizovací cena. Přestože společnost Google u jiných svých produktů podporu hlasové češtiny má, tak zde zatím chybí.

1.3.3 Komunikační protokoly

Chytrá zařízení využívají pro vzájemné předávání informací různých komunikačních protokolů. V oblasti IoT jsou hlavními požadavky na tyto protokoly rychlost, bezpečnost, spolehlivost, nízká cena a energetická nenáročnost. Vznikla tak řada komunikačních protokolů určených speciálně pro zařízení IoT.

Zigbee

Zigbee je bezdrátovým protokolem využívajícím převážně pásma mikrovln (2,4 GHz), ale na některých místech světa může z licenčních důvodů pracovat i na jiných frekvencích. V Evropě je to například 866 MHz a v Americe 915 MHz. V pásmu mikrovln je reálný dosah okolo 20 metrů s přenosovou rychlostí 250 Kbps. Protokol využívá tzv. smíšené topologie, ve které se zařízení napájená ze sítě chovají jako opakovače. Součástí přenosu dat je také informace o stavu zařízení [6].

Z-Wave

Bezdrátový protokol Z-Wave využívá ke komunikaci pásmo 900 MHz s dosahem do 40 metrů. Jedná se o spolehlivý stavový protokol s rychlostmi až 100 Kbps. Jednotlivé uzly jsou propojené v rámci smíšené topologie a jsou-li napájeny ze sítě, tak mohou sloužit i jako opakovače [6].

1.4 Vývojové platformy

Pořizovací náklady komerčních řešení IoT jsou pro jednotlivce mnohdy příliš velké. Tím se vytvořil prostor pro menší firmy a jejich vývojové desky. Výrobci takových desek ke svým produktům velmi často dodávají i specializovaný software usnadňující vývoj, nebo dokonce i celé operační systémy. To umožňuje i méně zkušeným uživatelům vytvářet svá vlastní řešení s výrazně nižšími náklady.

1.4.1 Mikrokontrolery

Nejlevnější vývojovou platformou jsou desky osazené AVR mikrokontrolery ATmega. Takové desky zpravidla disponují velmi malým výpočetním výkonem a omezenými paměťovými prostředky. Potřebuje-li uživatel například ukládat naměřená data, tak musí pomocí některé ze sběrnic připojit další speciální desku, která mu to umožní. Na desce pak běží jediný program, který je vykonáván ve smyčce. Výhodou je menší spotřeba energie.

Do kategorie mikrokontrolerů patří:

- Arduino,
- STM32,
- ESP8266,
- Teensy.

1.4.2 Minipočítače

Pro složitější projekty existuje kategorie minipočítačů. Ty už jsou víceméně plnohodnotnými stolními počítači, na kterých lze zároveň i vyvíjet další software. Výjimkou není přítomnost grafického procesoru, podpory velkokapacitních paměťových médií, ani ethernetového adaptéru. Spolu s operačním systémem na minipočítači může běžet hned několik dalších programů či služeb. Minipočítač tak může sloužit současně jako webový server, síťové úložiště, bezpečnostní systém nebo třeba domácí kino. Na rozdíl od mikrokontrolerů mají také širší podporu vyšších programovacích jazyků (Java, C++, Python, Ruby a další).

Mezi minipočítače se řadí:

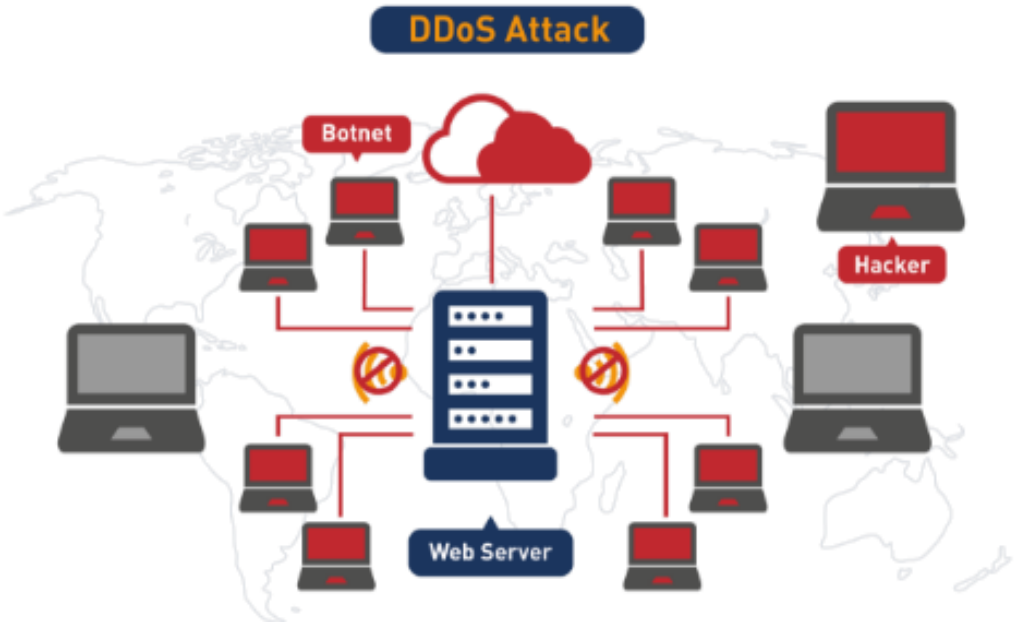
- Raspberry Pi,
- Banana Pi,
- ASUS Tinker Board,
- Orange Pi.

1.5 Bezpečnost

S navyšujícím se počtem zařízení připojených k internetu vzrůstá i zájem kybernetických útočníků o chytré věci. Uživatelé chytrých domácností velice často zabezpečení těchto systémů podceňují, protože se na první pohled může zdát, že rizika nejsou tak velká, jako u osobních počítačů. Nicméně opak je pravdou.

U chytrého termostatu může útočníkovi postačit i pouhá informace o časovém plánu vytápění. Z něj lze velice snadno vydedukovat, kdy se v domě nikdo nenachází. S přístupem k zabezpečovacímu systému se zase útočníkovi otevře celá řada možností, jak toho využít. Od otevření dveří až po špehování s pomocí bezpečnostních kamer [7].

Dalším případem zneužití chytrého zařízení je zařazení do tzv. botnetu[1]. Ten následně může útočníkovi sloužit k znepřístupnění libovolného webového serveru metodou DDoS[2] útoku [7]. Propojení počítačů v botnetu znázorňuje Obrázek 2.



Obrázek 2 – Ilustrace DDoS útoku [7]

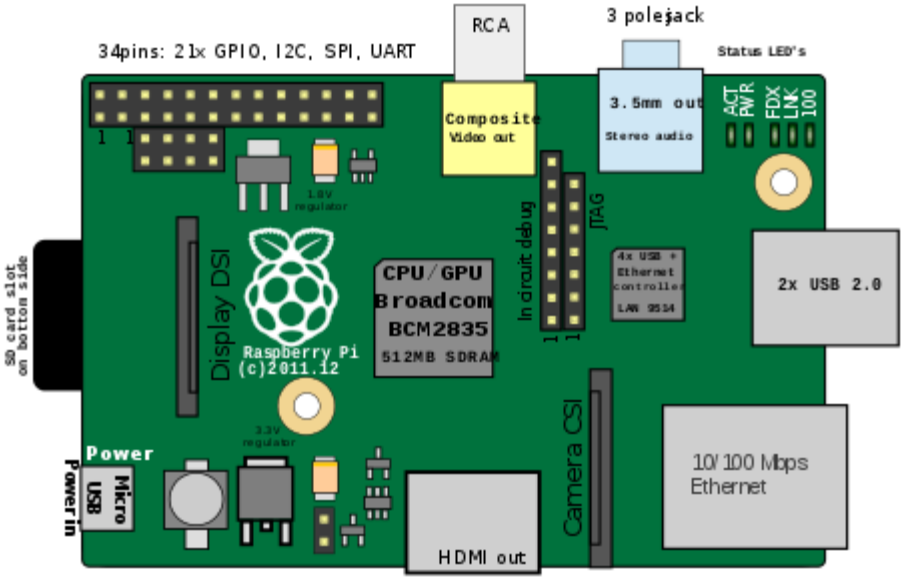
Test provedený Danielem Buentellem v roce 2017 dokázal fakt, že jsou výše uvedená rizika zcela reálná. Během pouhých 15 sekund se mu totiž podařilo převzít kontrolu nad chytrým termostatem firmy Nest [7].

Prevencí před těmito bezpečnostními riziky pak mohou být běžné zásady pro tvorbu silných hesel, pravidelné aktualizace firmware[3] a především funkční firewall[4] [7].

[1] Uskupení tisíců zařízení připojených k internetu, které slouží k realizaci kybernetických útoků.
[2] Typ kybernetického útoku, jehož cílem je znepřístupnění webového serveru či služby zasíláním požadavků z velkého množství zařízení během stejného okamžiku.
[3] Software nahraný výrobcem do elektronického zařízení při jeho vzniku.
[4] Zařízení či software oddělující lokální síť od veřejné.

2 RASPBERRY PI

Raspberry Pi je jednodeskovým minipočítačem od společnosti Raspberry Pi Foundation sídlící ve Velké Británii. Společnost vznikla v roce 2009 za účelem podpoření výuky informačních technologií a automatizace na školách. Docílit toho chtěla tak, že nabídne cenově dostupný výkonný počítač pro všechny. Prvním takovým počítačem určeným k prodeji byl Raspberry Pi Model B (Obrázek 3), který byl představen v roce 2012 a okamžitě zaznamenal velký úspěch. Během prvního půl roku se společnosti podařilo prodat přes půl milionu kusů. Velký úspěch to znamenalo i pro charitu, jelikož společnost vznikla s charitativním podtextem. Zisky z prodeje Raspberry Pi tak putují zpět do vývoje dalších minipočítačů a rozvoje vzdělávání. Díky tomuto faktu je Raspberry Pi Foundation podporována řadou organizací a dalších technologických společností, například Broadcom [8].



Obrázek 3 – Raspberry Pi Model B [8]

2.1 Modely

Od svého vzniku si Raspberry Pi upevnilo pozici na trhu a stalo se tak jedničkou na poli minipočítačů. Pokračující zájem o zařízení Raspberry Pi umožnil vznik dalších verzí vycházejících z původního počítače. Ty lze rozdělit do 4 hlavních kategorií dle určení:

- Model A – nízká cena,
- Model B – vysoký výkon,
- Zero – malé rozměry,
- Compute Module – průmyslové užití.

Srovnání vybraných modelů pro osobní použití poskytuje následující tabulka.

Tabulka 1 – Srovnání posledních modelů Raspberry Pi [9]

Název	SoC[5]	Takt	RAM	USB	Ethernet	WiFi	Bluetooth
Raspberry Pi 3 Model B	BCM2837A0/ B0	1,2 GHz	1 GB	4	10/100 Mbps	802.11n	4.1
Raspberry Pi 3 Model A+	BCM2837B0	1,4 GHz	512 MB	1	Ne	802.11ac/n	4.2
Raspberry Pi 3 Model B+	BCM2837B0	1,4 GHz	1 GB	4	300 Mbps	802.11ac/n	4.2
Raspberry Pi 4 Model B	BCM2711	1,5 GHz	1-4 GB	4	300 Mbps	802.11ac/n	5.0
Raspberry Pi Zero	BCM2835	1 GHz	512 MB	1	Ne	Ne	Ne
Raspberry Pi Zero W	BCM2835	1 GHz	512 MB	1	Ne	802.11n	4.1

2.2 Hardwarové vybavení

Jak již bylo nastíněno v předchozí kapitole, vybavení minipočítačů Raspberry Pi se liší dle modelu a jeho určení. U modelů řady A mohou velmi často chybět některé konektory a je nutné počítat i s nižším výkonem. Tyto nevýhody jsou však vykoupeny nižší cenou.

Modely typu B se naopak snaží poskytnout maximum uživatelského pohodlí. Pro připojení k internetu u nich nechybí ethernetový konektor pro připojení pomocí kabelu. S řešením situací, kdy pro uživatele není vhodné využít ethernetového připojení, naopak pomůže některé z bezdrátových řešení. U nových verzí z řady modelu B, lze pro bezdrátovou komunikaci využít technologie Bluetooth nebo WiFi [9]. Samozřejmostí je také vyšší výkon oproti ostatním modelům. Nejaktuálnější Raspberry Pi 4 Model B je nově nabízen rovnou v několika variantách s různou velikostí pamětí. Vybírat lze z verzí s 1 GB, 2 GB a 4 GB RAM. Další novinkou je podpora USB verze 3.

Na první pohled odlišnějšími modely jsou Zero a Compute Module. Oba modely mají o něco kompaktnější rozměry, což je činí vhodnějšími pro využití v IoT. Na modelu Zero se tato miniaturizace podepsala na velikosti desky, která je podstatně menší. Zmenšování se nevyhnulo ani konektorům, kterými tyto počítače běžně disponují. Pro napájení a data je zde k dispozici menší microUSB konektor a pro připojení k obrazovce miniHDMI. Jelikož jsou mnohdy zařízení v IoT napájeny i s pomocí baterie, tak je zapotřebí minimalizovat jejich spotřebu energie. V případě řady Zero je to řešeno snížením výkonu.

[5] Integrovaný čip, ve kterém je společně s procesorem i grafická jednotka a operační paměť.

Modely řady Compute Module si svůj výkon ponechávají a jsou určeny pro modulární systémy. Jejich deska má tvar DDR2-SODIMM. Díky tomu mají vyšší počet GPIO a konektory už nejsou vůbec zapotřebí.

Tabulka 2 zahrnuje srovnání běžné spotřeby u nejžádanějších modelů.

Tabulka 2 – Srovnání spotřeby vybraných modelů Raspberry Pi [9]

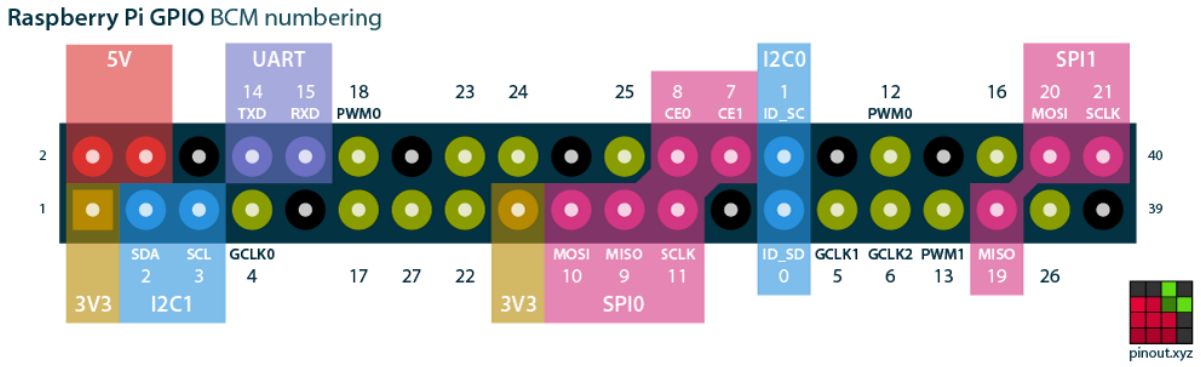
Název	**Běžná spotřeba**
Raspberry Pi 2 Model B	350 mA
Raspberry Pi 3 Model B	400 mA
Raspberry Pi 3 Model A+	350 mA
Raspberry Pi 3 Model B+	500 mA
Raspberry Pi 4 Model B	600 mA
Raspberry Pi Zero W/WH	150 mA
Raspberry Pi Zero	100 mA

Všechny modely mají obdobně řešené umístění procesoru, grafické jednotky i operační paměti. Ty jsou totiž umístěny ve společném pouzdru a navenek tak působí jako jediná komponenta. Další společné vlastnosti modelů (vyjma řady Compute Module) jsou:

- datové USB konektory,
- USB konektor pro napájení,
- digitální i analogový video výstup,
- analogový audio výstup,
- GPIO rozhraní,
- CSI port pro kamery,
- DSI port dotykové displeje [9].

2.2.1 GPIO rozhraní

Každý z minipočítačů Raspberry Pi disponuje univerzálním vstupně-výstupním rozhraním (GPIO). Toto rozhraní je na desce připájené v podobě hřebínku s jednotlivými piny. První verze modelu A a B měly na desce k dispozici pouze 26 GPIO pinů. Všechny následující verze již disponovaly 40 piny.



Obrázek 4 – GPIO rozhraní Raspberry Pi 2 B [10]

Jedno z možných rozmístění pinů znázorňuje Obrázek 4. Jednotlivé piny se od sebe liší svou funkcí a jsou značeny následovně:

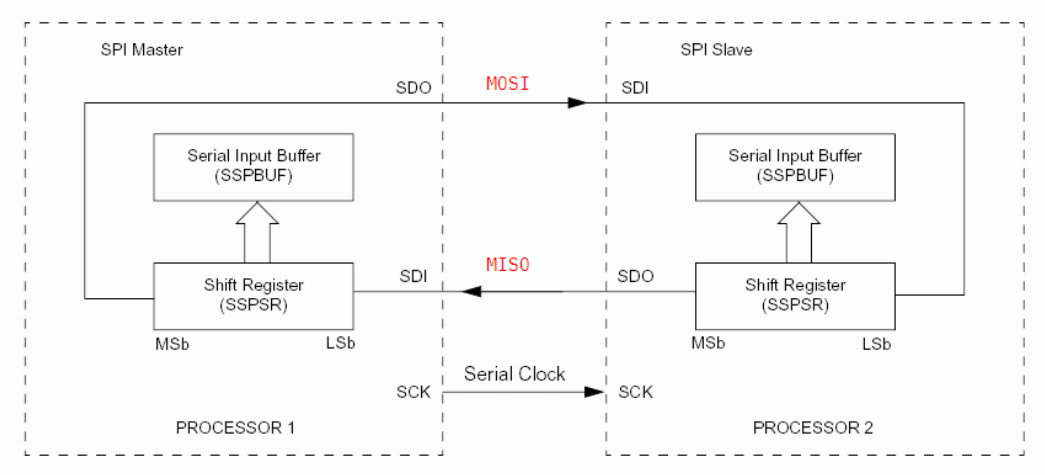
- 5V – napájení 5 V,
- 3V3 – napájení 3,3 V,
- GND – uzemnění 0 V,
- GPIO – digitální vstup/výstup,
- PWM – digitální piny s pulzně šířkovou modulací,
- SPI – sériové periferní rozhraní,
- I2C – interní integrovaná sériová datová sběrnice,
- UART – univerzální přijímač/vysílač.

2.2.2 Sériová sběrnice SPI

Pro připojení externích periférií jsou všechna zařízení Raspberry Pi vybavena sériovou sběrnicí SPI. S využitím této sběrnice lze k minipočítači připojit například hodiny reálného času (RTC), některé druhy pamětí, displeje nebo AD/DA převodníky [11]. U běžných modelů se vývody této sběrnice nacházejí na GPIO rozhraní. Jejich přesné umístění u vybraného modelu zobrazuje Obrázek 4.

Komunikace s využitím SPI sběrnice probíhá mezi tzv. uzly. Aby mohla komunikace vůbec probíhat, musí být propojeny alespoň 2 uzly, ale může jich být připojeno i více. Přičemž jeden z uzlů musí být řadičem sběrnice (tzv. master). Takový uzel pak generuje hodinový signál, který je rozveden do ostatních uzlů (tzv. slave) a slouží k synchronizaci vysílání či příjmu dat každého z uzlů. Pro předávání dat musí být jednotlivé uzly vybaveny posuvnými registry (SSPSR). Při každém hodinovém impulsu pak dojde k posunutí jednoho bitu v registru. Uzel tak odešle bit, který byl v registru uložen a zároveň přijme a uloží jiný bit z druhého uzlu. Uzel může obsahovat ještě i druhý tzv. záchytný registr (SSPBUF). Do záchytného registru, se zapíšou

data z posuvného registru až v momentě, kdy posuvný registr přijal celý bajt. Posuvný registr zde slouží jako jednoprvková fronta, čímž je vyloučena ztráta dat [11]. Proces komunikace dvou uzlů znázorňuje Obrázek 5.



Obrázek 5 – Propojení dvou uzlů pomocí SPI [11]

Propojení dvou uzlů se skládá z následujících vodičů:

- SCK (Serial Clock) – slouží pro synchronizaci uzlů hodinovým signálem,
- MOSI (Master Out, Slave In) – propojení datového výstupu řadiče se vstupem jiného uzlu,
- MISO (Master In, Slave Out) – datový vstup řadiče a výstup slave uzlu,
- CS (Chip Select) – pro výběr zařízení se kterým bude probíhat komunikace.

2.3 Softwarové vybavení

Stejně jako si nelze představit auto bez kol, tak si nelze představit ani počítač bez softwaru. V případě Raspberry Pi má navíc na software velký vliv skutečnost, že je společnost vyrábějící tento minipočítač také charitativním projektem. Proto je velká část vyvíjeného softwaru licencována jako open-source.

2.3.1 Operační systém

Aby byl vývoj pro tato zařízení co nejpřívětivější, tak bylo nutné vybrat operační systém s minimem omezení. Tím se stal otevřený unixový systém Linux a jak u něho bývá zvykem, tak i v tomto případě je šířen formou řady různých distribucí. Nicméně uživatel není omezen pouze na operační systém Linux. Využít může i systém Windows 10 IoT Core od společnosti Microsoft či Risc OS a další systémy určené pro architekturu ARM.

Mezi populární linuxové distribuce patří:

- Raspbian,
- Ubuntu Core,
- Ubuntu Mate,
- Linutop OS,
- LibreELEC,
- OSMC.

Raspbian

Oficiální operační systém pro všechny modely Raspberry Pi. Je součástí instalačního průvodce pro začátečníky. Po jeho instalaci je pro uživatele v systému připravena celá řada užitečných open-source programů a součástí jsou také předinstalovaná vývojová prostředí pro Python a Javu. Samotný systém je pak odvozen od Debianu.

2.3.2 WiringPi

WiringPi je knihovna pro ovládání GPIO rozhraní s využitím programovacích jazyků C, C++ nebo RTB[6]. Vznikla jako jakýsi most pro uživatele Arduina a jeho knihovny Wiring, kteří byli zvyklí na způsob ovládání GPIO pinů z Arduina. Díky své kvalitě se tato knihovna stala velice populární a nyní je dodávána společně s oficiálním systémem Raspbian [12]. Dočkala se i řady portů pro další programovací jazyky:

- Python (WiringPi-Python),
- Java (Pi4J).
- Ruby (WiringPi-Ruby),
- Perl (WiringPi-Perl),
- Node.js (WiringPi-Node),
- PHP (WiringPi-PHP).

[6] Return to Basics je programovací jazyk inspirovaný jazykem BASIC.

3 NÁVRH SYSTÉMU ZAVLAŽOVÁNÍ

Systém může být definován jako uskupení jednotlivých částí, které spolupracují za účelem poskytnutí nějaké formy výstupu na základě jednoho či více vstupů. Takové uskupení má své hranice, které oddělují systém od okolního prostředí. Interakce mezi systémem a okolním prostředím pak probíhá za pomocí signálů, které tyto hranice překračují [13]. Příkladem systému může být obyčejné rádio, jehož vstupem je příjem radiového signálu skrze anténu, čtení zvukového souboru z vyměnitelného média, ale klidně také stisk tlačítek uživatelem. Rádio tyto vstupy vnitřně zpracuje a následně produkuje výstup ve formě zobrazení informací na displej či přehráním hudby z reproduktorů. Systém je tak svým způsobem vždy abstrakcí nějakého komplexnějšího problému. U zmíněného rádia pro uživatele není důležité, jakým způsobem se demoduluje signál z antény, nebo jak funguje vnitřní zesilovač, ale pouze až přehrávaná hudba.

Vývoj informačního systému se skládá z následujících postupů:

0. business modelování – zachycení požadavků zadavatele a identifikace možných rizik,
1. stanovení požadavků – vytvoření modelu případů užití stanovením funkčních a nefunkčních požadavků, definováním aktérů a činností v systému,
2. analýza a návrh – vytvoření modelu analýzy na základě specifikovaných požadavků a modelu návrhu, například pomocí diagramu tříd,
3. implementace – část která se věnuje tvorbě výsledné aplikace, zahrnuje samotnou realizaci modelu návrhu,
4. testování – posledním krokem je ověření splnění požadavků, funkčnosti a spolehlivosti systému [14].

Kapitola návrhu systému se zabývá především fází zachycení požadavků, ale také výběrem vhodných hardwarových komponent a softwarových technologií. Samotné implementaci softwarové části jsou posléze věnovány samostatné kapitoly „*Backend aplikace systému zavlažování*“ a „*Frontend aplikace systému zavlažování*“.

3.1 Požadované vlastnosti

Cílem práce bylo vytvořit systém pro automatické zavlažování rostlin. Systém by měl umožňovat zejména:

- správu senzorů půdní vlhkosti (vstupů),
- správu akčních členů (výstupů, vodní čerpadla nebo ventily),

- automatické provádění úloh jejichž spouštění se řídí časovým plánem, nebo splněním stanovených podmínek na základě dat ze senzorů,
- zaznamenávání dat ze senzorů do databáze, tak aby bylo možné se k nim později vracet a přehledně je vyobrazovat do grafů,
- zachování svého stavu i po případném výpadku napájení či pádu aplikace,
- ovládání skrze aplikační rozhraní,
- přístup pouze po zadání platných přihlašovacích údajů,
- vzdálené ovládání systému.

Posouzením požadovaných vlastností bylo možné specifikovat základní obor funkčních požadavků pro softwarovou část. Jednotlivé požadavky jsou uvedeny v následující tabulce.

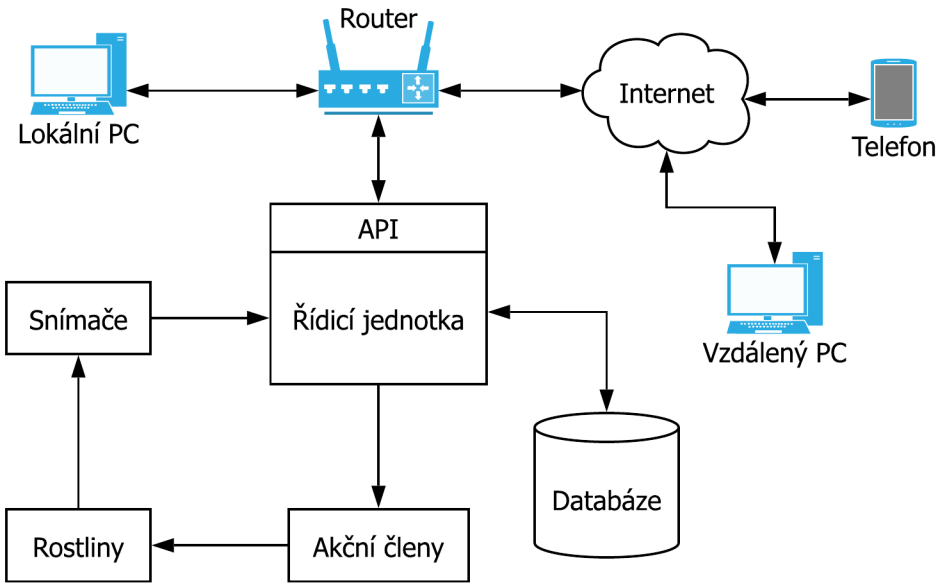
Tabulka 3 – Funkční požadavky

ID[7]	Specifikace
FP01	Uživatel bude přihlášen do systému.
FP02	Uživatel bude spravovat senzory.
FP03	Uživatel bude spravovat výstupy.
FP04	Uživatel bude spravovat automatické úlohy.
FP05	Uživatel bude spravovat své přihlašovací údaje.
FP06	Systém bude evidovat senzory.
FP07	Systém bude evidovat výstupy.
FP08	Systém bude evidovat automatické úlohy.
FP09	Systém bude spouštět automatické úlohy.
FP10	Systém bude evidovat uživatele.
FP11	Systém bude zaznamenávat hodnoty senzorů.
FP12	Systém bude kontrolovat platnost přihlášení uživatele.
FP13	Systém bude poskytovat informace o stavu senzorů.
FP14	Systém bude poskytovat informace o stavu výstupů.
FP15	Systém bude poskytovat informace o stavu provádění automatických úloh.
FP16	Systém bude poskytovat historii stavu senzorů.
FP17	Systém bude pravidelně mazat staré záznamy měření senzorů.
FP18	Uživatel bude spravovat interval zaznamenávání stavu senzorů.
FP19	Uživatel bude spravovat dobu uchovávání záznamů ze senzorů.
FP20	Systém bude schopný načíst svůj stav z předchozího spuštění.

[7] Pro identifikaci požadavků byl zvolen systém identifikátoru složeného ze zkratky „FP“ (funkční požadavek) a číselného id požadavku.

3.2 Blokové schéma

Na základě specifikovaných požadavků bylo dále možné sestavit obecný diagram návrhu systému. Obrázek 6 znázorňuje interakce systému s okolním prostředím a jeho aktéry. V diagramu jsou tak zahrnuty aspekty hardwarové, ale i softwarové části.



Obrázek 6 – Blokové schéma funkce systému

3.3 Hardwarové komponenty

Při návrhu systému byl brán ohled na maximální univerzálnost. Uživatel je tak při výběru hardwarových komponent omezen pouze u řídící jednotky Raspberry Pi, A/D převodníku MCP 3008 a bezdrátového relé Sonoff. Použití těchto komponent úzce souvisí s implementací softwarové části, a tudíž je nelze nahradit jinými alternativami. Výběr konkrétního snímače, ventilu či čerpadla už ale závisí pouze na doporučeních z následujících kapitol a preferencích uživatele.

3.3.1 Řídicí jednotka

Při určení řídicí jednotky se do užšího výběru dostaly dvě velmi oblíbené vývojové platformy. První zvolenou platformou byly mikrokontrolery Arduino a druhou minipočítače Raspberry Pi.

V případě Arduina se jako největší výhoda jevila možnost přímého čtení analogových hodnot ze senzorů. Dalším nezpochybnitelným kladem byla nízká cena, která se v České republice pohybuje u nejnižšího modelu okolo 100 Kč. Nicméně při zvažování požadavku na připojení k síti se naskytl první problém. Arduino totiž běžně nedisponuje adaptéry pro připojení k internetu. Řešením může být dokoupení některého z modulů, který připojení umožní, ale to

zároveň znamená i navýšení finančních nákladů. Dalšími zásadními překážkami jsou omezená programová paměť a absence současného běhu více programů. Tyto fakty by značně ztížily realizaci ostatních požadavků, protože se zcela evidentně nebude jednat o vývoj triviální aplikace o velikosti v řádech kilobajtů.

Pro realizaci systému byla nakonec zvolena platforma Raspberry Pi, která ve srovnání s Arduinem splňuje požadavek na vyšší výkon a konektivitu. Naneštěstí i tato volba s sebou nese jednu nevýhodu. Raspberry Pi totiž nedisponuje analogovými vstupy, nicméně tento problém lze vyřešit pořízením levného analogově-digitálního převodníku (ADC).

Posledním krokem byl výběr modelu Raspberry Pi. Jako nejvhodnější řešení se jevil model Raspberry Pi Zero W, který disponuje vestavěným WiFi adaptérem a lze ho sehnat zhruba za 300 Kč. Avšak nakonec byl pro vývoj zvolen starší model Raspberry Pi 2 B, který poskytuje vyšší výkon a do budoucna ho lze snadno nahradit novějšími modely.

Raspberry Pi 2 Model B

Přestože se jedná o starší model z roku 2015, tak svým výkonem nijak výrazně nezaostává. Zásadním je pak fakt, že novější modely disponují stejným či zpětně kompatibilním GPIO rozhraním[8]. Specifikace tohoto modelu jsou uvedeny v tabulce 4.

Tabulka 4 – Specifikace Raspberry Pi 2 Model B [9]

Název	Raspberry Pi 2 Model B
SoC	BCM2836/7
Takt	900 MHz
RAM	1 GB
USB	4 porty
Ethernet	10/100 Mbps

Následující části práce se věnují především vývoji pro tento model.

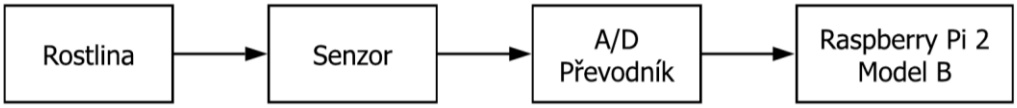
3.3.2 Snímače

Tato část se zabývá měřením fyzikálních veličin ve zkoumaném prostředí. Pro účely zavlažování je nejdůležitější veličinou půdní vlhkost. Tu lze změřit s využitím senzoru, jehož výstupem je analogový signál ve formě jedné z elektrických veličin. Řídicí jednotky s analogovým vstupem už si s takovým signálem dokáží poradit, protože mají zabudovaný A/D převodník. Nicméně Raspberry Pi analogovými vstupy nedisponuje, a proto je zapotřebí

[8] Při tvorbě této práce byl nejnovějším modelem Raspberry Pi 4 Model B.

takový převodník pořídit samostatně. Analogový senzor je posléze nutné připojit na vstup převodníku a výstup převodníku dále na SPI rozhraní řídicí jednotky.

Výběrem vhodného senzoru a převodníku, se podrobněji zabývají následující dvě podkapitoly „Senzor půdní vlhkosti“ a „A/D Převodník“. Po nich následuje podkapitola, která detailně řeší jejich vzájemné propojení. Blokový diagram snímání veličin je uveden na obrázku 7.



Obrázek 7 – Diagram snímání fyzikální veličiny

Senzor půdní vlhkosti

Měření půdní vlhkosti lze provádět celou řadou metod, přičemž každá z metod má své výhody a nevýhody. Metody lze dělit do kategorií hned několika způsoby. Prvním způsobem může být rozdělení na destruktivní a nedestruktivní metody. V případě destruktivní metody je zapotřebí odebrat půdní vzorek z místa měření [15]. Tuto metodu bylo tedy možné rychle zavrhnout, protože nesplňovala požadavek jednoduchého automatického měření.

Dále lze metody měření dělit na:

- přímé – měření pomocí přímých metod se zaměřuje přímo na fyzikální veličinu jejíž hodnota je zjišťována, v případě půdní vlhkosti je to množství vody v půdě, nevýhodou senzorů využívajících přímou metodu měření je nutnost časté údržby,
- nepřímé – tyto metody se zaměřují na jevy, které změny hodnot zkoumané veličiny doprovází, v případě půdy to může být elektrický odpor, jehož změny se projeví při průchodu proudu mezi elektrodami [15].

Pro využití s vývojovými platformami, se lze nejčastěji setkat se dvěma druhy senzorů využívajících nepřímé metody měření:

- odporové – využívají úměry zvyšování elektrické vodivosti, se zvyšováním vlhkosti půdy, zcela zásadní nevýhodou je náchylnost elektrod ke korozi při souvislém měření,
- kapacitní – při vystavení půdy elektrickému poli lze měřit kapacitu mezi elektrodami, jejíž velikost je rovněž závislá na vlhkosti půdy, na rozdíl od odporové metody lze měření provádět neustále, protože zde nedochází k tak silným chemickým reakcím [15].

Na základě těchto znalostí bylo možné stanovit požadavky na senzor vlhkosti.

Snímač vlhkosti musí splňovat:

- kapacitní způsob měření vlhkosti,
- výstupní napětí 3,3 V,
- přijatelnou teplotní závislost,
- odolnost vůči korozi,
- přibližně lineární křivku v rozsahu měření.

Vhodným kandidátem pro účely navrhovaného systému je například senzor s označením SEN0193[9] od společnosti DFROBOT.

A/D Převodník MCP3008

Raspberry Pi disponuje pouze digitálními vstupy, a proto k němu lze připojit pouze senzory s digitálním výstupem. Tímto způsobem lze na vstupu rozeznat pouze 2 rozdílné hodnoty, nula a jedna. Při takovém rozlišení by grafické znázornění naměřených hodnot nebylo příliš zajímavé. Takové měření by bylo zároveň zcela nedostačujícím pro spouštění zavlažování při poklesu půdní vlhkosti pod určitou úroveň. Proto bylo nutné vybrat vhodný převodník, který analogový signál převede do digitální podoby, se kterou už Raspberry Pi pracovat umí.

Pro převod analogového signálu byl vybrán desetibitový převodník MCP3008 od společnosti Microchip. Na jeho vstup lze připojit až 8 různých senzorů s analogovým výstupem. Spojitý signál z každého senzoru je pak se stanovenou periodou vzorkován, přičemž periodu vzorkování lze měnit velikostí napětí přivedeného na vstup V_{DD}. Jednotlivým vzorkům signálu, které mají v této podobě velkou přesnost, musí být přidělena hodnota z diskrétní množiny, kterou je počítač schopen rozlišit. Tento krok se nazývá kvantování a jeho množina hodnot se skládá z tzv. kvantizačních kroků. Ty jsou v případě MCP 3008 určeny jeho desetibitovým rozlišením a počet těchto kroků je 2^{10}, respektive 1024 rozdílných hodnot na výstupu.

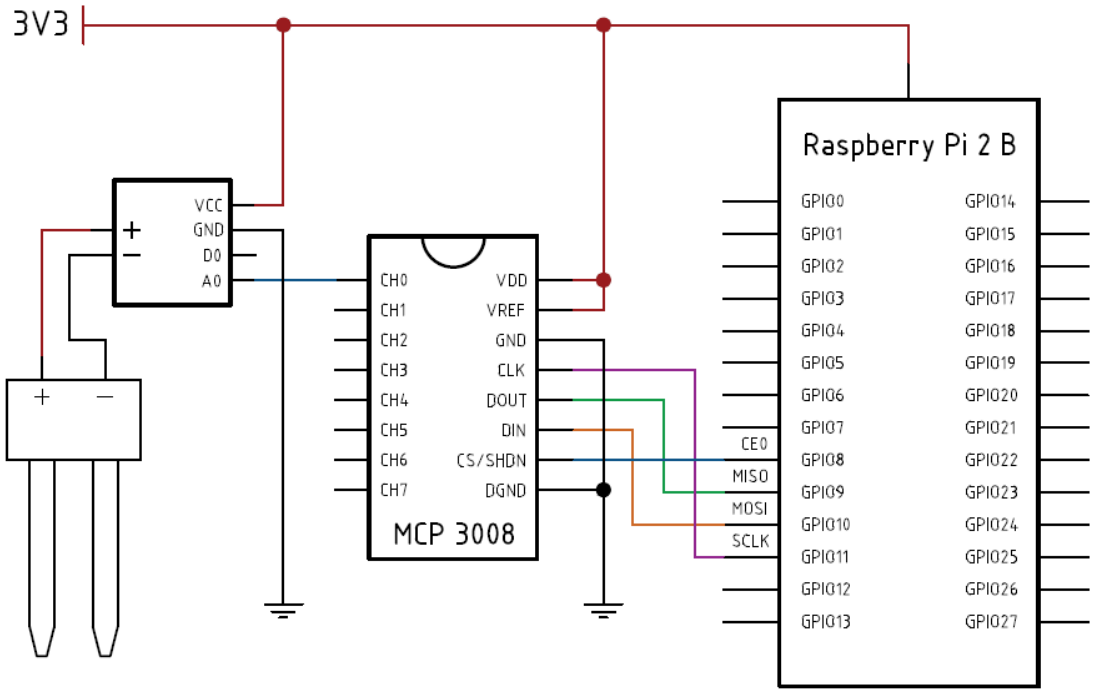
K Raspberry Pi lze převodník připojit pomocí SPI rozhraní[10]. Při tomto způsobu komunikace je řídicí jednotka v roli master a převodník pracuje jako slave. Detailní princip funkce je popsán v kapitole „Sériová sběrnice SPI“.

Schéma zapojení

Na obrázku 8 je znázorněn způsob připojení senzoru na jeden ze vstupních kanálů A/D převodníku. Pro připojení dalších senzorů lze využít obdobného zapojení, pouze s tím rozdílem, že bude využit jiný volný kanál převodníku. Dále je ve schématu znázorněno propojení SPI komunikace Raspberry Pi a převodníku. U převodníku si lze současně povšimnout zapojení

[9] https://arduino-shop.cz/arduino/4875-analogovy-snimac-vlhkosti-pudy-v1.2.html
[10] Kompletní specifikace MCP 3008 v dokumentaci na http://ww1.microchip.com/downloads/en/DeviceDoc/21295d.pdf

referenčního napětí (V_{REF}) a napětí pro vzorkování (V_{DD}). U části Raspberry Pi je nutno podotknout, že se nejedná o schéma GPIO rozhraní, nýbrž o výstupy ze SoC. Pro připojení k GPIO rozhraní lze využít Obrázek 4, přičemž číslování pinů je u tohoto obrázku stejné jako v následujícím schématu.



Obrázek 8 – Schéma zapojení snímače

3.3.3 Akční členy

Jedná se o část systému, která se stará o přísun vody k rostlinám. Protože jsou zařízení pracující v této části obvykle i náročnější na spotřebu energie, než je Raspberry Pi schopné dodat ze svých výstupních pinů, tak je nutné počítat s dalším zdrojem energie. Jednotlivé zdroje energie je dále nutné od sebe vzájemně elektricky oddělit, tak aby nemohlo dojít k poškození řídící jednotky. Pro tento účel je součástí návrhu systému bezdrátové relé.

Bezdrátové relé Sonoff SV

Sonoff SV[11] je elektronický obvod, jehož hlavními částmi jsou elektromagnetické relé a mikrokontroler ESP8266.

Relé se skládá z cívky s feromagnetickým jádrem a spínacích kontaktů. Při průchodu proudu cívkou dochází k indukci jejího jádra a stává se z ní magnet. Tento magnet posléze přitáhne kotvu, která mechanicky spojí, nebo rozpojí spínací kontakty. Na první kontakt je přivedeno

[11] https://www.itead.cc/smart-home/sonoff-sv.html

napětí ze zdroje a druhý kontakt je vyveden na čerpadlo či elektromagnetický ventil. Sepnutí nebo rozepnutí kontaktů následně zapříčiní, že je spotřebič napájen, nebo nikoliv.

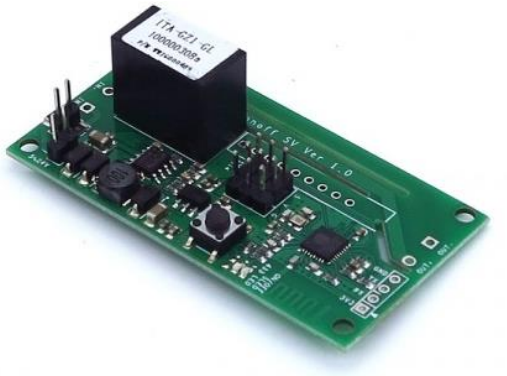
Výstupní kontakty relé lze zapojit ve dvou různých režimech:

- spínací (NO) – spínací kontakt je rozpojen, dokud cívkou neprochází proud,
- rozpínací (NC) – pokud cívkou neprochází proud, tak je spínací kontakt spojen.

Pro zavlažovací systém byla zvolena varianta, kdy na výstupu z Raspberry Pi, při aktivním zavlažování, bude logická hodnota 1. U většiny běžných čerpadel či elektromagnetických ventilů to znamená, že budou připojeny ve spínacím (NO) režimu.

Dalším důležitým prvkem celého obvodu je mikrokontroler ESP8266. Ten zde zajišťuje připojení k bezdrátové síti WiFi a zároveň umožňuje spárování s hlasovými asistenty Google Assistant či Amazon Alexa. Konfiguraci bezdrátového relé lze provést skrze výrobcem dodávanou mobilní aplikaci eWeLink[12].

Součástí plošného spoje jsou také vývody[13] některých pinů mikrokontroleru. Pro systém zavlažovaní je nejdůležitějším pin GPIO14, který je možné spojit s některým z digitálních výstupů Raspberry Pi. Hřebínek s vývody mikrokontroleru se nachází zhruba uprostřed destičky na obrázku 9.



Obrázek 9 – Ukázka bezdrátového relé Sonoff SV [16]

Čerpadlo

Čerpadlo je jednou z nejdůležitějších částí zavlažovacího systému. Toto zařízení umožňuje přepravu vody potrubím ze zdroje až k rostlinám. Pro navrhovaný systém zavlažování se počítá s kombinací zdroje vody v podobě nádrže a ponorného čerpadla.

[12] https://www.ewelink.cc/en/
[13] https://www.itead.cc/wiki/Sonoff_SV#Pin_Map

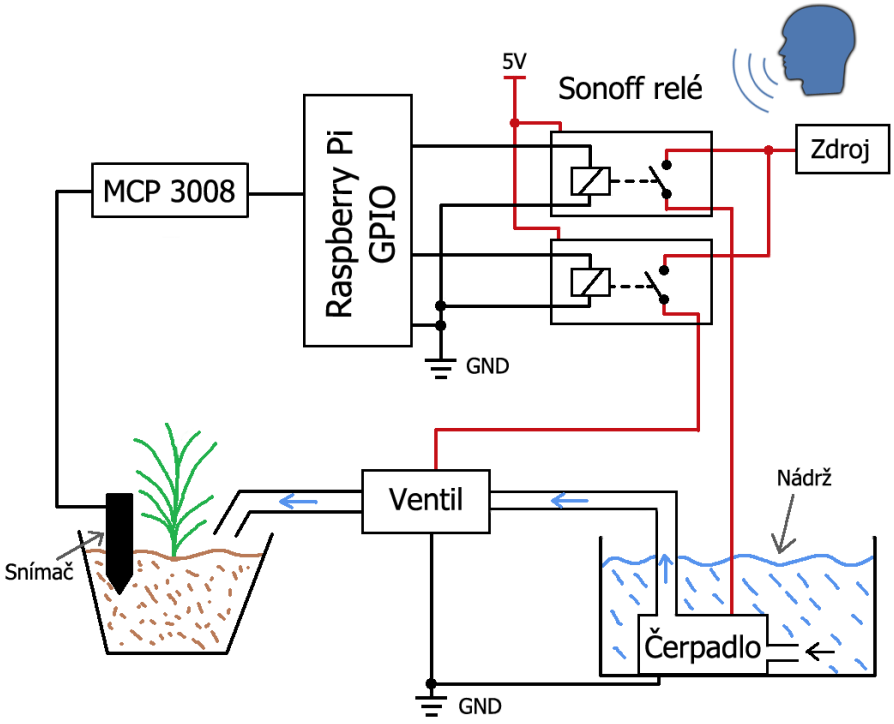
Elektromagnetický ventil

V kombinaci s čerpadlem či jiným přívodem vody lze také použít elektromagnetický ventil. Ten se svou funkcí nijak neliší od běžného vodovodního ventilu. Rozdíl spočívá v jeho ovládání, obdobně jako využívá relé magnetismu cívky, tak elektromagnetický ventil využívá magnetismu k uzavírání a otevírání průchodnosti potrubí. Tím lze jednoduše zajistit, zdali k rostlině bude proudit voda, nebo nikoliv.

Příklad zapojení

Zavlažovací část lze řešit různými způsoby. Uživatel může k zavlažování využít například skupiny čerpadel, ve které bude mít každá rostlina své vlastní čerpadlo. Další variantou může být hlavní společné čerpadlo, přičemž ovládání přísunu vody k rostlinám pak mohou řídit jednotlivé elektromagnetické ventily.

Na následující ilustraci hardwarové části systému (Obrázek 10) je znázorněn příklad, kdy je připojeno jedno hlavní čerpadlo a jeden či více elektromagnetických ventilů.



Obrázek 10 – Příklad zapojení hardwarové části

3.4 Softwarové technologie

Při výběru technologií pro softwarovou část bylo nutné zaměřit se především na plnění stanovených požadavků. Spolu s tím měly vybrané technologie umožňovat jejich snadné propojení a nasazení, a to i na platformu s omezenějšími prostředky, kterou Raspberry Pi bezpochyby je.

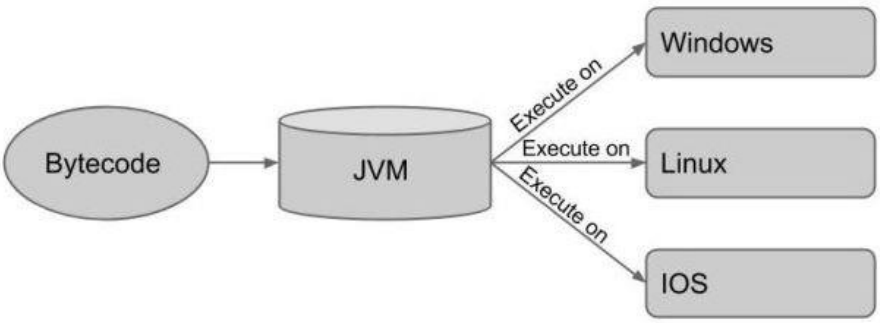
Výběr programovacího jazyka byl nejdůležitější částí návrhu aplikace pro systém zavlažování. S ohledem ke komplexnosti celého systému se ukázalo, že vhodným bude některý z objektově orientovaných (OOP) programovacích jazyků. Seznam možných kandidátů byl díky zvolené platformě poměrně široký. Na Raspberry Pi s unixovým systémem Linux lze totiž provozovat většinu z moderních programovacích jazyků. Nicméně konečná volba padla na velmi rozšířený a uživatelsky přívětivý programovací jazyk Java.

Dále bylo nutné určit technologii pro vývoj klientské aplikace. V oblasti IoT jsou velice oblíbené webové aplikace, protože uživateli dávají svobodu při volbě zařízení, ze kterého bude systém ovládán. Autor práce se rozhodl tento potenciál naplno využít, a navíc ho posunout o něco dále. Toho bylo možné docílit výběrem moderního javascriptového frameworku Angular, který se zaměřuje na tvorbu dynamických webových stránek.

3.4.1 Java

Java je poměrně bezpečným programovacím jazykem, který plně využívá principů objektově orientovaných jazyků. Za pomocí abstrakce a zapouzdření umožňuje třídám ukrývat informace před svým okolím. To dovoluje znovupoužívání již vytvořených částí kódu, čímž se celý proces vývoje značně urychluje. Díky tomu má programátor k dispozici celou řadu knihoven s bohatým API. Orientaci v těchto knihovnách navíc velmi usnadňuje systém dokumentačních komentářů. Objektově orientovaný přístup zároveň dovoluje rozložení komplexních projektů na elementární části, které provádějí atomické operace [16].

Java běží ve svém vlastním virtuálním prostředím, které mu poskytuje virtuální stroj Java Virtual Machine (JVM). Ten navíc disponuje tzv. *garbage collectorem,* který se stará o správu přidělené paměti. Další výhodou je podpora vícevláknových aplikací a také jeho multiplatformnost. Napsaný kód je pak nutné zkompilovat na tzv. *bytecode,* který JVM dále převede na strojový kód a spustí [16]. Tento proces znázorňuje Obrázek 11.



Obrázek 11 – Znázornění funkce JVM [16]

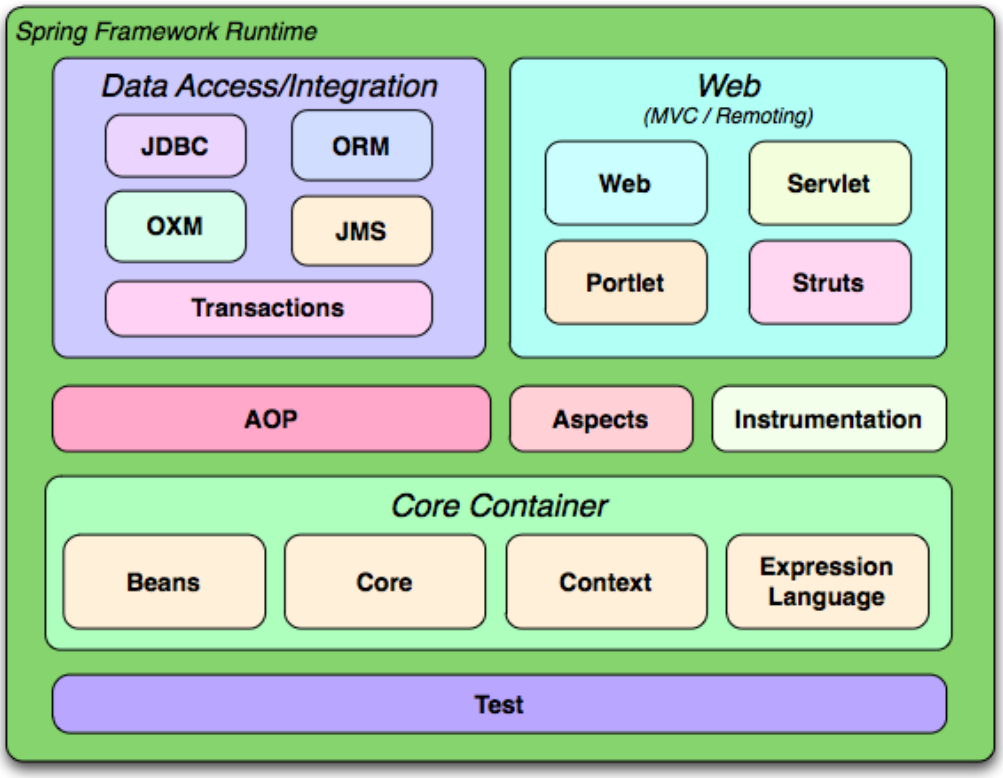
3.4.2 Spring Framework

Spring Framework (dále jen Spring) je otevřeným aplikačním rámcem, který usnadňuje vývoj podnikových aplikací. Jeho vrstvená architektura je tvořena moduly, které na sobě nejsou nijak závislé. Jednou z nejdůležitějších vlastností Springu je tzv. Inversion of Control (IoC), neboli obrácené řízení. S jeho pomocí je možné, že jednotlivé třídy mezi sebou nemusejí mít pevné vazby. Takovéto třídy se nazývají Beans a o jejich vytvoření se stará samotný framework. Pokud některá z takto vytvořených tříd potřebuje použít funkce jiné třídy, tak ji framework pomocí tzv. Dependecy Injection (vkládání závislostí) předá referenci na požadovanou třídu. Vložení závislostí lze provést několika metodami, implementováním rozhraní, parametrem konstruktoru a *set* metodou. Nejsnadnějším způsobem je použití anotace *@Autowired,* kterou lze mimo konstruktor a *set* metodu použít i na atributy třídy. Spolu s konfigurací aplikace tvoří IoC modul jádra frameworku [18]. Strukturu jednotlivých modulů Spring Frameworku pak znázorňuje Obrázek 12.

Další často používané moduly se zaměřují na:

- perzistenci – za pomoci Java Database Connectivity (JDBC), Spring poskytuje abstrakci pro správu zdrojů a zachytávání výjimek při práci s různými databázemi, zároveň integruje řadu ORM (objektově relačních mapovacích) frameworků, které usnadňují uchovávání dat po ukončení aplikace,
- web – modul stavějící na návrhovém vzoru MVC (Model View Controller), je určen pro webové aplikace, přičemž umožňuje komunikaci klienta a serveru za pomocí webových soketů, dále dovoluje tvorbu dynamických webových stránek či REST rozhraní [19],
- komunikaci – poskytuje mechanismy pro tvorbu a zasílání zpráv,

- testování – pro provádění testů nechybí podpora JUnit či TestNG, zároveň usnadňuje načítání kontextu a vytváření testovacích objektů (tzv. Mock), které umožňují provádět testování v izolovaném prostředí [20].



Obrázek 12 – Struktura aplikačního rámce Spring Framework [20]

3.4.3 Knihovna Pi4J

Knihovna Pi4J je řešením pro ovládání GPIO rozhraní při využití jazyka Java. Jedná se o port knihovny WiringPi, která je určena pro aplikace psané v programovacím jazyku C či C++. Její API lze využít jak pro ovládání GPIO vstupů či výstupů, tak pro složitější komunikaci v podobě některé ze sběrnic. Součástí knihovny jsou i speciální třídy připravené pro práci s vybranými zařízeními, které lze přes GPIO připojit. Knihovna dále pracuje s konstrukcemi tříd často využívaných v jazyce Java, přičemž mezi ně patří například Observer, Listener, Trigger či Monitor.

Jedním z vybraných zařízení, která mají v knihovně svá zastoupení, je také A/D převodník MCP3008. Ten má v knihovně svou vlastní třídu MCP3008GpioProvider, která umožňuje přístup k funkcím převodníku. U této třídy může programátor využít konstrukce monitoru pro pravidelné čtení hodnot z připojených analogových senzorů. Interval načítání hodnot lze navíc pohodlně nastavit metodou *setMonitorInterval*. Metodou *setEventThreshold* je dále možné nastavit, jak velká musí být změna čtené hodnoty, aby vyvolala událost převodníku. Zpracování

této události je pak možné definovat pomocí listeneru s tzv. Handle metodou. Ta blíže specifikuje, co se má v programu dále vykonat [21].

Při používání této knihovny je zapotřebí dávat pozor na číslování pinů. Knihovna totiž nevyužívá číslování pinů ze SoC, ale přímo z knihovny WiringPi. Značení pinů se tak ve většině případů liší. Schéma značení pro Raspberry Pi 2 B obsahuje příloha A.

3.4.4 Quartz Enterprise Job Scheduler

Open-source knihovna pro plánování úloh, kterou je možné využít jak pro jednoduché aplikace, tak i pro komplexnější projekty. Pro naplánování úlohy je možné využít *CRON* formátu známého z unixových systémů. S jeho využitím lze provádění úloh naplánovat opakovaně například každé pondělí, jednou za dva dny nebo třeba první den v měsíci. S tímto formátem dále pracuje třída *Trigger*, která úlohy aktivuje. Plánovač pak může spouštět jakoukoliv třídu, která bude implementovat rozhraní *Job* [22]. Toto rozhraní definuje jedinou metodu s názvem *executeInternal*, která se při aktivaci plánovačem vykoná.

Knihovna je zároveň připravena pro uchování dat v databázi, k tomuto účelu využívá JDBCJobStore. Pro spouštění úloh využívá knihovna tzv. *pool*, který se skládá ze stanoveného počtu vláken připravených pro spuštění úkolu. Při aktivaci úkolu plánovačem je úloze přiděleno jedno z vláken, které danou úlohu vykoná. Pokud jsou všechna vlákna v daný okamžik využívána, tak je poslední aktivovaná úloha zařazena do fronty. Tam čeká do doby, dokud se některé z vláken neuvolní [22].

Spring Framework tuto knihovnu dokonce integroval, a proto se v něm lze setkat například s třídou QuartzJobBean, která vychází z již zmíněné třídy *Job*.

3.4.5 MariaDB

Pro uchovávání dat byla zvolena odnož databáze MySQL s názvem MariaDB, která je dostupná i pod bezplatnou licencí. MariaDB jakožto relační databáze, se vyznačuje strukturou dat ve formě tabulek. Každý sloupec má přidělený datový typ a jednoznačný název. Na jednotlivých řádcích jsou pak zaznamenávány data, které má databáze uchovávat. Takových tabulek může být v databázi více, a dokonce mezi sebou mohou mít různé vazby. Tím lze zamezit duplikaci dat a zajistit jejich znovupoužitelnost. Pro práci s databází slouží jazyk SQL a operace v ní jsou prováděny atomicky, díky čemuž jsou eliminovány nekonzistentní stavy dat. Pokud v jeden okamžik vznikne více požadavků nad stejnými daty, tak jsou vykonávány popořadě. Data jsou v rámci databáze ukládána na souborový systém a tím je zaručena jejich dostupnost i při opětovném spuštění aplikace [23].

U navrhovaného systému zavlažování je možné databázi provozovat jak přímo na Raspberry Pi, tak na jakémkoliv jiném stroji. Ke které databázi se zavlažovací systém připojí, tak záleží pouze na uživateli a jeho konfiguraci.

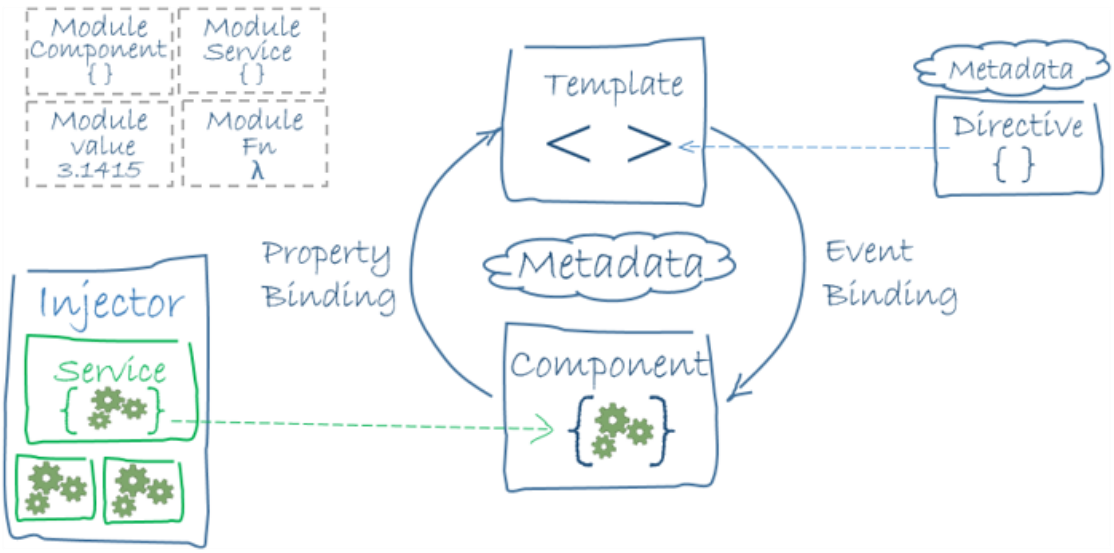
3.4.6 Angular

Angular je novou verzí původního AngularJS frameworku pro tvorbu dynamických webů. Na AngularJS probíhal vývoj za pomoci JavaScriptu. Přestože nový Angular ke svému běhu JavaScript využívá, tak je samotný kód doporučeno psát v TypeScriptu. Ten se od JavaScriptu liší svým zaměřením na aspekty objektově orientovaného programování, a to především podporou statického typování. Při kompilaci kódu psaného v TypeScriptu dochází ke kontrole typování a převodu do JavaScriptu.

Aplikace vytvořená pomocí Angularu je složením různých modulů (kontejnerů). Součástí modulu je komponenta, kterou může být například příspěvek, formulář, menu nebo třeba hlavička stránky. Hlavní částí každé komponenty je třída, na kterou se pomocí direktiv váže rozložení a vzhled stránky. Vzhled komponenty lze definovat pomocí šablony v HTML a CSS. Výsledný vzhled stránky je jakýmsi stromem složeným z jednotlivých komponent, přičemž hlavní komponentou, která je předkem všech ostatních komponent, je tzv. *root* komponenta [24].

Další částí modulu mohou být služby, které se starají o data a logiku aplikace. Pro doplnění dat do HTML šablony slouží tzv. *data binding* (navázání dat), který se provede ještě před vykreslením šablony. Datová vazba může být i oboustranná, a proto se například vyplnění políčka formuláře ihned projeví i v datech. Pro navigaci po stránce slouží modul Router, jehož služba spravuje URL adresy a propojuje je s komponentami [24].

Strukturu a vazby mezi jednotlivými komponentami znázorňuje Obrázek 13.



Obrázek 13 – Struktura frameworku Angular

4 BACKEND APLIKACE SYSTÉMU ZAVLAŽOVÁNÍ

Backend aplikace zavlažovacího systému je prostředníkem mezi reálným a digitálním světem. Hlavními objekty na této hranici jsou senzory a výstupy, které aplikace vnitřně spravuje. Primárním účelem je ovšem shromažďování dat o půdní vlhkosti z těchto senzorů. Na základě takto získaných dat je pak aplikace dále schopna provádět úlohy při splnění stanovených podmínek. Úlohy mohou být dále spouštěny také dle časového plánu. Vzhledem k cílům práce se jedná převážně o provádění zavlažování, nicméně aplikace účel jednotlivých úloh nerozlišuje. Úlohy se tak liší především dobou provádění a výstupním pinem který ovládají.

Samotná aplikace nedisponuje grafickým uživatelským rozhraním, ale své funkce poskytuje v rámci REST API. Takto postavené aplikační rozhraní umožňuje předávání dat v režimu klient-server s využitím HTTP protokolu. Díky tomu mohla být systémová aplikace vyvíjena zcela nezávisle na klientské a naopak.

V aplikaci se lze setkat s následujícími technikami:

- Model-View-Controller (MVC) – návrhový vzor oddělující datovou část, uživatelské rozhraní a kontroler, který řídí aplikační logiku,
- Factory (tovární metoda) – návrhový vzor pro získání instance třídy bez použití konstruktoru, instanci zde vytváří metoda,
- Observer – návrhový vzor ve kterém jednoho vydavatele odebírá více posluchačů a při události na straně vydavatele jsou všichni přihlášení posluchači informováni,
- Dependency Injection (vkládání závislostí) – technika předání reference na objekt jinému objektu.

Dále je aplikace rozčleněna do těchto balíčků:

- hlavní balíček – obsahuje startovní bod celé aplikace, třídu *IrrigationApllication*, která inicializuje SpringBoot spolu s webovým serverem Tomcat,
- model – obsahuje třídy reprezentující entity systému a informace které nesou,
- controller – balíček tříd které slouží jako další vrstva pro práci s daty a starají se o aplikační logiku, tyto třídy úzce spolupracují s datovým modelem,
- api – třídy REST API pro navazující klientské aplikace,
- quartz – skládá se z konfigurace knihovny Quartz Enterprise Job Scheduler a tříd implementujících její rozhraní Job,
- repository – balíček služeb pro práci s databázovým úložištěm,

- security – je složením tříd zajišťujících autentizaci uživatelů a jejich autorizaci k provádění operací nad systémem.

Maven

Pro správu závislostí a automatizaci sestavování aplikace byl vybrán nástroj Maven. Základem tohoto nástroje je konfigurační XML soubor, který definuje model projektu. Velkou výhodou Mavenu je veřejný repositář, ve kterém si při sestavování projektu sám vyhledá a stáhne závislosti uvedené v projektu. V konfiguračním souboru je také možné definovat spouštění testů [25]. Konfigurační soubor modelu projektu nese název *pom.xml* a nachází se v kořenovém adresáři systémové aplikace.

Součástí modelu projektu jsou následující závislosti:

- skupiny *org.springframework.boot* – zahrnující Data JPA, Quartz, Security, Web, Test,
- *mysql-connector-java* – pro připojení k MySQL (MariaDB) databázi,
- *jjwt* – digitální podpis a hašování webových tokenů,
- *pi4j-gpio-extension* – pro přístup k GPIO rozhraní Raspberry Pi,
- *springfox-swagger-ui* – automatická dokumentace API.

4.1 Objektový model

Identifikace objektů vychází z požadavků na systém. Jedná se o třídy, jejichž objekty nesou informace, které přímo definují vnitřní stav systému. Na základě vlastností, které má zavlažovací systém splňovat, bylo možné definovat základní entity:

- Sensor,
- Output,
- ConditionalTask,
- ScheduledTask.

Některé atributy těchto entit bylo možné dále zobecnit, čímž vznikla řada dalších tříd, které se vážou na entity pomocí vzájemných vztahů. Především se jednalo o výčtové třídy:

- Channel – definuje pevně daný seznam kanálů, kterými na svém vstupu disponuje převodník MCP3008,
- Unit – specifikuje jednotky fyzikálních veličin pro připojené senzory,
- ComparisonOperator – určuje sadu operátorů, které slouží pro porovnávání hodnot v podmínkových úlohách.

Třída Sensor

Tato třída slouží k definování objektu snímače, který je připojen k převodníku. Každý takto definovaný snímač obsahuje jednoznačný číselný identifikátor, vlastní název a volitelný popisek. Jedním z atributů této třídy je *inputPin* typu *GpioPinAnalogInput*, který umožňuje knihovně Pi4J fyzicky rozlišit ke kterému vstupu je senzor doopravdy připojen. Tento atribut systém zároveň využívá k vyčítání hodnoty senzoru z převodníku. Atribut *unit* určuje fyzikální veličinu a jednotku, kterou daný senzor měří. Na základě této hodnoty pak pomocná třída *IrrigationUtils* naměřenou hodnotu (v rozsahu 0 až 1023) převede na hodnotu v požadovaných jednotkách.

Třída Output

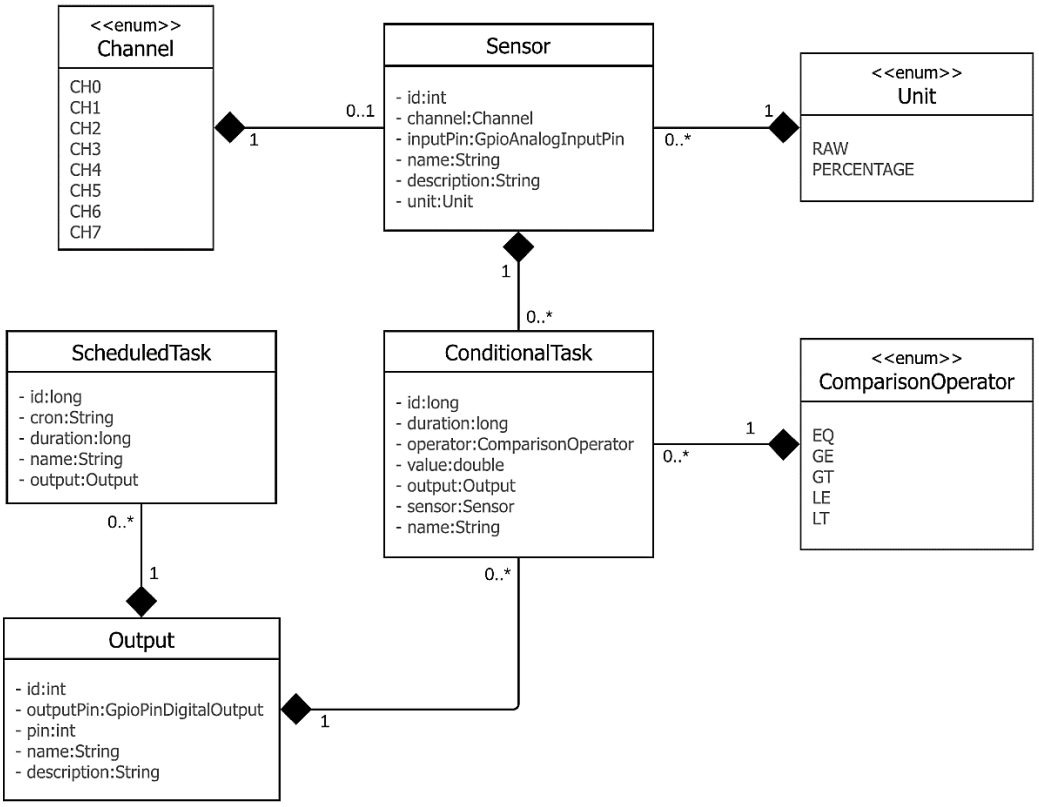
Slouží k obecnému definování některého z akčních členů (výstupů). Třída nerozlišuje, co je fyzicky připojeno k výstupu. Může to být čerpadlo, elektromagnetický ventil, ale také cokoliv jiného. K rozlišení v systému slouží pouze číselný identifikátor, název a popisek. Objekt této třídy je především definicí fyzického pinu GPIO rozhraní, ke kterému bude dále něco připojeno, respektive se na něm objeví logická 0 či 1.

Třída ScheduledTask

Objekt třídy ScheduledTask obsahuje informace potřebné k provádění úloh s časovým plánem. K rychlé identifikaci slouží číselné id a uživatelský název úlohy. Pro určení doby provádění úlohy byl zvolen unixový formát CRON, který je ve třídě zahrnut textovým atributem se stejnojmenným názvem. Takto stanovená úloha musí dále znát objekt, se kterým má při své aktivaci pracovat. Tím je reference na objekt třídy Output. Parametr *duration* pak stanovuje dobu, po kterou se má úloha provádět.

Třída ConditionalTask

Objekt třídy *ConditionalTask* nese informaci o úloze prováděné na základě podmínek. Spouštěčem takto stanovené úlohy je splnění podmínky. Dynamickou proměnnou této podmínky je atribut třídy *Sensor*, který slouží k čtení aktuální hodnoty senzoru. Podmínku dále tvoří operátor porovnávání (třídy ComparisonOperator) a výchozí hodnota, proti které je hodnota senzoru porovnávána. Kromě podmínky jsou dalšími atributy objekt *Output*, číselné id a uživatelský název úlohy.



Obrázek 14 – Diagram tříd základních objektů

Na uvedeném diagramu jsou mimo atributy tříd znázorněny také vztahy a multiplicity mezi těmito třídami. Jedná se o vztahy tzv. kompozice. Tento vztah lze popsat například na objektu typu *ConditionalTask*. U tohoto objektu je jasné, že pokud by nenesl informaci o výstupu, vstupu, nebo dokonce porovnávacím operátoru, tak by se stal neúplným až zbytečným. Nebylo by možné vyhodnotit podmínku či aktivovat výstup.

4.2 Perzistentní vrstva

Pro uchovávání dat v databázi slouží perzistentní vrstva. Aplikace využívá specifikace JPA (Java Persistence API), která je součástí Java EE (Java Enterprise Edition). Toto rozhraní poskytuje programátorovi prostředky objektově-relačního mapování (ORM). Pomocí anotace *@Entity* je například možné definovat, které objekty budou do databáze uchovávány. Anotace *@Table* umožňuje specifikovat tabulku do které bude objekt ukládán. Pro specifikaci sloupce atributu objektu slouží *@Column*. Pomocí dalších anotací lze určovat primární klíče nebo dokonce stanovit vazby mezi tabulkami.

Přestože bylo JPA původně vytvořeno pouze pro práci s relačními databázemi, tak ho lze dnes používat i pro jiná datová uložiště, například NoSQL databáze. Samotné JPA není aplikačním rámcem, který by implementoval způsob, jakým se má s objekty a databází pracovat.

Nejznámější ORM implementací je framework Hibernate, který byl zároveň použit i pro zavlažovací systém. Hibernate se taktéž stará o vytvoření databázového modelu, který sestavuje na základě použitých JPA anotací [26].

Aplikace ukládá do databáze všechny třídy objektového návrhu, které nesou informace o stavu systému. Tento návrh doplňují další 2 třídy, které slouží pouze pro uchování dat v databázi.

Třída SensorLog

Objekt této třídy je záznamem hodnoty z daného senzoru. Obsahuje unikátní identifikátor, identifikátor senzoru, naměřenou hodnotu a čas zaznamenání.

Třída User

Přestože systém vnitřně nerozlišuje uživatele systému ani jejich role, tak byla vytvořena ORM třída *User*. Ta slouží pro účely autentizace uživatele pomocí jeho uživatelského jména a hesla. Implementace této ORM třídy je naznačena v následující ukázce.

```
@Entity
@Table(name = "user")
public class User {
      @Id
      @GeneratedValue(strategy = GenerationType.IDENTITY)
      @Column(name = "id_user", updatable = false, nullable = false)
      private int id;

      @NotBlank
      @Column(nullable = false, unique = true)
      private String username;

      @NotBlank
      @Column(nullable = false)
      @JsonIgnore
      private String password;

      @CreationTimestamp
      @Column(nullable = false)
      private LocalDateTime created;
      // gettery a settery
}
```

4.2.1 Repozitáře

Při práci s databází je běžně nutné definovat celou řadu metod, jejichž struktura se u jednotlivých objektů velmi často liší pouze v maličkostech. Tento proces značně zjednodušuje modul Spring Data JPA, který tuto činnost vykoná za programátora. Spring Data poskytuje obecné rozhraní tzv. Repository, které lze dědit v rozhraních s vlastními definicemi

metod. Názvy těchto metod musejí být složeny z klíčových slov[14]. Na základě názvů pak Spring Data tyto metody automaticky naimplementuje.

V balíčku *Repository* jsou definovány rozhraní pro všechny tabulky, přičemž balíček obsahuje tato rozhraní:

- ISensorRepository – senzory,
- IOutputRepository – výstupy,
- ISchedTaskRepository – úlohy s časovým plánem,
- IConditionTaskRepository – úlohy s podmínkou,
- ISensorLogRepository – historie měření senzorů,
- IUserRepository – oprávnění uživatelé systému.

Ukázka repozitáře uživatelů

```
@Repository
public interface IUserRepository extends CrudRepository<User, Integer> {
     public Optional<User> findByUsername(String username);
     public boolean existsUserByUsername(String name);
}
```

4.3 Správa senzorů a výstupů

Ke správě senzorů a výstupů slouží třída *IrrigationService*, která je označena anotací *@Service*. Takto označené třídy Spring rozpozná jako tzv. beany a zaregistruje je do své pomocné třídy *ApplicationContext*. Zaregistrované beany pak Spring může pomocí vkládání závislostí poskytnout dalším třídám. Této vlastnosti je v aplikaci využito například u REST kontrolerů.

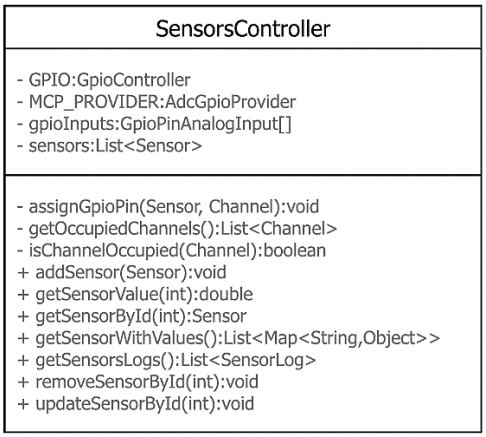
Služba *IrrigationService* se zároveň stará o inicializaci *GpioControlleru* z knihovny Pi4J. K tomu využívá třídy *GpioFactory*, která má statickou metodu *getInstance*. Tato metoda neposkytuje žádné mechanismy, které by zabránily vícevláknovým aplikacím vytvoření více instancí. S tímto faktem bylo nutné počítat při implementaci dalších částí. Tento fakt byl zároveň důvodem, proč je třída *IrrigationService* anotována jako služba. Spring totiž ve svém výchozím nastavení takovéto třídy vytváří dle návrhového vzoru Singleton, který zajistí, že se v programu bude nacházet vždy právě jediná instance. Ostatní třídy pak získávají instanci *GpioControlleru* uvedením závislosti na této službě.

[14] https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repository-query-keywords

Součástí služby *IrrigationService* jsou také kontrolery *SensorsController* a *OutputsController*. Ty se starají o aplikační logiku senzorů a výstupů. Služba pak tvoří mezivrstvu mezi dalšími službami pro práci s databází a datovým modelem.

SensorsController

Smyslem této třídy je poskytnutí rozhraní pro správu všech snímačů. Jednotlivé instance senzorů jsou udržovány v generické kolekci s dynamickou velikostí. Dalším důležitým atributem třídy je *AdcGpioProvider* z knihovny Pi4J, který slouží pro přístup k analogově-digitálnímu převodníku MCP3008. Pro přístup k jednotlivým vstupním pinům převodníku slouží rozhraní *GpioPinAnalogInput*. Přičemž instance pinů implementujících toto rozhraní udržuje kontroler v poli.



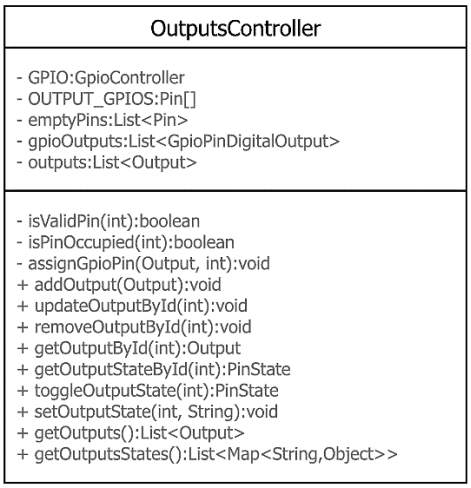
Obrázek 15 – Třída SensorsController

Metody kontroleru:

- *assignGpioPin* – přiřazuje senzoru referenci na pin GPIO rozhraní ze seznamu pinů, popřípadě vytvoří novou instanci pinu,
- *getOccupiedChannels* – vrací seznam obsazených kanálů převodníku,
- *isChannelOccupied* – zjistí, zdali je daný kanál obsazen,
- *addSensor* – přidá nový sensor do systému,
- *removeSensorById* – odstraní senzor,
- *updateSensorById* – upraví vlastnosti senzoru,
- *getSensorById* – vrací senzor se zadaným id,
- *getSensorValue* – zjistí hodnotu daného senzoru,
- *getSensorWithValues* – vrací objekt senzoru obohacený o jeho hodnotu,
- *getSensorsLogs* – zaznamená hodnoty všech senzorů (objekty třídy SensorLog) a vrátí je v kolekci.

OutputsController

Obdobně jako u senzorů je zde kolekce jednotlivých výstupů. Přístup k pinům GPIO rozhraní zajišťuje instance třídy *GpioController*. Součástí třídy je konstanta, která definuje pole pinů použitelných pro výstupy. Dále je zde kolekce pinů, které jsou volné a nejsou používány. Chybět nesmí ani kolekce objektů třídy *GpioPinDigitalOutput*, které umožňují fyzické ovládání daného výstupu.



Obrázek 16 – Třída OutputsController

Metody kontroleru:

- *isValidPin* – ověří, zdali lze pin se zadaným číslem použít jako výstupní,
- *isPinOccupied* – zjistí, zdali je zadaný pin obsazený,
- *assignGpioPin* – přiřadí výstupu referenci na pin GPIO rozhraní,
- *addOutput* – přidá výstup do systému,
- *updateOutputById* – upraví vlastnosti výstupu,
- *removeOutputById* – odebere výstup ze systému,
- *getOutputById* – vrací výstup se zadaným id,
- *getOutputStateById* – vrátí stav ve kterém se výstup nachází,
- *toggleOutputState* – přepne výstup do opačného stavu,
- *setOutputState* – nastaví výstup do stanoveného stavu,
- *getOutputs* – vrací kolekci všech výstupů v systému,
- *getOutputsStates* – vrací seznam výstupů obohacených o jejich stav.

4.4 Služby plánování

Součástí mezivrstvy oddělující databázové služby a objektový model, jsou služby pro kalendářní a podmínkové plánování úloh. Ty jsou opět opatřeny anotací *@Service*, čímž je zajištěna jejich snadná dostupnost a jedinečnost. Společným jmenovatelem těchto služeb je vytváření paralelně prováděných úloh. V praxi to znamená, že každá úloha, kterou systém v daný okamžik provádí, musí běžet ve svém vlákně. Proto je do systému zařazen Quartz Enterprise Job Scheduler (dále jen Quartz), který dokáže automaticky spouštět úlohy (vlákna) ve specifikovaný okamžik.

Quartz Enterprise Job Scheduler

Quartz má v aplikaci svůj vlastní balíček, kde se mimo jím prováděné úlohy nachází také třídy pro základní konfiguraci. První je třída *QuartzAutowireJobBean,* jejíž jedinou funkcí je zařazení kontextu plánovače, spouštěčů a dat do kontextu Springu. Druhou třídou je *QuartzSchedulerConfig*, která nastavuje třídu *ScheadulerFactoryBean* dle konfiguračního souboru. Konfigurační soubor s názvem *quartz.properties* se nachází v adresáři *resources* a má následující možnosti nastavení.

```
# Způsob uchování dat plánovače, výchozí hodnota RAM
org.quartz.jobStore.class=org.quartz.simpl.RAMJobStore
# Počet vláken připravených pro provádění úloh
org.quartz.threadPool.threadCount=10
# Název plánovače
org.quartz.scheduler.instanceName=irrigationQuartz
# Identifikátor plánovače
org.quartz.scheduler.instanceId=AUTO
# Nastavení pluginu pro zachycení ukončení JVM
org.quartz.plugin.shutdownhook.class=org.quartz.plugins.management.ShutdownHookPlug
in
# Zapnutí pluginu pro vypnutí plánovače spolu s vypnutím JVM
org.quartz.plugin.shutdownhook.cleanShutdown=true
```

4.4.1 Úlohy s časovým plánem

Požadavky na systém definovaly hned několik případů, ve kterých bylo nutné zajistit provádění operací ve stanovenou dobu, nebo dokonce s pravidelným opakováním. Přičemž se jednalo o tyto činnosti:

- plánované provádění zavlažování,
- pravidelné zaznamenávání dat ze senzorů,
- mazání starých záznamů ze senzorů.

Rozhraní pro správu těchto činností zajišťuje třída *SchedulesService*, která je anotována jako služba. Všechny plánované úlohy (třídy *ScheduledTask*) jsou udržovány v dynamické kolekci. Spring dále dosazuje referenci na instanci třídy *ConfigService*, která poskytuje rozhraní pro konfiguraci intervalu zaznamenávání hodnot a mazání starých záznamů. O práci s konfiguračním souborem se stará Preferences API obsažené v Javě. Soubor se nachází v uživatelském adresáři Raspberry Pi na následující cestě.

```
~/.java/.userPrefs/irrigation/prefs.xml
```

Služba plánovače obsahuje privátní metody pro tvorbu spouštěčů a detailů úloh pro práci s knihovnou Quartz. Takto vytvořené objekty už plánovač zná a umí s nimi pracovat.

Plánování zavlažování

Aby bylo možné zavlažování plánovat, tak bylo nutné specifikovat, jak se má samotné zavlažování v programu vykonat. Vznikla tak třída *ScheduleJob*, která je potomkem třídy *QuartzJobBean*. V tomto důsledku bylo nutné implementovat metodu *executeInternal*, o jejíž vykonávání se stará plánovač. Implementace této metody je naznačena v následujícím kódu (pro přehlednost bylo odstraněno zachycení a zpracování výjimky).

```
@Override
protected void executeInternal(JobExecutionContext jec) throws
JobExecutionException {
  Output output = (Output) jec.getJobDetail().getJobDataMap().get("output");
  long duration = jec.getJobDetail().getJobDataMap().getLongValue("duration");
  output.setHigh();
  Thread.sleep(duration);
  output.setLow();
}
```

Metoda se skládá z následujících kroků:

1. získání instance výstupu a doby provádění z datového kontextu úlohy,
2. přepnutí výstupu do stavu High (logická hodnota 1),
3. uspání úlohy na stanovenou dobu provádění,
4. přepnutí výstupu zpět do stavu Low (logická hodnota 0).

Pro účely plánování zavlažování disponuje *SchedulesService* metodami pro vytvoření, smazání a zjištění stavu úloh. Jedná se o metody:

- *addScheduledTask* – přidá úlohu do kolekce a zařadí ji do plánovače,
- *deleteScheduledTask* – odebere úlohu z kolekce i plánovače,
- *getCount* – vrátí počet naplánovaných úloh typu *ScheduledTask*,
- *getScheduledTasks* – vrátí seznam naplánovaných úloh,

- *getScheduledTasksStates* – vrátí seznam úloh spolu s jejich stavem,
- *getScheduledTaskById* – vrací instanci dané úlohy,
- *getJobState* – vrací stav dané úlohy.

Část metody *addScheduledTask*, která se stará o vytvoření datového kontextu, spouštěče a zařazení úlohy do plánovače, je implementována následovně.

```
JobDataMap jdm = new JobDataMap();
jdm.put("duration", task.getDuration());
jdm.put("output", task.getOutput());
JobDetail job = buildJobDetail(task.getId(), ScheduleJob.class, jdm);
Trigger trig = buildJobTrigger(job, task.getCron());
Scheduler scheduler = schedulerBean.getScheduler();
scheduler.scheduleJob(job, trig);
```

Zaznamenávání dat ze senzorů

Pro tento účel slouží třída *LoggerJob*, která je opět potomkem *QuartzJobBean*. *LoggerJob* disponuje instancemi *IrrigationService* a *ISensorLogRepository*, které jsou dosazeny Springem pomocí vkládání závislostí. První služba slouží pro přístup k jednotlivým senzorům a druhá poskytuje rozhraní pro přístup k databázi. Překrytá metoda *executeInternal* se pak stará o získání aktuálních hodnot senzorů, které následně ukládá do databáze. Pro spuštění a konfiguraci této úlohy poskytuje služba SchedulesService metody:

- *scheduleLoggerTask* – tato metoda slouží pro zařazení úlohy do plánovače a je vykonána při každém spuštění systémové aplikace,
- *updateLoggerTaskInterval* – změní interval zaznamenávání hodnot.

Mazání starých záznamů

Třída *CleanerJob* slouží k pravidelnému mazání starých záznamů z databáze. Aby mohla být spouštěna plánovačem, tak je opět definována jako potomek *QuartzJobBean*. Doba, po kterou mají být záznamy v databázi uchovávány, je definována uživatelským nastavením, které je získáno pomocí služby *ConfigService*. Výchozí hodnota stanovuje dobu uchování záznamů na 4 týdny, přičemž k samotnému spouštění úlohy dochází každý den ve 2 hodiny ráno. Pro spuštění a konfiguraci úlohy slouží metody:

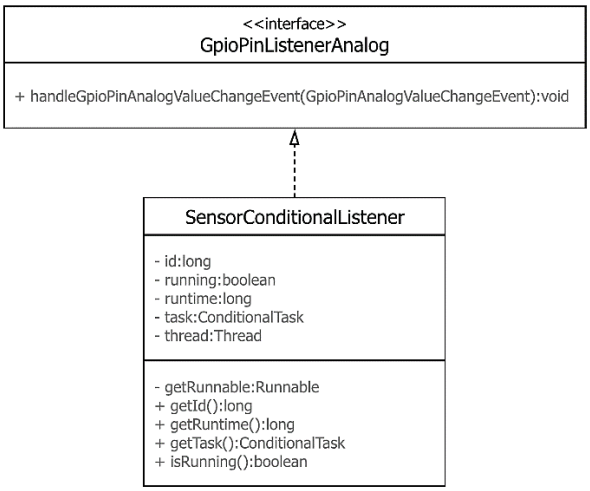
- *scheduleCleanerTask* – zařadí úlohu do plánovače, je vykonána při každém spuštění aplikace,
- *updateCleanerTaskLifetime* – změní dobu životnosti záznamů.

4.4.2 Podmíněné úlohy

Dalším požadavkem na aplikaci bylo spouštění zavlažování při splnění stanovených podmínek. Protože nelze takto stanovenou úlohu naplánovat na specifický čas, tak bylo nutné zvolit jiné řešení než v předchozím případě. Pro tento účel bylo využito vlastností knihovny Pi4J, která u analogových pinů respektuje návrhový vzor Observer. Díky této vlastnosti pak bylo možné přihlásit podmíněné úlohy ke změnám hodnot jednotlivých senzorů.

SensorConditionalListener

Pro přihlášení podmíněné úlohy ke změnám hodnot senozoru vznikla nová třída *SensorConditionalListener*, která implementuje rozhraní *GpioPinListenerAnalog*. Ta pro svou funkci obsahuje referenci na podmíněnou úlohu a informace o běhu úlohy.



Obrázek 17 – Diagram třídy SensorConditionalListener

Při změně hodnoty senzoru je vytvořena událost, která volá tzv. *handle* metodu Listeneru. V té následně dochází k ověření podmínky. Při jejím splnění je úloha spuštěna v novém vlákně a zároveň je zaznamenán i čas jejího spuštění. Dokud úloha běží, tak ji nemůže nová událost (změna hodnoty) opětovně spustit.

ConditionsService

Rozhraní pro práci s podmíněnými úlohami poskytuje třída *ConditionsService*, která je anotována jako služba. Služba poskytuje především následující veřejné metody:

- *addConditionalTask* – přidá úlohu do seznamu a přihlásí ji ke změnám senzorů,
- *deleteConditionalTask* – odebere úlohu ze seznamu a odhlásí ji ze změn senzorů,
- *getConditionalTaskById* – vrátí instanci dané úlohy,
- *getConditionalTaskState* – vrátí instanci dané úlohy spolu s jejím stavem,
- *getConditionalTasks* – vrátí seznam všech úloh,
- *getConditionalTasksStates* – vrátí seznam všech úloh s jejich stavy,

- *getCount* – vrátí počet podmíněných úloh typu *ConditionalTask*.

4.5 Aplikační rozhraní

Aplikační rozhraní systémové aplikace je navrženo dle architektury REST. Tato architektura staví na zdrojích, které vždy reprezentují nějakou informaci. Takovým zdrojem může být například obrázek, dokument, JSON reprezentace objektu nebo dokonce celá kolekce dalších zdrojů. Zdroje od sebe musejí být odlišeny unikátními identifikátory. V případě použití REST architektury společně s HTTP protokolem je takovým identifikátorem URL adresa. REST rozhraní musí mimo jiné dodržovat další principy:

- klient-server – nezávislost rozhraní klienta a serveru,
- bezstavovost – požadavky musejí obsahovat veškeré potřebné informace ke zpracování požadavku, stav je uchováván pouze na straně klienta,
- kešovatelnost – rozlišení kešovatelných požadavků, výhodou kešovaných odpovědí je rychlost a škálovatelnost [27].

Protokol HTTP poskytuje pro zaslání požadavku na zdroj řadu metod. Ty se liší podle operace, která se má se zdrojem provádět. Rozlišování těchto metod není pro funkčnost REST rozhraní zcela nutné, ale je vhodné jej dodržet alespoň pro účely přehlednosti. HTTP má k dispozici tyto dotazovací metody:

- *GET* – získání dat,
- *POST* – vytvoření dat,
- *PUT* – úprava dat,
- *DELETE* – smazání dat.

Odpověď na požadavek zaslaný jednou z předchozích metod se skládá z hlavičky s metadaty a těla zprávy, které obsahuje vyžádaná data. Součástí hlavičky jsou mimo jiné stavové kódy, které podávají informaci o tom, jak byl požadavek zpracován. V aplikaci se lze setkat s následujícími stavovými kódy:

- *200 OK* – požadavek byl v pořádku zpracován,
- *201 CREATED* – na základě požadavku došlo k vytvoření zdroje,
- *400 BAD REQUEST* – požadavek nemohl být zpracován kvůli jeho špatné specifikaci,
- *401 UNAUTHORIZED* – požadavek pro své zpracování potřebuje autorizaci,
- *404 NOT FOUND* – požadovaný zdroj není dostupný,
- *500 INTERNAL SERVER ERROR* – blíže nespecifikovaná chyba na straně serveru.

4.5.1 REST kontrolery

Ke zdrojům lze v rámci aplikačního rozhraní přistupovat skrze tzv. endpointy (koncové body). Koncovým bodem je unikátní URL adresa, na kterou je zasílán požadavek. V backendu aplikaci systému zavlažování jsou tyto koncové body poskytovány třídami z balíčku Api. Ty využívají schopností modulu Spring Web a Spring Boot. Především pak *DispatcherServletu*, který se stará o nasměrování HTTP požadavků na správné místo v programu. Ten ke svému běhu navíc vyžaduje servletový kontejner, přičemž aplikace pro tento účel využívá vestavěného webového serveru Apache Tomcat.

Pro přiřazení požadavků ke správným třídám a jejich metodám bylo nutné využít anotace *@RestController*. Tato anotace dává najevo, že takto označené třídy mohou být směrovány *DispatcherServletem* a návratové hodnoty jejich metod mají být vloženy přímo do těla odpovědi. Spring navíc objekty, které jsou součástí odpovědi, automaticky převádí do formátu JSON. Aby bylo možné požadavky přiřadit ke správným metodám, musí být metody mapovány na svou URL. Pro tento účel slouží několik anotací, které se liší podle typu zasílaného dotazu:

- *@GetMapping*,
- *@PostMapping*,
- *@PutMapping*,
- *@DeleteMapping*.

REST kontrolery z balíčku Api pro svou činnost hojně využívají vkládání závislostí, zejména pro repozitáře a služby popsané v předchozích kapitolách. Koncové body jsou definovány pomocí kontrolerů:

- AuthenticationRest – správa uživatelských účtů a autentizace,
- SensorRest – správa snímačů,
- OutputRest – správa výstupů,
- ScheduledTaskRest – správa plánovaných úloh,
- ConditionalTaskRest – správa podmíněných úloh.

Jedním z koncových bodů REST kontroleru *SensorRest*, který navíc využívá mapování adresy pomocí anotace *@RequestMapping("/api/sensors")*, je například metoda *getSensorLogs*. Ta slouží k získání historie měření senzorů z databáze.

Implementace metody *getSensorLogs* je znázorněna v následující ukázce kódu.

```
@GetMapping("/logs")
public ResponseEntity<?> getSensorsLogs(@Valid @RequestParam int weeks) {
     LocalDateTime date = IrrigationUtils.getDateTimeBeforeWeeks(weeks);
     List<SensorLog> logs =
       logsRepository.findByCreatedAfterOrderByCreatedAsc(date);
     return ResponseEntity.ok(logs);
}
```

Tabulka 5 – Koncový bod historie měření

SensorRest			
/api/sensors			
URL	Požadavek	Parametr dotazu	Popis
/logs	GET	weeks:integer	vrací historii měření senzorů mladší než zadaný počet týdnů

4.5.2 Zpracování výjimek

Stejně jako u kteréhokoliv jiného programu, tak i u aplikace pro zavlažování může docházet k neočekávaným stavům. Ty mohou být způsobené samotným autorem programu, ale také uživatelským vstupem. Příkladem může být zadání záporného čísla kanálu při vytváření senzoru. Aplikace na takový stav není navržena a je nutné tuto událost včas odchytit, tak aby nedošlo k dalším chybovým stavům či dokonce pádu aplikace. Nicméně zachycením této události nenastane výsledek, který uživatel očekával, proto je nutné mu tuto informaci nějakým způsobem sdělit.

Pro tento účel byl vytvořen speciální REST kontroler *IrrigationExceptionHandler*, který veškeré výjimky při zpracovávání požadavků zaznamená a odešle odpověď s chybovým kódem a dalšími informacemi. Tento kontroler je potomkem třídy *ResponseEntityExceptionHandler* a spolu s anotací *@ControllerAdvice* dává Springu najevo, že mají být výjimky směřovány právě sem. Jeho metody mají vždy anotaci *@ExceptionHandler* s parametrem třídy výjimky, kterou zpracovávají.

Implementace metody pro zpracování všech blíže nespecifikovaných výjimek je uvedena v následující ukázce kódu třídy *IrrigationExceptionHandler*.

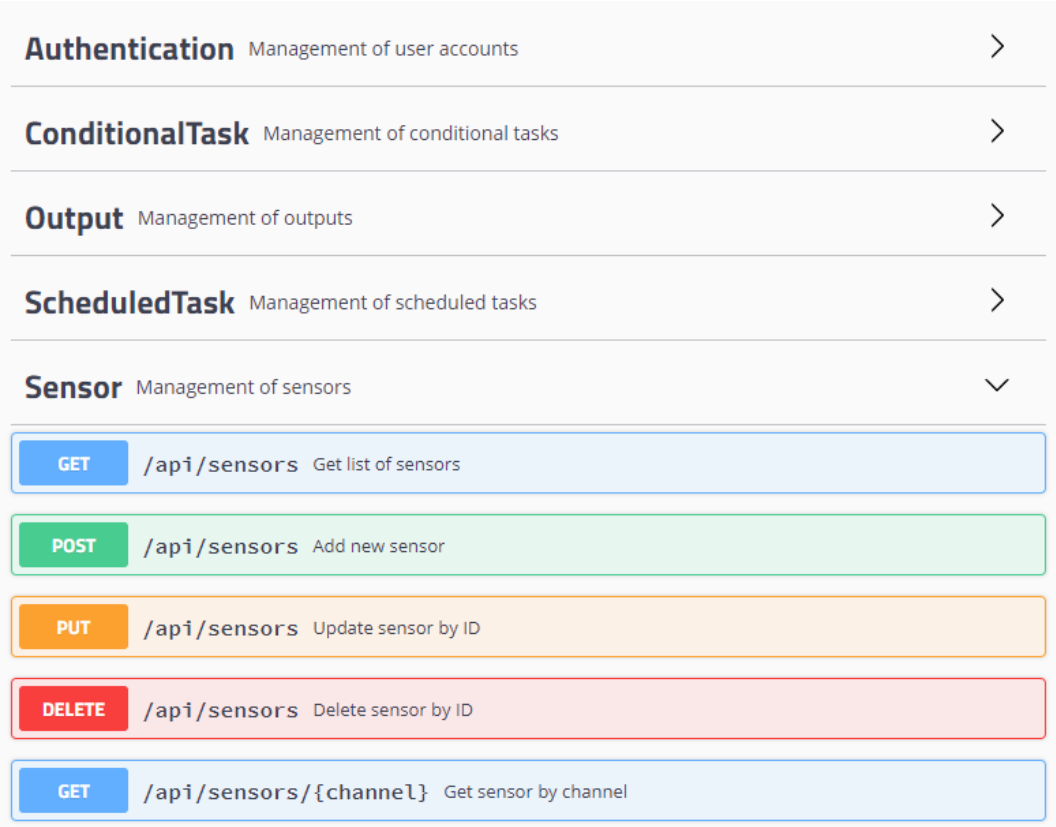
```
@ControllerAdvice
public class IrrigationExceptionHandler extends ResponseEntityExceptionHandler {

  @ExceptionHandler(Exception.class)
  public final ResponseEntity<ErrorResponse> handleAllExceptions(Exception ex,
        WebRequest request) {
     ErrorResponse error = new ErrorResponse(new Date(), 500, ex.getMessage(),
     request.getDescription(false));
     return new ResponseEntity<>(error, HttpStatus.INTERNAL_SERVER_ERROR);
  }
  // ostatní metody
}
```

4.5.3 Dokumentace Swagger UI

Swagger UI je součástí sady nástrojů Swagger, které usnadňují návrh, testování, dokumentaci a dodržování zásad REST API. Samotný Swagger UI pak slouží pro automatickou tvorbu přehledné dokumentace a testování API. Pro tento účel Swagger prochází strukturu projektu a hledá koncové body rozhraní. Při tomto prohledávání zároveň zaznamenává datový typ návratové hodnoty a jeho parametrů. Po dokončení těchto operací vytvoří nový koncový bod, který výslednou dokumentaci zpřístupní a Swagger UI ji přehledně vyobrazí pomocí webové stránky. Výsledná dokumentace je dostupná na následující adrese.

```
http://<ADRESA>:<PORT>/swagger-ui.html
```

Pro zprovoznění Swaggeru byla vytvořena konfigurační třída *SwaggerConfig* označená anotací *@EnableSwagger2*. Součástí třídy je tzv. Docket bean, která zpřístupňuje koncové body aplikace a zároveň umožňuje doplnění dalších informací. Pomocí dalších anotací pak lze detailněji popsat koncové body, jejich parametry i návratové hodnoty. Ukázka výsledné dokumentace REST API zavlažovacího systému je uvedena na obrázku 18.



Obrázek 18 – Ukázka dokumentace Swagger UI

4.5.4 Zabezpečení

Z požadavků na zavlažovací systém vyplývá, že má sloužit pouze omezenému počtu oprávněných uživatelů a jeho funkce nemají být veřejně dostupné. Nicméně dalším z požadavků je zároveň zajištění vzdáleného přístupu k systému. Aby bylo možné tyto požadavky skloubit, tak bylo nutné vybrat vhodný způsob zabezpečení, který umožní ověření uživatele s následným zpřístupněním funkcí systému. Pro tento účel byl zvolen modul Spring Security, který obsahuje nástroje pro autentizaci a autorizaci. Výhodou tohoto modulu je snadná integrace do již vytvořeného aplikačního rozhraní.

Proces zpřístupnění zavlažovacího systému se skládá ze dvou hlavních částí:

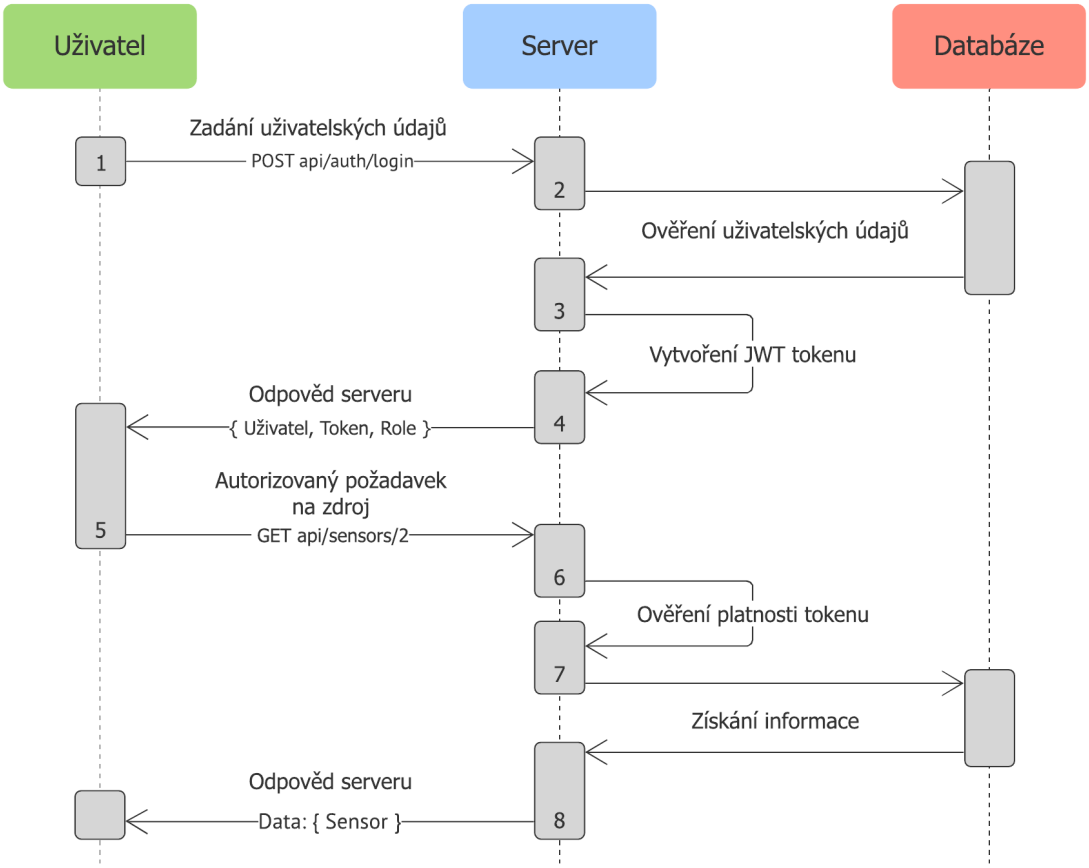
- autentizace – dochází k ověření identity uživatele pomocí jeho přístupových údajů (uživatelské jméno a heslo),
- autorizace – následuje ihned po autentizaci a jedná se o proces ověření oprávnění k provádění daných činností.

Pro zajištění autenticity komunikace mezi klientem a serverem byl zvolen JSON Web Token (JWT), který se skládá z hlavičky, přenášených dat a podpisu. Hlavička nese informaci o algoritmu použitém k vytvoření podpisu. Součástí datové části jsou mimo jiné také

informace o vydavateli a expiraci podpisu. Podpis se skládá ze zakódované hlavičky a dat, přičemž jeho další částí je tajný klíč [28].

Při samotném používání aplikačního rozhraní jsou prováděny tyto kroky:

1. Uživatel zadá své uživatelské údaje pomocí koncového bodu, který nevyžaduje autorizaci.
2. Server porovná zadané údaje s databází.
3. Pokud údaje souhlasí, tak je uživatel ověřen a následně je mu vygenerován autorizační JWT token.
4. Server zahrne tento token do své odpovědi a zašle ji uživateli.
5. Uživatel zadá autorizovaný požadavek na zdroj.
6. Server ověří platnost přijatého tokenu.
7. Je-li token platný, tak je požadavek na zdroj zpracován.
8. Server odešle informaci o zpracování požadavku.



Obrázek 19 – Zabezpečení aplikačního rozhraní

Implementační část je z větší části inspirována uvedenou literaturou (viz. [28]). V aplikaci ji lze nalézt v balíčku *Security*, jehož součástí jsou mimo jiné následující třídy.

SecurityConfig

Jedná se o konfigurační třídu Spring Security, která specifikuje oprávnění přístupů ke koncovým bodům. Nastavení této třídy povoluje zasílání požadavků na části Swagger UI a přihlašování do systému. Všechny ostatní požadavky vyžadují pro přístup ověření uživatele. Pro tento účel je pro všechny HTTP požadavky přidán autentizační filtr. Dalším nastavením je zpracování neúspěšných pokusů o autentizaci. O ty se stará třída *AuthenticationHandler*.

UserDetailsServiceImpl

Smyslem této třídy je nalezení a načtení uživatele z databáze na základě zadaného přihlašovacího jména. Pokud uživatel vůbec neexistuje, tak je vyhozena výjimka. Je-li uživatel nalezen, tak se s pomocí třídy *AuthenticationManagerBuilder* provede porovnání hašovaných hesel. Systém pro hašování hesel využívá hašovací funkci Bcrypt.

AuthTokenProvider

Pomocná třída pro generování a validaci tokenů. Pro svou funkci využívá knihovnu Jsonwebtoken, která umožňuje stanovení doby expirace, podpisového algoritmu, vydavatele a dalších údajů.

AuthEntryPointImpl

Definuje filtr aplikovaný na každý požadavek, který vyžaduje oprávnění. Z přijatého požadavku vyjme token a ověří jeho platnost. Následně udělí uživateli oprávnění v rámci bezpečnostního kontextu.

Dále je nutné podotknout, že výše uvedené řešení je dostatečně zabezpečené až s použitím zabezpečeného HTTPS protokolu. U nezabezpečeného HTTP hrozí riziko tzv. man-in-the-middle útoků.

4.6 Konfigurace

Hlavní konfigurační soubor celého systému se nachází v adresáři *resources* a nese název *application.properties*. Jsou v něm obsaženy výchozí hodnoty uživatelských nastavení, například interval zaznamenávání hodnot ze senzorů, životnost těchto záznamů, ale také výchozí přihlašovací údaje, které je vhodné před prvním spuštěním změnit.

Další 2 kategorie nastavení jsou technického rázu a zajišťují správnou funkci systému. První se týká vlastností JWT tokenu, přičemž lze u něj změnit typ tokenu, dobu expirace a tajný klíč. Změny těchto hodnot nejsou nutné, ale je vhodné upravit například dobu expirace. Druhou velice důležitou kategorií je nastavení perzistentní části aplikace. Zcela zásadním pro funkci celé aplikace je pak konfigurace údajů pro připojení k databázi. Ty se musí shodovat s nastavením databáze a mohou vypadat následovně.

```
# Adresa pro připojení k databázi, včetně názvu databáze
spring.datasource.url=jdbc:mysql://localhost:3306/db_irrigation
# Uživatelské jméno
spring.datasource.username=spring
# Uživatelské heslo
spring.datasource.password=letmein
```

Komunikace s klientskou aplikací

Pokud se uživatel bude k systému připojovat ze stejné sítě, tak zpravidla není nutné provádět žádná další nastavení. Problém by ovšem nastal při připojení z jiné sítě. V tomto případě je nutné počítat s použitím virtuální privátní sítě (VPN), ze které bude možné k systému přistupovat. Další možností, která ale vyžaduje veřejnou IP adresu, je tzv. port forwarding (směrování portů) síťových uzlů. Při tomto řešení klient zasílá požadavky přímo na veřejnou adresu se správným portem a směrovač je dále adresuje na Raspberry Pi.

Do budoucna by bylo možné stávající řešení rozšířit o veřejný server, který bude poskytovat API. Požadavky zaslané na tento server by byly ukládány do fronty a zavlažovací aplikace by se pak v pravidelných intervalech dotazovala serveru, zdali existují požadavky ke zpracování. Tyto požadavky by postupně vyřizovala a zasílala zpět na veřejně dostupný server.

5 FRONTEND APLIKACE SYSTÉMU ZAVLAŽOVÁNÍ

Smyslem webové frontend aplikace je vytvoření grafického rozhraní, které uživateli poskytne snadný způsob ovládání a rychlý přehled nad systémem. Řešení klientské části formou webové stránky s sebou nese řadu výhod v podobě platformní nezávislosti a snadné distribuce. Jediným požadavkem, který musí cílové zařízení splňovat, je schopnost spuštění aplikace v některém z moderních webových prohlížečů s podporou JavaScriptu. Uživatel tak může zavlažovací systém ovládat běžným způsobem ze svého počítače či mobilu, ale také například pomocí chytré televize.

Pro realizaci webu byla zvolena forma tzv. single-page aplikace (SPA), která uživateli umožňuje procházet stránkami bez znatelných přerušení. Oproti běžným webovým stránkám zde totiž při uživatelské interakci nedochází k opětovnému načítání stránek. Web je načten pouze jednou a obsah stránky je poté dynamicky upravován pomocí JavaScriptu. Protože by byla tvorba takovéto aplikace pomocí čistého JavaScriptu značně zdlouhavá, tak byl zvolen aplikační rámec Angular 8, který potřebné mechanismy pro dynamické úpravy stránek poskytuje.

Pro vývoj Angular aplikací je zapotřebí mít nainstalované Node.js[15] spolu s Angular CLI, které rozšiřuje příkazovou řádku o řadu nápomocných příkazů. Tím úplně prvním, se kterým se lze setkat, je příkaz pro vytvoření nového Angular projektu.

```
ng new <název projektu> [další možnosti]
```

5.1 Struktura aplikace

Součástí každého Angular projektu je předdefinovaná adresářová struktura, konfigurační soubory a soubory tvořící webovou prezentaci. Přičemž těmi nejdůležitějšími částmi jsou:

- node_modules – adresář pro moduly třetích stran,
- package.json – konfigurační soubor projektu definující závislosti,
- src/ – adresář zdrojových souborů,
    - main.ts – vstupní bod aplikace, který zavádí kořenový modul,
    - index.html – hlavní HTML stránka do které jsou vkládány další části,
    - styles.css – definice kaskádových stylů,
    - assets – adresář pro multimedia a další soubory,
    - app/ – adresář pro aplikační logiku,

[15] https://nodejs.org/en/about

- app.module.ts – kořenový modul s deklaracemi dalších komponent (slouží k sestavení aplikace),
- app-routing.module.ts – obsahuje definice URL adres pro jednotlivé komponenty.

5.2 Konzumace REST API

Komunikaci s backend aplikací obstarává servisní vrstva složená z tříd označených jako služby. Služby se v Angularu definují anotací *@Injectable* a pomocí vkládání závislostí je možné k nim přistupovat z ostatních částí aplikace. Implementace těchto tříd úzce vychází z dokumentace REST API backend aplikace. Jelikož komunikace s API probíhá skrze HTTP protokol, tak se lze v každé službě setkat s importem třídy *HttpClient* zastřešující rozhraní pro dotazovací metody tohoto protokolu.

Pro účely jednotného nastavení adresy serveru byla vytvořena globální třída s příznačným názvem *Globals*. Nachází se v adresáři *app* a adresa je v ní definována pomocí veřejné konstanty *API_URL*. Díky tomu bylo možné ve službách konzumujících API definovat URL koncových bodů pouze částečně a následně s použitím konstanty složit celou adresu.

Nové služby lze vytvářet pomocí následujícího příkazu Angular CLI.

```
ng generate service <název služby> [další možnosti]
```

Ve webové aplikaci se služby API nacházejí v adresáři services a jedná se o následující:

- AuthenticationService – správa uživatelských účtů a autentizace uživatele,
- SensorsService – správa senzorů,
- OutputsService – správa výstupů,
- SchedulesService – kalendářní plánování úloh,
- ConditionsService – podmíněné plánování úloh.

Implementace těchto služeb jsou si z velké části podobné. Výjimkou je pouze služba pro autentizaci, která kromě zasílání požadavků obstarává také informace o aktuální relaci. Ostatní služby disponují pouze metodami pro dané koncové body API. Pro ukázku je uveden fragment služby *SensorsService* a její metoda pro získání seznamu senzorů.

```
@Injectable({
  providedIn: 'root'
})
export class SensorsService {
  private sensorsUrl = Globals.API_URL.concat('/sensors');
  // adresy dalších koncových bodů

  constructor(private httpClient: HttpClient) {}

  getSensors(): Observable<any[]> {
     return this.httpClient.get<any[]>(this.sensorsUrl);
  }
  // ostatní metody API
}
```

Další velmi důležitou součástí komunikace s API je třída *ResponseInterceptor*, která implementuje rozhraní *HttpInterceptor*. Třídy implementující toto rozhraní slouží k zachytávání požadavků a odpovědí v rámci HTTP protokolu. Obsluhu pak zajišťuje metoda *intercept*. V případě *ResponseInterceptor* v ní dochází k zachytávání chybových stavů a dalších hlášek serveru.

5.2.1 Autentizace

Jak již bylo zmíněno v předchozí části, tak o přihlašování uživatele do systému se stará služba *AuthenticationService*, která zároveň ukládá přístupový token do relační proměnné prohlížeče. Nicméně tato služba nezajišťuje celou funkcionalitu, doplňují ji ještě tyto třídy:

- ResponseInterceptor – v případě odpovědi serveru s chybovým kódem 401 uživatele automaticky odhlásí ze systému (vymaže relační proměnnou),
- AccesInterceptor – zachytává odesílané požadavky a doplňuje jejich hlavičku o přístupový token,
- AuthenticationGuard – pokud se uživatel pokusí vstoupit do systému bez uloženého tokenu, tak je automaticky přesměrován na přihlašovací obrazovku [29].

5.3 Komponenty

Komponenty jsou základními stavebními prvky prezentační vrstvy. Vzhled komponenty určuje šablona, která je složena z HTML elementů a kaskádových stylů. Aplikační logiku komponenty pak definuje třída, která šablonu plní obsahem pomocí tzv. direktiv a navázání dat. Direktivy lze dále rozdělit:

- strukturální – přidávají či odebírají elementy HTML dokumentu,
- atributové – upravují atributy elementů (například styl),

- komponentové – každá komponenta má definovaný svůj selektor (název), pomocí kterého ji lze umístit do dokumentu jako kterýkoliv jiný HTML element.

Následujícím příkazem je možné vygenerovat novou komponentu, včetně souboru šablony, stylu a třídy.

```
ng generate component <název komponenty> [další možnosti]
```

5.3.1 Implementace

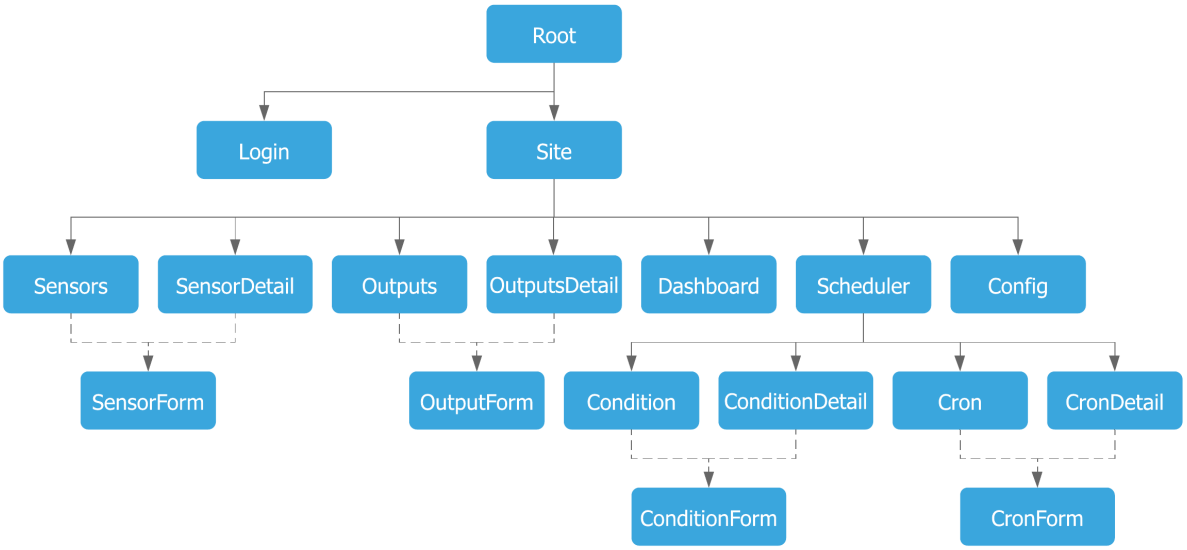
V aplikaci jsou navíc komponenty použity i pro dvě různá rozvržení webu. První varianta *LoginComponent* slouží pro přihlašovací stránku, ve které se nachází pouze přihlašovací formulář bez dalších komponent. V druhém rozvržení *SiteComponent* se nachází menu aplikace a kontejner pro hlavní obsahovou část, která se liší dle aktuální adresy v aplikaci. Obsahovou část doplňují komponenty:

- Dashboard – rychlý přehled celého systému,
- Outputs – přehled a správa výstupů,
- OutputDetail – detail vybraného výstupu,
- Sensors – přehled a správa senzorů,
- SensorDetail – detail vybraného senzoru,
- Scheduler – přehled plánování úloh,
    - ConditionSchedules – správa podmíněných úloh,
    - ConditionDetail – popis a aktuální stav podmíněné úlohy,
    - CronSchedules – správa úloh s časovým plánem,
    - CronDetail – popis a aktuální stav časované úlohy,
- Config – konfigurace systému.

Adresace těchto komponent v modulu *AppRouting* vypadá následovně.

```
const routes: Routes = [{
        path: '',
        component: SiteComponent,
        canActivate: [AuthenticationGuard],
        children: [
            {path: '', component: DashboardComponent, pathMatch: 'full'},
            {path: 'outputs', component: OutputsComponent},
            {path: 'outputs/:id', component: OutputDetailComponent}
                // ostatní komponenty ]},
    {path: 'login', component: LoginComponent},
    {path: '**', redirectTo: ''}
];
```

Hiearchie jednotlivých komponent je naznačena na obrázku 20, přičemž hlavní komponentou je kořen *Root*.



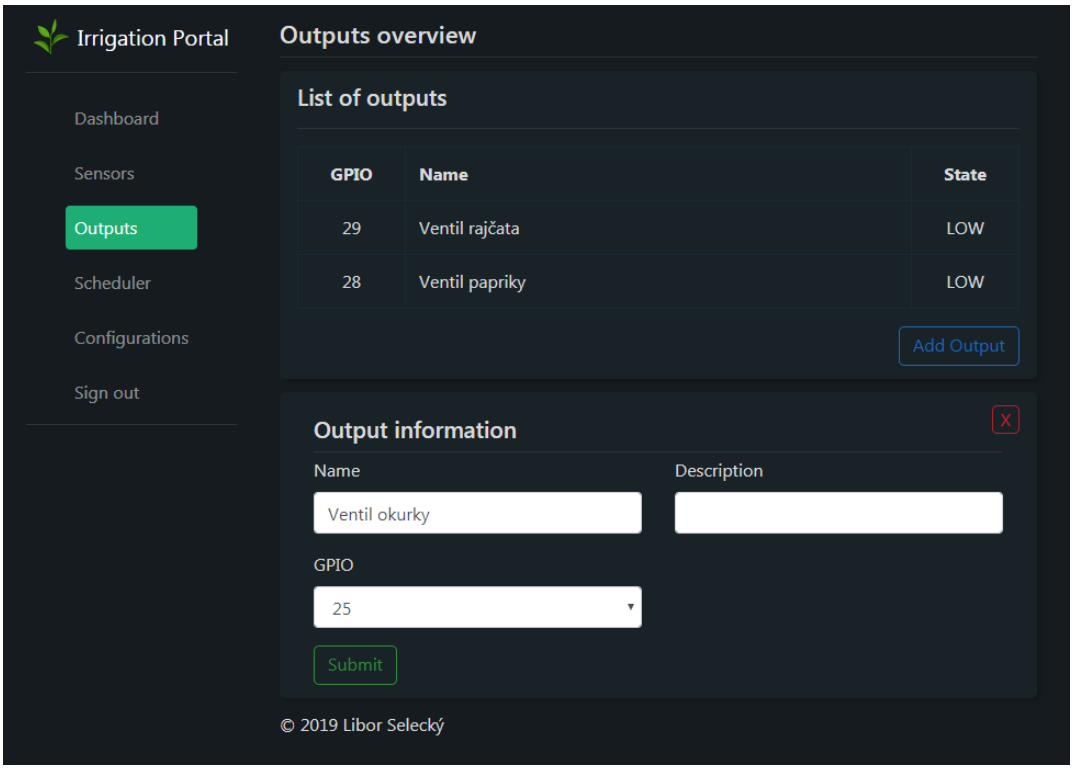
Obrázek 20 – Hiearchie komponent

Ukázka šablony výstupů

```
<tr *ngFor="let output of outputs" class="clickable-row"
      [routerLink]="['/outputs/', output.id]">
  <td class="text-center">{{output.pin}}</td>
  <td>{{output.name}}</td>
  <td class="text-center">{{output.state}}</td>
</tr>
```

V uvedeném fragmentu kódu lze vidět výpis jednotlivých řádků tabulky výstupů. Je zde vyznačena strukturální direktiva[16] *ngFor*, která prochází polem výstupů definovaného v třídě komponenty. Tato direktiva pro každý prvek pole vytvoří nový řádek a doplní ho o uvedené buňky tabulky. Ty jsou navíc ještě doplněny atributy aktuálního objektu. Pro tento účel se proměnné ohraničují 2 složenými závorkami. Vykreslení a doplnění uvedené šablony prohlížečem si lze prohlédnout v ukázce na obrázku 21.

[16] https://angular.io/guide/structural-directives



Obrázek 21 – Ukázka správy výstupů

5.3.2 Moduly třetích stran

Spolu s instalací Node.js má vývojář k dispozici i správce balíčků Npm. Pomocí tohoto správce je možné stahovat a instalovat hotové Javascriptové moduly. Npm tyto moduly stahuje ze své veřejné databáze[17], ve které lze mimo jiné nalézt i celou řadu modulů určených přímo pro Angular. Instalaci nového modulu lze provést následujícím příkazem.

```
npm install <název balíčku> --save
```

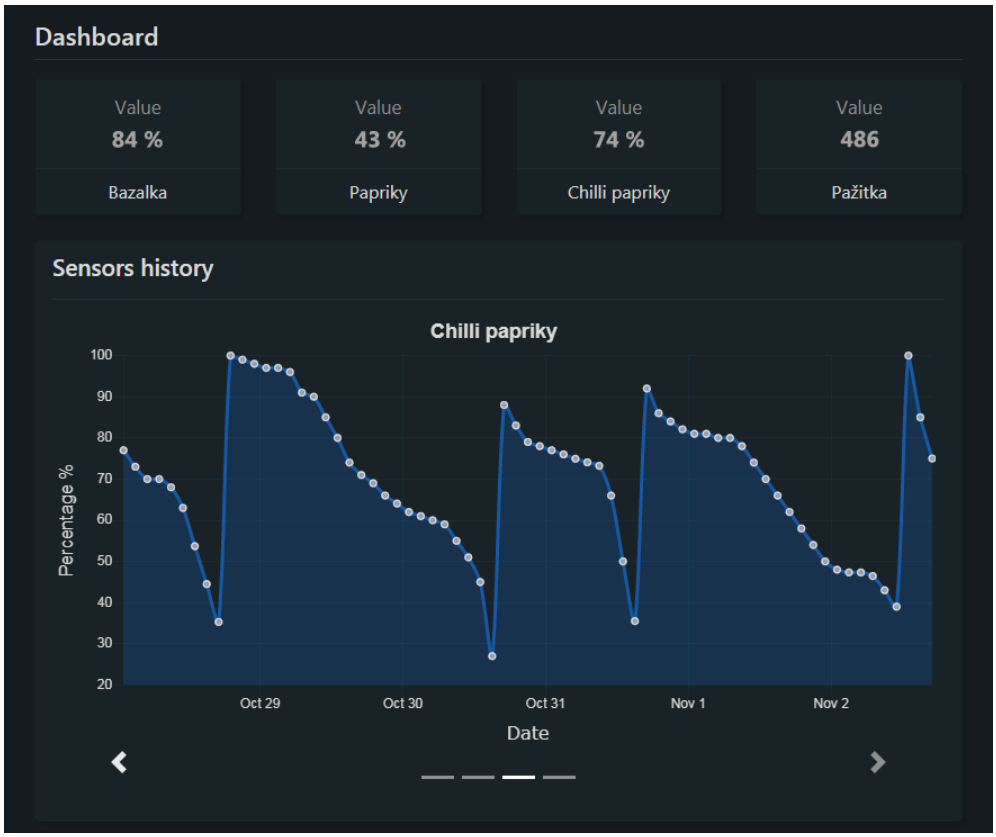
Webová aplikace využívá těchto balíčků:

- Bootstrap – knihovna nástrojů pro vývoj responzivních webů,
- jQuery – knihovna pro tvorbu interaktivních stránek,
- Fontawesome – sada ikon dostupných jako kaskádové styly,
- ngx-bootsrap – upravené verze Bootstrap komponent pro použití v Angularu,
- ngx-cron-editor – formulářový generátor cron výrazů,
- cronstrue – slouží pro převod cron výrazů do textového popisu,
- ngx-toastr – zajišťuje vyskakující notifikace systému,
- ng-toggle – komponenta jednoduchého dvoustavového přepínače,

[17] https://www.npmjs.com

- Chart.js – sada různých typů grafů,
- ng2-charts – Angular direktivy pro knihovnu Chart.js.

Ukázku praktického využití modulu Chart.js v komponentě *Dashboard* obsahuje Obrázek 22.



Obrázek 22 – Ukázka přehledu s grafem

5.3.3 Responzivita

Responzivní rozložení webové aplikace zajišťuje, že se každá stránka správně zobrazí na kterémkoliv běžném zařízení. Pro tento účel aplikace využívá předdefinovaných šablon Bootstrapu a především pak jeho systému zarovnání do mřížky. Tento systém vychází z dřívějších tabulkových rozvržení stránek. Díky tomu zde lze nalézt styl pro řádek a jeho jednotlivé sloupce, kterých může být v jednom řádku až 12. Dynamickou šířku těchto sloupců pak určují předem definované kroky pro různé šířky obrazovek[18].

Dalším použitým prvkem Bootstrapu je navigační lišta, která při zobrazení stránky na malém displeji skrývá menu a nahrazuje ho tlačítkem. Při stisku tlačítka se následně menu opět vysune.

[18] https://getbootstrap.com/docs/4.3/layout/grid/#grid-options

ZÁVĚR

Cíle bakalářské práce byly splněny v plném rozsahu. Úvodní kapitola se zabývala definicí pojmu internetu věcí, jeho historií a zároveň v ní byly představeny stávající technologie pro řízení chytrých domů. Závěr této kapitoly se navíc věnoval problematice zabezpečení chytrých zařízení.

Druhá kapitola se zaměřila na vznik a účel minipočítačů Raspberry Pi. Součástí kapitoly bylo srovnání jednotlivých modelů tohoto minipočítače, a to i s ohledem na jejich spotřebu energie. Dále bylo detailněji popsáno hardwarové vybavení, včetně principu jeho funkce. Nakonec byl uveden také software, který lze na Raspberry Pi provozovat.

Zbývající kapitoly byly věnovány realizaci praktické části bakalářské práce. Ve třetí kapitole byla provedena analýza požadovaných vlastností a samotný návrh zavlažovacího systému. Při rozboru požadavků byly zvoleny vhodné hardwarové komponenty a softwarové technologie. Ty byly následně představeny včetně jejich vlastností a funkcionalit. Předposlední kapitola popisuje backend aplikaci pro systém zavlažování. Je v ní uveden objektový model a implementace funkcí systému. Backend aplikace spravuje jednotlivé senzory, výstupy a umožňuje plánování zavlažování. Systém dále poskytuje aplikační rozhraní pro navazující klientské aplikace. Přístup ke koncovým bodům API vyžaduje bezpečnostní token, který je uživateli poskytnut po přihlášení. Součástí API je dále automatická dokumentace dostupná z webového prohlížeče.

Poslední kapitola je zaměřena na tvorbu frontend aplikace v podobě dynamické webové stránky. S pomocí této aplikace je možné celý systém vzdáleně ovládat. Vytvořená aplikace je plně responzivní, a proto ji lze používat i na mobilních zařízení.

POUŽITÁ LITERATURA

[1] POHANKA, Pavel. Internet věcí. *Pavelpohanka.cz* [online]. ©2017 [cit. 2019-09-19]. Dostupné z: http://www.pavelpohanka.cz/internet-of-things

[2] Internet of Things: Status and implications of an increasingly connected world. *United States Government Accountability Office* [online]. Květen 2017, s. 4 [cit. 2019-09-19]. Dostupné z: https://www.gao.gov/assets/690/684590.pdf

[3] VOJÁČEK, Antonín. Co se skrývá pod výrazy Industry 4.0 / Průmysl 4.0 ? *Automatizace.HW.cz* [online]. 19. 3. 2016 [cit. 2019-09-16]. Dostupné z: https://automatizace.hw.cz/mimochodem/co-je-se-skryva-pod-vyrazy-industry-40-prumysl-40.html

[4] CHIN, Monica a Althea CHANG. Apple HomeKit: What Is It, and How Do You Use It? *Tom's Guide* [online]. 15. 4. 2019 [cit. 2019-09-21]. Dostupné z: https://www.tomsguide.com/us/apple-homekit-faq,review-4195.html

[5] DOLEJŠ, Jan. Google Home: Chytrá domácnost na dobré cestě (recenze). *SvetAndroida.cz* [online]. 30. 1. 2017 [cit. 2019-09-22]. Dostupné z: https://www.svetandroida.cz/google-home-recenze

[6] LUDLOW, David. What are Z-Wave, Zigbee and other smart home protocols? *TrustedReviews.com* [online]. 3. 4. 2018 [cit. 2019-09-22]. Dostupné z: https://www.trustedreviews.com/opinion/z-wave-zigbee-smart-home-protocols-3426057

[7] LOM, Michal a Ondřej PŘIBYL. Rizika chytrých zařízení a jejich zabezpečení. *TZB-info* [online]. 3. 4. 2017 [cit. 2019-09-22]. ISSN 1801-4399. Dostupné z: https://elektro.tzb-info.cz/inteligentni-budovy/15569-rizika-chytrych-zarizeni-a-jejich-zabezpeceni.

[8] The awesome story of Raspberry Pi. *RaspberryTips.com* [online]. ©2019 [cit. 2019-09-24]. Dostupné z: https://raspberrytips.com/raspberry-pi-history

[9] Raspberry Pi FAQs. *RaspberryPi.org* [online]. ©2019 [cit. 2019-09-24]. Dostupné z: https://www.raspberrypi.org/documentation/faqs

[10] The comprehensive GPIO Pinout guide for the Raspberry Pi. pinout.*xyz* [online]. ©2019 [cit. 2019-09-24]. Dostupné z: https://pinout.xyz

[11] TIŠNOVSKÝ, Pavel. Externí sériové sběrnice SPI a I²C. *Root.cz* [online]. 30. 12. 2008 [cit. 2019-09-26]. ISSN 1212-8309. Dostupné z: https://www.root.cz/clanky/externi-seriove-sbernice-spi-a-i2c

[12] About. *Wiringpi.com: GPIO Interface library for the Raspberry Pi* [online]. ©2019 [cit. 2019-09-27]. Dostupné z: http://wiringpi.com

[13] BOLTON, William. *Instrumentation and Control Systems*. Newnes, 2004. ISBN 978-0750664325.

[14] ŠKUTOVÁ, Jolana. *Projektování informačních systémů* [online]. Ostrava: Vysoká škola báňská – Technická univerzita Ostrava, 2011 [cit. 2019-09-28]. ISBN 978-80-248-2766-7. Dostupné z: http://projekty.fs.vsb.cz/147/ucebniopory/978-80-248-2766-7.pdf

[15] LITSCHMANN, Tomáš. Měření půdní vlhkosti. *Informační systém Masarykovy univerzity* [online]. 19. 4. 2010 [cit. 2019-10-01]. Dostupné z: https://is.muni.cz/el/1431/jaro2010/Z0075/um/Prednaska_Dr_Litschmann_PudniVlhkost.pdf

[16] Sonoff SV. *ITEAD Wiki* [online]. ©2019 [cit. 2019-10-12]. Dostupné z: https://www.itead.cc/wiki/Sonoff_SV

[17] CHAUDHARY, Anas. Why To Choose Java. *C-sharpcorner.com* [online]. 7. 1. 2015 [cit. 2019-10-02]. Dostupné z: https://www.c-sharpcorner.com/UploadFile/a608e4/why-choose-java

[18] KUMAR, Pankaj. Spring Framework. *JournalDev.com* [online]. ©2019 [cit. 2019-10-05]. Dostupné z: https://www.journaldev.com/16922/spring-framework

[19] BALANI, Navveen. Introduction To Spring Framework. *Navveenbalani.dev* [online]. ©2019 [cit. 2019-10-04]. Dostupné z: https://navveenbalani.dev/index.php/articles/spring-framework/introduction-to-spring-framework

[20] Reference Documentation: Introduction to Spring Framework. *Spring.io* [online]. ©2019 [cit. 2019-10-06]. Dostupné z: https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/overview.html

[21] The Pi4J Project: Java I/O library for the Raspberry Pi. *Pi4J.com* [online]. ©2019 [cit. 2019-10-02]. Dostupné z: https://pi4j.com/1.2/index.html

[22] Quartz Enterprise Job Scheduler: Overview. *Quartz-scheduler.org* [online]. ©2019 [cit. 2019-10-02]. Dostupné z: http://www.quartz-scheduler.org/overview

[23] ČÁPKA, David. Lekce 1 - MySQL krok za krokem: Úvod do MySQL a příprava prostředí. *Itnetwork.cz* [online]. ©2019 [cit. 2019-10-03]. Dostupné z: https://www.itnetwork.cz/mysql/mysql-tutorial-uvod-a-priprava-prostredi

[24] Angular – Architecture overview. *Angular.io* [online]. ©2019 [cit. 2019-10-03]. Dostupné z: https://angular.io/guide/architecture

[25] HORDĚJČUK, Vojtěch. Maven. *Voho.eu* [online]. ©2019 [cit. 2019-10-09]. Dostupné z: http://voho.eu/wiki/maven

[26] TYSON, Matthew. What is JPA? Introduction to the Java Persistence API: The Java ORM standard for storing, accessing, and managing Java objects in a relational database. *JavaWorld* [online]. 2. 10. 2019 [cit. 2019-10-19]. Dostupné z: https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html

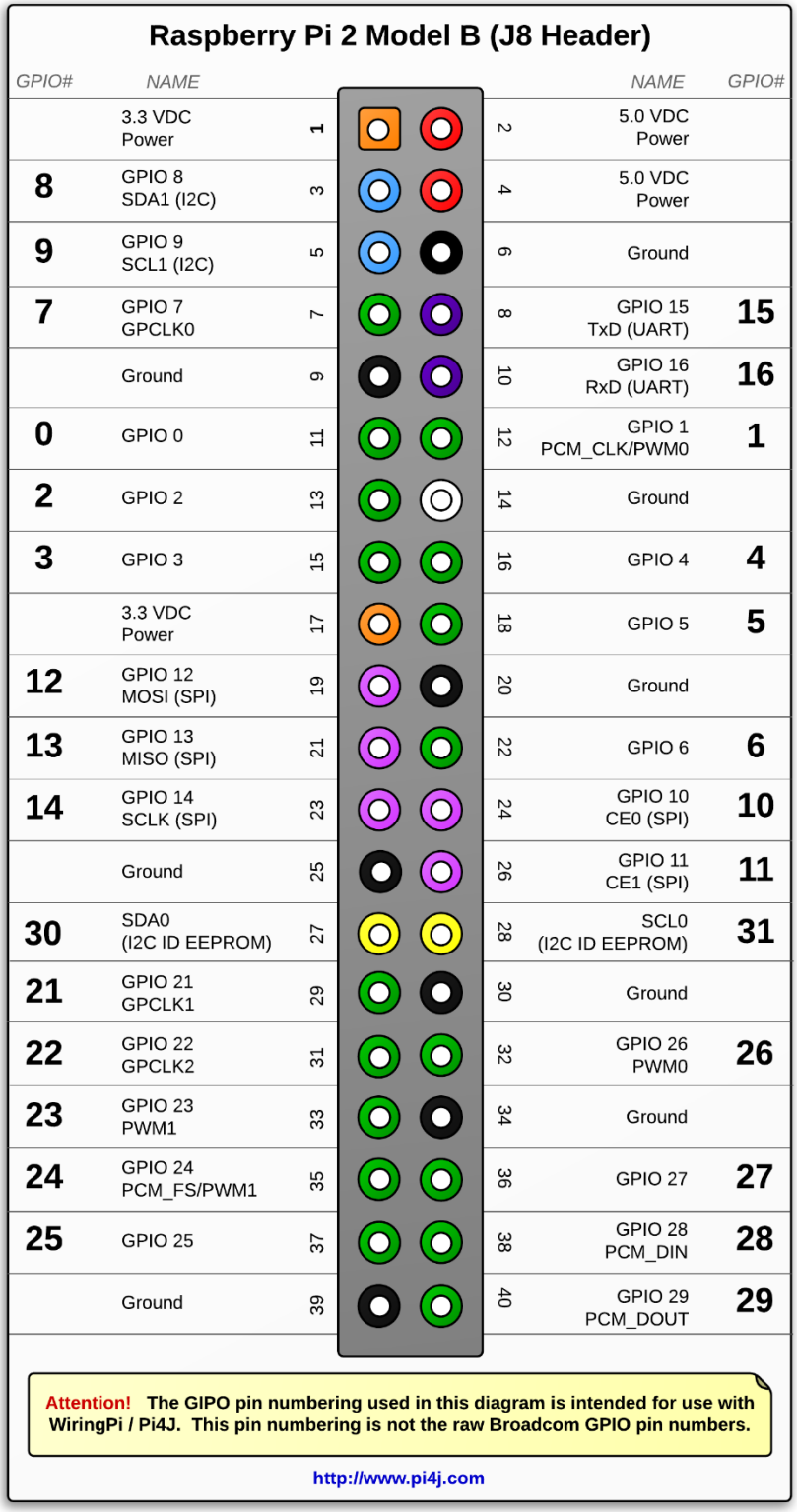
[27] What is REST. *Restfulapi.net* [online]. [cit. 2019-10-20]. Dostupné z: https://restfulapi.net

[28] LE, Loi a Tien NGUYEN. Angular Spring Boot JWT Authentication example | Angular 6 + Spring Security + MySQL Full Stack: Overview and Architecture. *Grokonez.com* [online]. 24. 10. 2018 [cit. 2019-10-26]. Dostupné z: https://grokonez.com/spring-framework/spring-security/angular-spring-boot-jwt-authentication-example-angular-6-spring-security-mysql-full-stack-part-1-overview-and-architecture

[29] WATMORE, Jason. Angular 7 - JWT Authentication Example & Tutorial. *Jasonwatmore.com* [online]. 16. 11. 2018 [cit. 2019-10-30]. Dostupné z: https://jasonwatmore.com/post/2018/11/16/angular-7-jwt-authentication-example-tutorial

PŘÍLOHY

PŘÍLOHA A – ZNAČENÍ GPIO DLE KNIHOVNY PI4J



Obrázek 23 – Značení GPIO rozhraní Raspberry Pi 2 B dle Pi4J [21]