Diplomová práce

České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Sada příkladů k frameworku pro automatizované testování IoT systémů

Bc. Aneta Volfová

Vedoucí práce: doc. Ing. Miroslav Bureš, Ph.D.
Obor: Otevřená informatika, Softwarové inženýrství
Leden 2021

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Volfová** Jméno: **Aneta** Osobní číslo: **435280**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Sada příkladů k frameworku pro automatizované testování IoT systémů

Název diplomové práce anglicky:

A set of examples for a framework for automated testing of IoT systems

Pokyny pro vypracování:

Vytvořte sadu příkladů dokumentujících použití open source frameworku PatrIoT pro automatizované testování IoT systémů. Definujte strukturu tří fiktivních IoT systémů (např. chytrá domácnost nebo inteligentní doprava a pro ně s využitím k tomu určených existujících komponent frameworku vytvořte moduly simulující IoT zařízení (např. různé senzory poskytující data nebo akční členy). Tyto moduly využijte při sestavování příkladů automatizovaných integračních testů pro definované příklady IoT systémů. Navrženou sadu ověřte pomocí vytvoření sady integračních testů ve frameworku PatrIoT používajících dané moduly.

Seznam doporučené literatury:

Ahmed, B. S., Bures, M., Frajtak, K., & Cerny, T. (2019). Aspects of Quality in Internet of Things (IoT) Solutions: A Systematic Mapping Study. IEEE Access, 7, 13758-13780.
Hanes, David, et al. IoT fundamentals: Networking technologies, protocols, and use cases for the internet of things. Cisco Press, 2017.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Miroslav Bureš, Ph.D., laboratoř inteligentního testování systémů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **29.07.2020** Termín odevzdání diplomové práce: **05.01.2021**

Platnost zadání diplomové práce: **19.02.2022**

doc. Ing. Miroslav Bureš, Ph.D.	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce		podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomantka bere na vědomí, že je povinna vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. Podpis studentky

Datum převzetí zadání

Poděkování

Chtěla bych poděkovat především svému vedoucímu diplomové práce doc. Ing. Miroslavu Burešovi, Ph.D. za odborné vedení a za pomoc a rady při zpracování této diplomové práce. Mé díky patří také rodině a příteli, který mě během mé práce podporoval a dodával mi motivaci k práci.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 3. ledna 2021

Abstrakt

V dnešní době jsou IoT aplikace běžně dostupné mezi námi a lze říci, že formují svět kolem nás. Mezi jejich charakteristické základní vlastnosti patří automatizace, nízká energetická náročnost a většinou i bezdrátová technologie. Za účelem dodržení a zachování zmíněných vlastností, je třeba ověřovat jejich funkčnost – tedy testovat. Obsahem mé diplomové práce je popis návrhu a implementace sady příkladů k frameworku PatrIoT pro automatizované testování IoT systémů. V rámci návrhu tří fiktivních systémů jsou vytvořeny testovací případy a zamýšlené integrační testy, které jsou realizovány v testovací části. Obsaženou analýzu doplňuje popis využitých technologií a požadavků. Tato diplomová práce se dále zabývá popisem realizovaných modulů systémů a systémů samotných. Následující část je zaměřena na implementaci návrhu s ukázkami kódu a současně na testování systémů, včetně jejich instalace a spuštění. Závěrečná část je věnována možným vylepšením projektu (frameworku) do budoucna.

Klíčová slova: automatizované testy, IoT, PatrIoT, ukázkové komponenty a zařízení, simulace

Vedoucí práce: doc. Ing. Miroslav Bureš, Ph.D.

Abstract

Today, IoT applications are commonly available among us and can be said they shape the world around us. Their characteristic basic features include automation, low energy consumption and mostly wireless technology. In order to comply with and preserve the mentioned properties, it is necessary to verify their functionality - that is, to test. The content of my diploma thesis is a description of the design and implementation of a set of examples of the PatrIoT framework for automated testing of IoT systems. Within the design of three fictitious systems, test cases and intended integration tests are created and are implemented in the test part. The included analysis is complemented by a description of the technologies used and requirements. This diploma thesis also deals with the description of the implemented modules of systems and the systems themselves. The following section focuses on implementing a design with code samples and at the same time on testing the systems, including their installation and execution. The final part is devoted to possible improvements of the project (framework) in the future.

Keywords: automated tests, IoT, PatrIoT, example components and devices, simulation

Title translation: A set of examples for a framework for automated testing of IoT systems

Obsah

Kapitola 1
Úvod

Téma IoT je v dnešní době často diskutovaným tématem, a to jak v médiích, tak ho lze zaznamenat například i během cesty v MHD a lze konstatovat, že se jedná o populární námět k diskusím. Co je ovšem již opomíjeno a běžnému laikovi skryto, je způsob, jakým je možné IoT systémy vyvíjet a testovat.

IoT systémy mají potenciál stát se součástí každé domácnosti. Mají schopnost sbírat a odesílat data a přijímat informace a následně na ně reagovat. Při zapojení BigData[1] a umělé inteligence můžeme zautomatizovat běžné procesy a celkově zefektivnit naši práci a to v každodenním soukromém i pracovním životě. Využití IoT v kombinaci s kritickými systémy (tj. systémy, jejichž kolaps nebo nespolehlivost může vést v nejzazším případě až k životu ohrožujícím následkům) pak zvyšuje důležitost testování a ověřování spolehlivosti systému.

Hlavním cílem této diplomové práce je navrhnout a implementovat sadu příkladů dokumentujících použití open source frameworku PatrIoT pro automatizované testování IoT systémů, která splňuje všechny požadavky uvedené v kapitole Analýza požadavků 3.1. Sada příkladů je vytvářena ve snaze ukázat řešení problémů souvisejících s testováním IoT systémů.

Nejprve jsou definovány tři fiktivní IoT systémy – konkrétně Chytrý domov, Sklad a Chytré parkování s využitím k tomu určených existujících komponent frameworku a nově implementovaných zařízení, senzorů a akčních členů. Současně pro tyto fiktivní IoT systémy jsou vytvořeny moduly simulující IoT zařízení.

Pro účely vypracování mé diplomové práce byly zjištěny potřebné informace o použitých technologiích, osvojena technologie Apache Camel (viz kapitola 3.1.3), vyhledána a nastudována potřebná literatura a s ohledem na zjištěná fakta bylo navrženo samotné řešení a nakonec provedena implementace včetně testování. Dále byla přepsána část patriot-gateway (modul zajišťující komunikaci se simulovanými systémy), aby využívala jen samotný Apache Camel. Pro názorné vysvětlení problematiky jsou přiloženy ukázky kódu.

Výstupem diplomové práce jsou tři výše zmiňované systémy skládající se celkem ze čtyř modulů – modulu pro generování dat, modulu simulujícího daný systém a jeho zařízení, modulu zajišťujícího komunikaci (tyto moduly

[1] BigData – pojem pro enormní objemy dat, které je obtížné zpracovávat v rozumném čase tradičními nástroji

jsou využity při sestavování příkladů automatizovaných integračních testů) a v neposlední řadě modulu s integračními testy.

Navržená sada je poté ověřena pomocí vytvořených integračních testů ve frameworku PatrIoT používajících dané moduly.

Téma testování mě skutečně zajímá a považuji ho za důležitou součást vývoje projektu a s ohledem na to, že se již téměř 6 let věnuji problematice testování, zvolila jsem si shora uvedené téma. Považuji za přínosné možnost využití svých zkušeností, jež jsem načerpala z praxe vykonáváním práce na pozicích testera a test inženýra juniora v několika společnostech.

Podle mého názoru je testování neoddělitelnou součástí tvorby každé aplikace. Klíčovou úlohu plní hlavně při vývoji aplikací, jejichž nefunkčnost by mohla mít pro okolí negativní až fatální následky. Jako příklad mohu uvést aplikace ve finanční sféře, v průmyslu, ve zdravotnictví atd.

Následující kapitoly jsou rozděleny do šesti celků – na část rešeršní, analytickou, návrhovou, implementační, část popisující testování a integrační testy, instalaci a spuštění projektu a také část Pokračování projektu do budoucna pojednávající o zvažovaných doporučení k projektu do dalšího vývoje.

Rešeršní část je rozborem vědeckých článků a publikací s tématikou testování IoT a popisem frameworku PatrIoT. Analytický oddíl práce je věnován analýze požadavků systému a shrnutí použitých technologií. Návrhový úsek se zabývá popisem tří fiktivních systémů, testovacími případy a zamýšlenými integračními testy. Implementační celek je zaměřen na popis vytváření projektu (systémů) spolu s demonstrací kódu a vysvětlením, jak celý systém funguje. Předposlední kapitola se zaobírá testováním a integračními testy a také instalací a spuštěním systémů a testů. Poslední část ve zkratce prezentuje možná vylepšení, která by mohla být v budoucnu do frameworku implementována nebo doplněna/upravena.

Kapitola 2
Rozbor problematiky

Tato kapitola vysvětluje, co je to IoT a k čemu se používá a ukazuje existující problémy v IoT systémech. Další část popisuje framework PatrIoT a jeho části (moduly).

2.1 Internet of Things

Internet of Things (IoT) neboli „Internet věcí“ je označení pro moderní přístroje ovladatelné na dálku pomocí internetu, která mezi sebou mohou komunikovat právě díky připojení k internetu (nejčastěji k WiFi[1]). Lze je obsluhovat pomocí chytrých telefonů anebo je možné je připojit k ovládací jednotce, která bude řídit všechna zařízení [18]. Nejznámějším příkladem je chytrý domov.

Uživatel tedy může z dovolené na horách ovládat například světla v domě, v místě svého bydliště, aby případní zloději nezjistili, že je dům prázdný. S centrální ovládací jednotkou lze nastavit různá pravidla, kterými se má dům v určitou časovou jednotku řídit.

Oblast IoT je v poslední době často diskutovaná, a to zejména díky možnému dopadu na život ve společnosti. Vize průmyslových společností byla ještě před několika lety brána jako sci-fi. Předpokládá se, že budou v budoucnosti automobily, veškeré budovy i celá města a také běžně dostupné „předměty“ fungovat prostřednictvím připojení k internetu. Toto spojení internetu a „věcí“ by mohlo změnit způsob života lidí [13].

Aplikace a systémy využívající IoT jsou dnes již běžné – ať už v nových chytrých budovách, městech nebo například v oblasti obrany a dokonce i žárovka dnes umožňuje připojení k internetu. S nárůstem množství zařízení připojených k internetu však souvisejí možné problémy, které mohou ovlivnit i nás, náš osobní život, naši bezpečnost a další oblasti našeho života.

Bez nadsázky lze říci, že IoT je rozvíjející se technologie, která má moc změnit naši budoucnost.

[1] WiFi – označení pro bezdrátové propojení různých zařízení (počítač, notebook, tablet) a jejich připojování k počítačové síti

2.2 Existující řešení IoT systémů

V dnešní době je IoT všude kolem nás, i když to na první pohled nemusí být vidět. Tento systém může být využíván například pro provoz budovy, ve které máte své pracoviště. Tato budova si pak sama řídí, v jakých místnostech se bude topit, kdy se bude zapínat klimatizace anebo osvětlení.

V průběhu roku 2020 bylo identifikováno přes 50 miliard zařízení (mobilních telefonů, tabletu, počítačů, ...) připojených k internetu, což odpovídá více než šestinásobku populace lidí na světě [25].

Současné IoT systémy se dají rozdělit na dvě hlavní skupiny:

- průmyslové
- spotřebitelské

2.2.1 Průmyslové systémy

IoT systémy, které spadají do skupiny průmyslových se zaměřují na klíčové úlohy průmyslu. Jsou založeny na komunikaci stroje se strojem a analýze dat. V průmyslových odvětvích se jedná o využití v průmyslové automatizaci (diagnostika přístrojů, sestavování výrobků, ...), energetickém průmyslu (výroba elektřiny, řízení spotřeby energie, . . .) nebo v dopravním průmyslu (řízení křižovatek, železničních tratí, . . .) [19].

Cílem zmíněných průmyslových systémů je zefektivnění využívání zdrojů, zvýšení produktivity a nebo například předcházení výpadkům.

Typickým příkladem průmyslového IoT systému jsou:

- ulice s inteligentním osvětlením - reagují na přítomnost chodců a automobilů a umožňují šetřit energii
- chytré budovy - regulují osvětlení a výměnu vzduchu v budově
- inteligentní silnice - poskytuje informace o nehodách a počasí

Obdobných příkladů je velké množství - proto bylo vybráno jen několik z nich.

2.2.2 Spotřebitelské systémy

Systémy spadající do skupiny spotřebitelských IoT systémů se zaměřují na spotřebitele - na jeho spotřebiče a komunikační zařízení. Jejich cílem je usnadnit a zpříjemnit lidem život.

Zde mluvíme o chytrých zařízeních jako pračky, osvětlení, kávovary, myčky nádobí a další. Spotřebitelské IoT systémy se vyznačují zejména malými toky dat a tím, že nejsou životně důležité [19].

Typickým příkladem spotřebitelského IoT systému jsou:

- chytré domácnosti - ovládání spotřebičů, řízení spotřeby domácnosti
- platby za zboží pomocí NFC[2] - placení pomocí mobilu

2.2.3 Problémy

Mezi problémy a výzvy, se kterými se většina IoT systémů potýká, patří [2]:

1. komunikační protokoly a standardy
2. vysoká variabilita koncových zařízení
3. aktualizace
4. snižování výrobních nákladů a kompatibilita
5. testování

Komunikační protokoly a standardy

Existuje rozsáhlá sada komunikačních protokolů a standardů, které se v dnešních IoT systémech využívají [2]. Jako příklad lze uvést MQTT[3], což je protokol pro posílání zpráv v IoT. Takových protokolů je velké množství a každý IoT systém využívá jiný, proto je problém zaručit fungování několika systémů dohromady. Je vysoká pravděpodobnost, že bude každý z nich používat jiný standard.

Pro porovnání: V případě webových stránek (aplikací/systémů) si každý hned představí HTTP[4].

Vysoká variabilita koncových zařízení

Dalším problémem je velké množství typů a variant koncových zařízení [2]. Může jít o světlo na ovládání přes internet, světlo se senzorem, světlo s časovačem spínání a další varianty. Všechna tato zařízení spolu musí komunikovat a podporovat stejné protokoly, pokud jsou zapojeny do jednoho IoT systému.

[2]NFC - technologie rádiové bezdrátové komunikace mezi zařízeními na krátkou vzdálenost
[3]MQTT - protokol, který umožňuje přenos zpráv mezi zařízeními
[4]HTTP - internetový protokol určený pro komunikaci s WWW (World Wide Web)

Aktualizace

Koncová zařízení bývají také napájena z různých zdrojů – z baterie, solární energie anebo třeba klasicky ze sítě. V rámci omezování spotřeby energie těchto zařízení se přistupuje k využívání nenáročných algoritmů (tj. algoritmů, které nepotřebují mnoho náročných výpočtů a tudíž nemají velkou spotřebu energie) [2]. Fakt, že se využívají právě odlehčené a nenáročné algoritmy snižuje možnost aktualizace zařízení.

Snižování výrobních nákladů a kompatibilita

Z důvodu konkurenčního tržního prostředí je snaha snížit výrobní náklady zařízení [2]. Tím vznikají produkty a zařízení v různých verzích. Novější verze pak často nemusí být kompatibilní se starší verzí. Zde lze uvést jako příklad problém s kamerovým systémem a kódováním. Ve starších systémech bývalo kódování H264[5], které fungovalo vcelku dobře. Novější kamery však podporují kódování H265[6], které není zpětně kompatibilní s H264 kódováním. Pokud má tedy uživatel kamerový systém staršího typu, tak si nemůže přikoupit kameru novou, neboť by nastaly zmíněné potíže s kompatibilitou.

Testování

Důležitou součástí je i testování. Pokud máme několik verzí zařízení (například kamer), tak musíme klást větší důraz na integrační testování a tím pádem i více finančně investovat [2]. Zvyšují se nám tím i nároky na testovací prostředí, protože musíme mít všechna fyzická zařízení k dispozici, musíme je udržovat v chodu a řešit jejich napájení.

2.2.4 Řešení

Jedním z možných řešení výše uvedených problémů je využití simulace [2]. Na trhu můžeme najít různé typy simulátorů IoT systémů, které bývají často placené. Mezi ně patří například:

- FS-IIoTSim [5] - simulátor sítě pro hodnocení výkonu IIoT[7]
- Smart Home Network Simulation Testbed for Cybersecurity Experimentation – simulátor pro testování zabezpečení systémů [4]
- NetSim[8] – simulátor sítě od firmy Netcos, placený

[5] H264 - kompresní formát obrazu (kódování)
[6] H265 - nastupující verze po H264
[7] IIoT - průmyslové IoT, propojení senzorů a zařízení s počítači v průmyslu
[8] NetSim - oficiální stránky dostupné na WWW: <https://www.tetcos.com/>

- OMNeT++[9] - jde o knihovnu pro vytváření simulátorů sítě v C++

Výhodou simulátorů je snížení potřeby fyzických zařízení a jejich udržování v provozu. Pro ověření integrace simulovaných zařízení se využívají integrační testy, které můžeme rozdělit na 2 typy:

1. E2E integrační testy
2. Unit integrační testy

E2E integrační testy

Testy typu E2E jsou testy, které testují v podstatě na principu black-box, čili uživatel zadá příkaz a čeká, co se mu nakonec vrátí za výsledek, aniž by věděl, jak celý systém funguje. Na podobném principu fungují E2E integrační testy s rozdílem, že jde o testování mezi různými subsystémy a zjišťování, zda spolu tyto systémy správně komunikují a spolupracují [6].

Představme si, že jedním z testovaných systémů je robot, který v ruce drží deštník. Senzor na hlavě robota dostane informaci, že začíná pršet a jeho reakce je stisknutí tlačítka deštníku s následným otevřením deštníku. V tomto případě by bylo možné vytvořit E2E integrační test, který by ověřoval, zda v případě předání informace robotovi o začínajícím dešti je vyslán signál ruce, která stiskne tlačítko deštníku a deštník se otevře. Pokud by se deštník otevřel, tak by byl náš E2E integrační test úspěšný.

Unit integrační testy

Mezi další z možností testování simulací IoT systémů patří Unit integrační testování. Jak už název Unit (v překladu jednotka) napovídá – jedná se o testování malých jednotek v systému [6]. Zjednodušeně lze z několika Unit integračních testů složit jeden E2E integrační test.

Příkladem jednoduchého Unit integračního testu by mohlo být zaslání požadavku robotovi, aby zvedl ruku. Test by byl úspěšný, pokud by byla ruka zvednuta.

2.3 PatrIoT framework

PatrIoT framework je framework (softwarové řešení) vyvíjený firmou Red Hat[10] ve spolupráci se System Testing IntelLigent Lab (STILL)[11] na Fakultě elektrotechnické ČVUT. PatrIoT umožňuje simulovat jednotlivé komponenty,

[9] OMNeT++ - oficiální stránky dostupné na WWW: <https://omnetpp.org/>

[10] Red Hat - společnost zabývající se vývojem informačních technologií, viz WWW: <https://www.redhat.com/en>

[11] STILL - laboratoř se sídlem na Katedře počítačů FEL ČVUT, viz WWW: <http://still.felk.cvut.cz/>

zařízení nebo senzory a v souvislosti s tím testovat distribuované IoT aplikace [7].

Integrační testování IoT aplikací přináší velké množství výzev a zároveň problémů, které se PatrIoT snaží vyřešit a poskytuje možnost simulace komponent, síťové virtualizace[12], simulace a generování dat anebo interakce hardwarových zařízení [7]. Všechno vyjmenované je spojené do jednoho frameworku.

Tento framework má několik modulů (částí), kde každá z částí plní jiný úkol. Jedná se o moduly:

- patriot-data-generator - modul sloužící ke generování dat. Umožňuje vytvářet data (například pro senzor) na základě volby konkrétních distribucí dat[13] (lineární, normálová, konstantní, ...) anebo si lze implementovat vlastní distribuce dat.
- patriot-api – modul definující základní API[14], které umožňuje integraci ostatních modulů a technologií.
- patriot-network - modul sloužící jako backend[15] určený k simulování sítě a nasazení jednotlivých částí frameworku na simulované prostředí.
- patriot-router – součást modulu patriot-network, který simuluje chování směrovače (router)[16] v síti.
- patriot-virtual-smart-home – modul simulující zařízení nebo senzory, které mohou získávat data z patriot-data-generator. Komponenty simulují reálná zařízení a jejich chování.
- patriot-gateway – modul sloužící jako „gateway“ (brána)[17] mezi patriot-virtual-smart-home a okolním prostředím. Jsou zde také popsána pravidla komunikace s patriot-virtual-smart-home.
- patriot-integration-tests – poslední z modulů sloužící k testování, resp. k tvorbě integračních testů pro patriot-gateway.

Praktická část diplomové práce se primárně věnuje čtyřem modulům z PatrIoT frameworku. Jedná se o moduly:

1. patriot-data-generator
2. patriot-virtual-smart-home

[12] Síťová virtualizace - proces kombinace hardwarových a softwarových zařízení v jednu entitu (síť)

[13] Distribuce dat (pravděpodobnosti) - pravidlo, kterým se každé hodnotě (jevu) přiřazuje určitá pravděpodobnost výskytu

[14] API - soubor procedur, funkcí a protokolů využívaných programátory při tvorbě aplikace

[15] Backend - část, která určuje, co bude daný program (aplikace) - dělat, tato část není vidět

[16] Router - směrovač, síťové zařízení, které přeposílá data správnému adresátovi

[17] Gateway - brána propojující dvě sítě, může fungovat i jako router

3. patriot-gateway
4. patriot-integration-tests

Tyto moduly jsou využity v každém ze tří navrhovaných fiktivních systémů – Chytrý domov, Sklad, Chytré parkování. Fiktivní systémy jsou dále popsány v kapitole 4.

Kapitola 3
Analytická část

Část analytická je zaměřena na rozbor požadavků a dále blíže rozvádí využité technologie v rámci tohoto projektu.

3.1 Analýza požadavků

V této kapitole jsou popsány a rozvedeny požadavky na systém, které byly identifikovány během práce na semestrálním projektu, vyplývající ze zadání a z konzultací s vedoucím práce doc. Ing. Miroslavem Burešem, Ph.D. a ve spolupráci s týmem z Red Hat – jmenovitě Ing. Miroslav Jaroš a Ing. Jakub Smolár.

Požadavky jsou rozděleny do dvou částí – na funkční (tj. co všechno musí framework umět, a které funkce budou podporovány) a nefunkční (tj. požadavky na design, použité technologie a kvalitu).

3.1.1 Funkční požadavky

Mezi funkční požadavky, které byly analyzovány, patří:

- Využití přepínání DataFeed[1] - Pro nasimulování hodnot senzorů bude framework využívat přepínání DataFeed (tj. setDataFeed() metodu), které se zavolá při dotazu (requestu) na danou nastavovací URL[2] daného senzoru.
- Implementace času – V modulu patriot-gateway bude možné nastavit si čas, aby bylo možné čas upravovat pro potřeby testů.
- Ukládání timestamp[3] – Framework bude ukládat timestamp poslední zaznamenané hodnoty (například, kdy byl naposledy zaznamenán pohyb).

[1] DataFeed - distribuce dat definovaná v patriot-data-generator
[2] URL - označuje jednotnou adresu pro dohledání webových stránek na internetu
[3] Timestamp - sekvence znaků označující dobu vzniku nějaké události

- Hit endpoint – Senzor pohybu při zaznamenání hodnoty (například pohybu) pošle zprávu na zvolený endpoint[4].
- Držení si stavu - Patriot-gateway si bude držet stav (kupříkladu, zda je/není rozsvíceno).
- Využít Exchange[5] objekty – Při posílání zpráv přes routes budou moduly využívat Exchange objekty.

3.1.2 Nefunkční požadavky

Požadavky, které patří mezi nefunkční, jsou:

- Přepis patriot-gateway – Přepsat patriot-gateway modul z Drools[6] do Apache Camel[7].
- Implementace v Java 8 – Implementovat všechny moduly v Java ve verzi 8.
- Zachování funkcionality patriot-gateway – Pokud to lze, tak zachovat stejnou funkcionalitu modulu při přepisování do Apache Camel.
- Využití stavových automatů – Při implementaci zvolit několik vhodných zařízení (komponent) a ty implementovat jako stavové automaty z patriot-data-generator.
- Vytvoření integračních testů – Navržené moduly budou otestovány pomocí integračních testů.

3.1.3 Použité technologie

V následujících podkapitolách jsou popsány technologie (resp. software) použité při implementaci praktické části této diplomové práce.

Apache Maven

Apache Maven je nástroj určený k řízení softwarových projektů založený na konceptu Project Object Model (POM)[8]. Maven umí spravovat sestavování projektů nebo třeba reportování. Maven umožňuje snadno vyvíjet projekt bez nutnosti zabřednutí do velkých detailů [8].

[4]Endpoint - koncový bod, příkladem může být i URL adresa

[5]Exchange objekty - v těchto objektech se posílají zprávy mezi jednotlivými komponentami

[6]Drools - technologie pro vytváření business pravidel fungování systému - viz WWW: <https://www.baeldung.com/drools>

[7]Apache Camel - framework sloužící k integraci mezi systémy - viz kapitola 3.1.3

[8]POM - soubor obsahující informace o projektu a detaily jeho nastavení včetně jeho závislostí

Apache Camel

Apache Camel [9] je Open Source[9] framework, který slouží k integraci různých systémů, které konzumují nebo produkují nějaká data. Apache Camel umožňuje vytváření „routes“ (cest, směřování) a pravidel v jazycích založených na Java (jako třeba Spring[10]). Umožňuje pracovat přímo s URL[11] nebo s posíláním zpráv přes protokoly jako HTTP[12].

RestAssured

RestAssured [20] je framework určený k testování a validaci aplikací založených na REST[13]. Jeho knihovna podporuje HTTP a jeho operace, JSON[14] a k ověření správných výsledků využívá Hamcrest[15].

[9] Open Source - software s otevřeným (volně přístupným) zdrojovým kódem
[10] Spring - framework v Java pro vývoj aplikací
[11] URL - označuje jednotnou adresu pro dohledání webových stránek na internetu
[12] HTTP - internetový protokol určený pro komunikaci s WWW (World Wide Web)
[13] REST - styl architektury aplikace definující podmínky a pravidla pro fungovaní aplikace na webu
[14] JSON - syntaxe pro ukládání a posílání zpráv (textů)
[15] Hamcrest - framework pro psaní testů v jazyku Java, viz WWW: <https://github.com/hamcrest/JavaHamcrest>

Kapitola 4
Návrhová část

V rámci této diplomové práce byly navrženy celkem tři systémy:

1. Chytrý domov
2. Sklad
3. Chytré parkování

Následující kapitoly popisují, jaké komponenty/zařízení byly vytvořeny a navrženy. Dále jsou zde rozvedeny testovací scénáře a jejich varianty, které jsou použity k otestování systémů.

4.1 Chytrý domov

Takovýto typ chytrého domova by se měl chovat jako reálný domek a měl by disponovat základními senzory a zařízeními, které jsou běžně dostupné v domácnostech. Domov tedy odpovídá jedné budově (stavbě) s nájemníky, který umožňuje volbu například intenzity osvětlení anebo složitější pravidla, jako je vyvětrání, zapnutí kávovaru nebo vytažení žaluzií v určitý čas před plánovaným probuzením.

Patriot-virtual-smart-home je modul, který simuluje virtuální zařízení, akční členy a senzory. Tento modul frameworku je určen především pro simulaci a testování chování komponent a různých zařízení a sledování interakce mezi nimi.

Některé z komponent již byly implementovány, ale musely být aktualizovány z důvodu nových změn vývojáře patriot-data-generator, který modul generátoru upravoval během celého roku. Mým úkolem bylo tedy i reagovat na změny a domek jim přizpůsobit.

Modul virtual-smart-home se skládá z částí:

- House - zde jsou definovány všechny senzory, akční členy a zařízení.

- Routes - část, ve které se nastavují linky (routes) a propojení (nebo trasy) a způsob, kam se přeposílají informace. Tato část komunikuje spolu s patriot-gateway pro Chytrý domov.

Zařízení, akční členy a senzory, které byly připraveny k použití, jsou:

- AC - klimatizace
- DHT11Device - zařízení pro měření teploty a vlhkosti
- Door - dveře a okna
- Fireplace - elektrický krb
- Hygrometer – měří vlhkost (senzor)
- RGBLight - ovládá všechna světla ve virtuálním domě
- Thermometer - měří teplotu (čidlo)
- TV – televize

Nově implementované komponenty:

- CO2Sensor - senzor pro měření koncentrace CO_2
- COSensor - senzor pro měření koncentrace CO
- CoffeeMaker - zařízení kávovar využívající StateMachine (stavový automat)
- EntranceGate - hlavní brána pro automobily
- FireSensor - senzor
- MotionSensor – senzor pohybu
- LightLevelSensor – senzor úrovně osvětlení
- SoundSystem - rádio
- Sunblinds – žaluzie využívající StateMachine (stavový automat)

4.2 Sklad

Sklad si lze představit jako skupinu budov (obchodů, skladů), které pracují s balíčky a kam přijíždějí dodatavelé nebo jiní dopravci. Při příjezdu dodavatele (dopravce) může být navržena možnost kontroly jeho povolení k vjezdu anebo lze přijímat dodávku zásilek bez provedení kontroly a přímo určovat místo jejích uložení.

Modul Sklad (neboli patriot-logistic-chain) byl vytvořen jako druhý systém s využitím dokončeného modulu patriot-virtual-smart-home. Pro potřeby simulace byly některé komponenty ponechány a také připsány nové.

Modul logistic-chain se skládá z částí:

- Warehouse - zde jsou definovány všechny senzory, akční členy a zařízení.
- Routes - část, ve které se nastavují linky (routes) a propojení (nebo trasy) a způsob, kam se přeposílají informace. Tato část komunikuje spolu s patriot-gateway pro Sklad.

Zařízení, akční členy a senzory, které byly ponechány z předchozího modulu, jsou:

- AC - klimatizace
- DHT11Device - zařízení pro měření teploty a vlhkosti
- Door - dveře a okna
- Hygrometer – měří vlhkost (senzor)
- Thermometer - měří teplotu (čidlo)
- CO2Sensor - senzor pro měření koncentrace CO_2
- COSensor - senzor pro měření koncentrace CO
- EntranceGate - hlavní brána pro dodávky
- MotionSensor – senzor pohybu

Nově implementované komponenty:

- DispensingPointPackage – část simulující výdejní místo, kde lze přijímat a vyzvedávat balíčky
- Shop – obchod přijímající objednávky

- VanTruck – zařízení zjišťující povolení k vjezdu dodávek a nákladních aut
- WarehousePackage – sklad přijímající zásilky a určující, kam se mají uskladnit

4.3 Chytré parkování

Chytré parkování si lze představit jako budovu nebo parkovací areál, kam je automobil vpuštěn za základě předchozí rezervace. V případě, že automobil při příjezdu nemá rezervaci, lze mu tuto rezervaci dodatečně vytvořit, a to v případě, pokud je ještě nějaké místo k parkování volné.

Automobily mají na starosti registraci do systému Chytrého parkování, kterou se prokážou při příjezdu a rovněž mají na starosti také placení za parkování.

Modul Chytrého parkování (neboli patriot-smart-parking) byl vytvořen jako třetí systém s využitím dokončeného modulu patriot-virtual-smart-home. Pro potřeby simulace byly některé komponenty ponechány a také připsány nové.

Modul smart-parking se skládá z částí:

- Parking - zde jsou definovány všechny senzory, akční členy a zařízení.
- Routes - část, ve které se nastavují linky (routes) a propojení (nebo trasy) a způsob, kam se přeposílají informace. Tato část komunikuje spolu s patriot-gateway pro Chytré parkování.

Zařízení, akční členy a senzory, které byly ponechány z předchozího modulu, jsou:

- AC - klimatizace
- DHT11Device - zařízení pro měření teploty a vlhkosti
- Door - dveře a okna
- Hygrometer – měří vlhkost (senzor)
- Thermometer - měří teplotu (čidlo)
- CO2Sensor - senzor pro měření koncentrace CO_2
- COSensor - senzor pro měření koncentrace CO
- EntranceGate - hlavní brána pro dodávky
- MotionSensor – senzor pohybu

Nově implementované komponenty:

- Cars - část mající na starost rezervaci a seznam aut a informaci o době jejich příjezdu a placení
- ParkingPlace - třída představující parkovací místa, včetně hlídání stavu parkoviště a informační tabule

4.4 Testovací případy

Tato kapitola popisuje navržené testovací scénáře a jejich varianty, na základě kterých budou příklady tří fiktivních systémů předvedeny.

4.4.1 Případy pro Chytrý domov

Noční vstávání

Tento testovací případ ověřuje integraci mezi patriot-gateway a patriot-virtual-smart-home ve chvíli, kdy senzor pohybu zachytí informaci o pohybu. Jinak řečeno senzor zjistí, že se někdo v noci probudil. Chytrý domov tedy rozsvítí světla na nejpravděpodobnější cestě – cestě k toaletě a do kuchyně.

1. Senzor zachytí pohyb ze snímače pohybu (uživatel se probouzí v noci).
2. Zapnou se světla na cestě do koupelny a kuchyně (tlumené světlo).
3. Pokud do 40 sekund nedojde k žádnému dalšímu „pohybu“, tak se zhasnou světla, pokud ano – světla zůstanou zapnutá dalších 40 sekund. Každý pohyb vynuluje časovač počítající dalších 40 sekund.
4. Uživatel se vrací zpět do postele. Pokud nikdo jiný není vzhůru, pak světla zhasnou. Pokud dojde k opětovnému pohybu po více než 40 sekundách, pokračuje se krokem č. 2.

Konkrétní testovací situace jsou popsány v části Varianty testů.

Předpoklady: Musí být nastavený čas v patriot-gateway v rozmezí od 23:00 do 6:00.

Očekávané výsledky: Světla jsou zapnutá po dobu 40 sekund od posledního zachyceného pohybu. Každý pohyb resetuje časovač. Pokud není zachycen

žádný další pohyb, tak se světla po 40 sekundách vypnou.

Varianty testů:

Varianta 1: wakeUpAtNightFastReturn

1. Senzor pohybu detekuje pohyb (2 pohyby celkem v krátkém intervalu) a zapne světla.
2. Uživatel se rychle vrátí do postele – vypneme simulování pohybu.
3. Světla svítí po 40 sekund od druhého zachyceného pohybu.
4. Test zkontroluje, že jsou světla vypnutá.

Varianta 2: wakeUpAtNightReturnAfterLongTime

1. Senzor pohybu detekuje pohyb a zapnou se světla.
2. Žádný další pohyb není během 40 sekund zachycen – světla se tedy vypnou.
3. Uživatel se vrací do postele (po celkem 60 sekundách) z koupelny – pohyb je opět zachycen, takže se rozsvítí světla.
4. Zastavíme generování pohybu.
5. Po 40 sekundách zkontrolujeme, že jsou světla zhasnutá.

Varianta 3: wakeUpAtNightWholeFamily

1. Senzor pohybu detekuje pohyb a zapnou se světla.
2. Po 12 vteřinách je zachycen další pohyb.
3. Po 12 vteřinách je zachycen třetí pohyb.
4. Po 12 vteřinách je zachycen čtvrtý pohyb.
5. Světla jsou zapnutá a časovač počítá 40 sekund od čtvrtého pohybu.
6. Po 40 sekundách jsou světla opět zhasnutá.

Ranní vstávání

Tento testovací případ ověřuje integraci mezi patriot-gateway a patriot-virtual-smart-home ve chvíli, kdy senzor pohybu zachytí informaci o pohybu

(vstávání v ranních hodinách). V reakci na probuzení se otevřou okna (záleží na vlhkosti vzduchu) po určitý čas. Dále se zapne rádio, světla (záleží na úrovni osvětlení) a kávovar.

1. Senzor pohybu zachytí pohyb (vstávání).
2. Otevřou se okna (záleží na vlhkosti vzduchu) a zůstanou otevřená po určitou dobu.
3. Zapne se rádio.
4. Zapnou se světla (záleží na úrovni osvětlení).
5. Zapne se kávovar.

Konkrétní testovací situace jsou popsány v části Varianty testů.

Předpoklady: Musí být nastavený čas v patriot-gateway na hodnotu větší než 6:00.

Očekávané výsledky: Rádio je zapnuté, kávovar je zapnutý. Světla jsou zapnutá/vypnutá (záleží na hodnotách ze senzoru), okna jsou otevřená po určitý časový úsek anebo jsou zavřená (záleží na variantě testu).

Varianty testů:

Varianta 1: wakeUpHighHumidityNoLights

1. Pohyb je zachycen.
2. Nasimulujeme vysokou vlhkost, aby se otevřela okna na 30 sekund (75 %).
3. Nasimulujeme normální hodnotu osvětlení (1500 lm).
4. Čekáme 20 sekund.
5. Zkontrolujeme, že jsou otevřená okna.
6. Zkontrolujeme, že je rádio zapnuté.
7. Zkontrolujeme, že je kávovar zapnutý.
8. Zkontrolujeme, že jsou světla vypnutá.
9. Čekáme 40 vteřin.
10. Zkontrolujeme, že jsou okna zavřená a světla vypnutá.

Varianta 2: wakeUpHighHumidityAndLightsOn

1. Pohyb je zachycen.
2. Nasimulujeme vysokou vlhkost, aby se otevřela okna na 30 sekund (75 %).
3. Nasimulujeme nízkou hodnotu osvětlení (500 lm).
4. Čekáme 20 sekund.
5. Zkontrolujeme, že jsou otevřená okna.
6. Zkontrolujeme, že je rádio zapnuté.
7. Zkontrolujeme, že je kávovar zapnutý.
8. Zkontrolujeme, že jsou světla zapnutá.
9. Čekáme 40 vteřin.
10. Zkontrolujeme, že jsou okna zavřená a světla zapnutá.

Varianta 3: wakeUpNormalHumidityNoLights

1. Pohyb je zachycen.
2. Nasimulujeme nízkou vlhkost (35 %).
3. Nasimulujeme normální hodnotu osvětlení (1500 lm).
4. Čekáme 20 sekund.
5. Zkontrolujeme, že jsou zavřená okna.
6. Zkontrolujeme, že je rádio zapnuté.
7. Zkontrolujeme, že je kávovar zapnutý.
8. Zkontrolujeme, že jsou světla vypnutá.
9. Čekáme 40 vteřin.
10. Zkontrolujeme, že jsou okna zavřená a světla vypnutá.

Ranní rutiny

Tento testovací případ ověřuje integraci mezi patriot-gateway a patriot-virtual-smart-home ve chvíli, kdy dostane domek informaci, že bude zvonit budík za 1 minutu.

1. Uživatel se vzbudí – budík.

2. Okna se otevřou (záleží na vlhkosti a koncentraci CO_2).
3. Žaluzie se vytáhnou nahoru.
4. Kávovar se zapne.
5. Světla se zapnou (záleží na úrovni osvětlení).

Konkrétní testovací situace jsou popsány v části Varianty testů.

Předpoklady: Musí být nastavený čas v patriot-gateway – například na 8:30 a čas buzení například na 8:32.

Očekávané výsledky: Všechna zařízení jsou zapnutá přesně 1 minutu před budíkem. Po budíku je zapnuté rádio, kávovar, světla jsou zapnutá/vypnutá (záleží na variantě testu), okna jsou otevřená po nějaký čas anebo jsou zavřená (záleží na konkrétní variantě testu).

Varianty testů:

Varianta 1: wakeUpAlarmClockOpenWindowsNoLights

1. Čekání 40 sekund.
2. Kontrola, že jsou okna zavřená.
3. Kontrola, že je rádio vypnuté.
4. Čekání 40 sekund.
5. Nasimulujeme vysokou vlhkost (75 %), aby se otevřela okna na 30 sekund a nízkou koncentraci CO_2 (800 ppm).
6. Nasimulujeme normální hodnotu osvětlení (1600 lm).
7. Kontrola, že jsou okna otevřená.
8. Kontrola, že jsou žaluzie vytažené nahoru.
9. Kontrola, že se spustil kávovar.
10. Kontrola, že jsou světla vypnutá.
11. Čekání několik sekund (40).
12. Kontrola oken. Okna by měla být po tomto čase již zavřená.

Varianta 2: wakeUpAlarmClockNoOpenWindowsNoLights

1. Čekání 40 sekund.

2. Kontrola, že jsou okna zavřená. Okna se otevřou až 1 minutu před budíkem.
3. Kontrola, že je rádio vypnuté.
4. Čekání 40 sekund.
5. Nasimulujeme nízkou vlhkost (45 %) a nízkou koncentraci CO_2 (800 ppm).
6. Nasimulujeme normální hodnotu osvětlení (1600 lm).
7. Kontrola, že jsou okna zavřená.
8. Kontrola, že jsou žaluzie vytažené nahoru.
9. Kontrola, že se spustil kávovar.
10. Kontrola, že jsou světla vypnutá.
11. Čekání několik sekund (40).
12. Kontrola oken. Okna by měla být po tomto čase stále zavřená.

Varianta 3: wakeUpAlarmClockNoOpenWindowsLightsOn

1. Čekání 40 sekund.
2. Kontrola, že jsou okna zavřená.
3. Kontrola, že je rádio vypnuté.
4. Čekání 40 sekund.
5. Nasimulujeme nízkou vlhkost (45 %) a nízkou koncentraci CO_2 (800 ppm).
6. Nasimulujeme nízkou hodnotu osvětlení (500 lm).
7. Kontrola, že jsou okna zavřená.
8. Kontrola, že jsou žaluzie vytažené nahoru.
9. Kontrola, že se spustil kávovar.
10. Kontrola, že jsou světla zapnutá.
11. Čekání několik sekund (40).
12. Kontrola oken. Okna by měla být po tomto čase stále zavřená.
13. Kontrola, že jsou světla stále zapnutá.

Návrat domů autem

Tento testovací případ ověřuje integraci mezi patriot-gateway a patriot-virtual-smart-home ve chvíli, kdy dostane domek informaci od auta s otiskem řetězce (hash), že auto právě dorazilo.

1. Auto odešle zprávu (požadavek), aby se otevřela brána.
2. Brána se otevře/neotevře (záleží na správnosti otisku).
3. Vchodové dveře se otevřou/neotevřou – musí být správný otisk.
4. Světla se zapnou (záleží na úrovni osvětlení).
5. Klimatizace se zapne.
6. Dveře se zavřou – po nějakém časovém úseku.

Konkrétní testovací situace jsou popsány v části Varianty testů.

Předpoklady: Musí být nastavený čas v patriot-gateway a také zde musí být přiřazena auta, která mají povolený vjezd k domu.

Očekávané výsledky: Pokud přijde správný otisk (zpráva), tak se otevře brána u vjezdu k domu a dům začne kontrolovat teplotu a senzor osvětlení. Pokud je otisk nesprávný, pak se nic neotevře a nespustí.

Varianty testů:

Varianta 1: homecomingWithCorrectCarNoLights

1. Odeslání správného otisku auta.
2. Kontrola, že se brána otevřela.
3. Čekání 20 sekund.
4. Kontrola, že se dveře otevřely.
5. Čekání 20 sekund.
6. Nasimulování normální úrovně osvětlení (1600 lm) a normální teploty (22 stupňů Celsia).
7. Kontrola, že jsou světla vypnutá.
8. Kontrola, že je klimatizace vypnutá.
9. Čekání 40 sekund.

10. Kontrola, že jsou dveře zavřené.

Varianta 2: homecomingIncorrectCar

1. Odeslání nesprávného otisku auta.
2. Kontrola, že se brána neotevřela.
3. Čekání 20 sekund.
4. Kontrola, že se dveře neotevřely.
5. Čekání 20 sekund.
6. Kontrola, že jsou světla vypnutá.
7. Kontrola, že je klimatizace vypnutá.
8. Čekání 40 sekund.
9. Kontrola, že jsou dveře zavřené.

Varianta 3: homecomingWithCorrectCarLightsOn

1. Odeslání správného otisku auta.
2. Kontrola, že se brána otevřela.
3. Čekání 20 sekund.
4. Kontrola, že se dveře otevřely.
5. Čekání 20 sekund.
6. Nasimulování nízké úrovně osvětlení (500 lm) a normální teploty (22 stupňů Celsia).
7. Kontrola, že jsou světla zapnutá.
8. Kontrola, že je klimatizace vypnutá.
9. Čekání 40 sekund.
10. Kontrola, že jsou dveře zavřené.

Romantická nálada

Tento testovací případ ověřuje integraci mezi patriot-gateway a patriot-virtual-smart-home ve chvíli, kdy dostane domek informaci, že se blíží čas, kdy má partner dorazit na rande.

1. Jednu minutu před časem příchodu partnera jsou spuštěny následující akce:
2. Rádio se spustí a zvolí se romantická hudba.
3. Pokud je večer, tak se rozsvítí tlumená světla. Pokud je den, tak se zatáhnou žaluzie a rozsvítí se tlumeně světla.
4. Krb se zapne.
5. Čekání jednu minutu.
6. Dveře se otevřou.

Konkrétní testovací situace jsou popsány v části Varianty testů.

Předpoklady: Musí být nastavený čas v patriot-gateway – například na 18:28 a nastaven čas, kdy má partner dorazit – například 18:30.

Očekávané výsledky: Pokud je normální úroveň osvětlení, tak jsou stažené žaluzie, aby v domě byla tma. Pokud ne, tak jsou žaluzie vytažené nahoru. Světla jsou nastavena do romantického módu a v rádiu hraje romantická stanice. Krb je zapnutý a dveře se otevřou přesně minutu před příchodem partnera.

Varianty testů:

Varianta 1: romanceInDay

1. Čekání 65 sekund.
2. Kontrola, že je rádio zapnuté a romantická hudba je naladěna.
3. Nasimulování normální úrovně osvětlení (1800 lm).
4. Kontrola, že jsou žaluzie stažené dolů.
5. Kontrola, že je krb zapnutý.
6. Kontrola, že jsou světla nastavena v romantickém módu.
7. Čekání 60 sekund.
8. Kontrola, že jsou vchodové dveře otevřené.

Varianta 2: romanceInTheEvening

1. Čekání 65 sekund.
2. Kontrola, že je rádio zapnuté a romantická hudba je naladěna.

3. Nasimulování nízké úrovně osvětlení (400 lm).
4. Kontrola, že jsou žaluzie vytažené nahoru.
5. Kontrola, že je krb zapnutý.
6. Kontrola, že jsou světla nastavena v romantickém módu.
7. Čekání 60 sekund.
8. Kontrola, že jsou vchodové dveře otevřené.

4.4.2 Případy pro Sklad

Obchod (koncový bod)

Tento testovací případ ověřuje integraci mezi patriot-gateway a logistic-chain ve chvíli, kdy dorazí nové balíky do obchodu, tedy jde o praktický průběh přejímky zboží.

1. Dorazí dodávka s balíky do obchodu.
2. Načítání (skenování) RFID kódů začíná.
3. Kontrolujeme, zda není naplněna kapacita obchodu.
4. Když je kód načtený, tak se uloží včetně data, adresáta a času.
5. Pokud je plná kapacita, tak je balík odmítnutý a nepřijmeme žádné další balíky. Když je kapacita stále volná, tak pokračujeme.
6. Zobrazí se informace, že byla zásilka akceptována.

Konkrétní testovací situace jsou popsány v části Varianty testů.

Předpoklady: Musí být nastavený čas v patriot-gateway (například na 10:28) a zároveň musí být nastavena informace udávající kapacitu daného obchodu.

Očekávané výsledky: Pokud není obchod ještě plný, tak začneme načítat balíčky. Po jejich načtení se uloží informace o datu, adresátovi a čase, a zobrazí se informace, že byl balík akceptován. Pokud je již plná kapacita, tak se všechny další balíky odmítnou.

Varianty testů:

Varianta 1: shopIsEmpty

1. Nasimulování, že dorazila dodávka.
2. Nasimulování načítání balíků (RFID kódy[1]).
3. Kontrola kapacity obchodu – volno.
4. Kontrola uložení data, adresáta a času.
5. Kontrola, že se zobrazila informace, že byl balík přijat.
6. Nasimulování načítání balíků (RFID kódy).
7. Kontrola kapacity obchodu – volno.
8. Kontrola uložení data, adresáta a času.
9. Kontrola, že se zobrazila informace, že byl balík přijat.
10. Nasimulování načítání balíků (RFID kódy).
11. Kontrola kapacity obchodu – volno.
12. Kontrola uložení data, adresáta a času.
13. Kontrola, že se zobrazila informace, že byl balík přijat.

Varianta 2: shopIsFull

1. Nasimulování, že dorazila dodávka.
2. Nasimulování načítání balíků (RFID kódy).
3. Kontrola kapacity obchodu – plno.
4. Kontrola, že byl balík odmítnut.

Varianta 3: shopCapacityIsAlmostFull

1. Nasimulování, že dorazila dodávka.
2. Nasimulování načítání balíků (RFID kódy).
3. Kontrola kapacity obchodu – volno.
4. Kontrola uložení data, adresáta a času.
5. Kontrola, že se zobrazila informace, že byl balík přijat.
6. Nasimulování načítání balíků (RFID kódy).

[1] RFID - identifikační technologie navazující na systém čárových kódů

7. Kontrola kapacity obchodu – plno.
8. Kontrola, že byl balík odmítnut.

Příjezd zboží

Tento testovací případ ověřuje integraci mezi patriot-gateway a logistic-chain ve chvíli, kdy dorazí nové zboží (balíky) do skladu , tedy jde o praktický průběh přejímky zboží.

1. Dorazí dodávka s balíky a zkontroluje se její povolení k vjezdu.
2. Pokud je vjezd povolen, tak se otevře brána (na určitou dobu).
3. Načítání (skenování) RFID[2] kódů začíná.
4. Když je kód načtený, tak se uloží včetně data, adresáta a času.
5. Zobrazí se informace, kam se má balík uložit.

Konkrétní testovací situace jsou popsány v části Varianty testů.

Předpoklady: Musí být nastavený čas v patriot-gateway (například na 10:28) a dodávky musejí mít povolení k vjezdu. Tato informace musí být nastavena a zároveň musí být nastavena informace, kam se mají dané typy balíků uskladnit.

Očekávané výsledky: Pokud má dodávka povolení k vjezdu, pak se otevře brána na 120 sekund a po tomto čase se zas zavře. Po naskenování balíků se uloží informace o datu, adresátovi a čase. Je vrácena informace, kam se má balík uskladnit. Pokud není vjezd pro danou dodávku povolen, tak se brána neotevře.

Varianty testů:

Varianta 1: truckIsAllowedOnePackage

1. Nasimulování, že dorazila dodávka (odeslání otisku).
2. Dodávka má povolení k vjezdu, takže se otevře brána.
3. Čekání 120 sekund.
4. Kontrola, že se brána zavřela.
5. Nasimulování načítání balíků (RFID kódy).

[2]RFID - identifikační technologie navazující na systém čárových kódů

6. Kontrola uložení data, adresáta a času.
7. Kontrola, že se zobrazila informace, kam se má balík uskladnit.

Varianta 2: truckIsNotAllowed

1. Nasimulování, že dorazila dodávka (odeslání otisku).
2. Dodávka nemá povolení k vjezdu, takže zůstane brána zavřená.
3. Čekání 120 sekund.
4. Kontrola, že je brána stále zavřená.

Varianta 3: truckIsAllowedLotOfPackages

1. Kontrola, že dorazila dodávka (odeslání otisku).
2. Dodávka má povolení k vjezdu, takže se otevře brána.
3. Čekání 120 sekund.
4. Kontrola, že se brána zavřela.
5. Nasimulování načítání balíků (RFID kódy).
6. Kontrola uložení data, adresáta a času.
7. Kontrola, že se zobrazila informace, kam se má balík uskladnit.
8. Nasimulování načítání balíků (RFID kódy).
9. Kontrola uložení data, adresáta a času.
10. Kontrola, že se zobrazila informace, kam se má balík uskladnit.
11. Nasimulování načítání balíků (RFID kódy).
12. Kontrola uložení data, adresáta a času.
13. Kontrola, že se zobrazila informace, kam se má balík uskladnit.
14. Nasimulování načítání balíků (RFID kódy).
15. Kontrola uložení data, adresáta a času.
16. Kontrola, že se zobrazila informace, kam se má balík uskladnit.

Výdejní místo

Tento testovací případ ověřuje integraci mezi patriot-gateway a logistic-chain ve chvíli, kdy dorazí nové balíky na výdejní místo.

1. Dorazí dodávka s balíky do obchodu.
2. Načítání (skenování) RFID kódů začíná a jejich stav se mění na „ready for pickup“
3. Je odeslán email adresátovi včetně času, do kdy si má balík vyzvednout. Adresát má omezený limit pro vyzvednutí.
4. Pokud si adresát balík vyzvedne, tak se balík označí jako „delivered“. Pokud si ho do stanoveného časového limitu nevyzvedne, tak se označí jako „returning back“.

Konkrétní testovací situace jsou popsány v části Varianty testů.

Předpoklady: Musí být nastavený čas v patriot-gateway (například na 10:28).

Očekávané výsledky: Když se načte balík, tak je odeslán email s časovým limitem pro vyzvednutí a balík je označen jako „ready for pickup“. Po nějakém čase zkontrolujeme, zda byl balík doručen (uživatel si ho vyzvedl) anebo uběhl časový limit a balík je označen jako „returning back“.

Varianty testů:

Varianta 1: pointPackageDelivered

1. Nasimulování, že dorazila dodávka.
2. Nasimulování načítání balíků (RFID kódy).
3. Kontrola, že je balík ve stavu „ready for pickup“.
4. Kontrola, že dorazil email.
5. Kontrola, že se zobrazila informace, že byl balík přijat.
6. Čekání po nějakou dobu (10 minut).
7. Nasimulování vyzvednutí balíku.
8. Kontrola, že je balík ve stavu „delivered“.

Varianta 2: pointPackageIsReturned

1. Nasimulování, že dorazila dodávka.
2. Nasimulování načítání balíků (RFID kódy).
3. Kontrola, že je balík ve stavu „ready for pickup“.
4. Kontrola, že dorazil email.
5. Kontrola, že se zobrazila informace, že byl balík přijat.
6. Čekání po nějakou dobu (10 minut).
7. Kontrola, že je balík stále ve stavu „ready for pickup“.
8. Čekání po nějakou dobu (10 minut).
9. Kontrola, že je balík ve stavu „returning back“.

4.4.3 Případy pro Chytré parkování

Registrace auta

Tento testovací případ ověřuje integraci mezi patriot-gateway a smart-parking ve chvíli, kdy si auto vytvoří registraci na parkování před svým příjezdem na parkoviště.

1. Auto odešle registraci (včetně platné SPZ) se zvoleným parkovacím místem.
2. Pokud je zvolené parkovací místo volné, tak se zarezervuje, pokud ne, tak autu přiřadíme jiné.
3. Auto přijede na parkoviště k závoře.
4. Zkontroluje se, zda má rezervaci a je volná kapacita a poté se otevře závora.

Konkrétní testovací situace jsou popsány v části Varianty testů.

Předpoklady: Musí být nastavený čas v patriot-gateway (například na 10:28).

Očekávané výsledky: Pokud je auto zaregistrované, tak se mu otevře závora a pustí ho dovnitř. Pokud není, tak bude auto vpuštěno dovnitř ve chvíli, kdy bude volné místo k parkování a dodatečně se zaregistruje.

Varianty testů:

Varianta 1: carIsRegistered

1. Nasimulujeme odeslání rezervace včetně SPZ se zvoleným parkovacím místem.
2. Kontrola, že je zvolené místo zarezervované.
3. Nasimulujeme, že auto přijelo na parkoviště k závoře.
4. Kontrola, že se závora otevřela.

Varianta 2: carIsNotRegisteredEmptyPlace

1. Nasimulujeme, že auto přijelo na parkoviště k závoře (bez rezervace).
2. Nasimulujeme, že je parkoviště volné (informace z informační tabule).
3. Kontrola stavu na tabuli.
4. Kontrola, že bylo auto zaregistrováno.
5. Kontrola, že se závora otevřela.

Varianta 3: carIsNotRegisteredFull

1. Nasimulujeme, že auto přijelo na parkoviště k závoře (bez rezervace).
2. Nasimulujeme, že je parkoviště již plné (informace z informační tabule).
3. Kontrola stavu na tabuli.
4. Kontrola, že je závora stále zavřená.

Varianta 4: carIsRegisteredNoPrefered

1. Nasimulujeme odeslání rezervace včetně SPZ se zvoleným parkovacím místem.
2. Kontrola, že je zvolené místo již obsazené.
3. Kontrola, že bylo vybráno jiné parkovací místo.
4. Nasimulujeme, že auto přijelo na parkoviště k závoře.
5. Kontrola, že se brána otevřela.

Parkování auta

Tento testovací případ ověřuje integraci mezi patriot-gateway a smart-parking ve chvíli, kdy auto přijíždí na parkoviště a po chvíli zase odjíždí.

1. Nasimulujeme, že je brána otevřená.
2. Závora se zavře za 30 sekund.
3. Auto zaparkuje na svém místě a čas přijezdu je zaznamenán.
4. Čekání po nějakou dobu (záleží na konkrétní variantě testu).
5. Auto odjíždí z parkoviště (záleží na testu).
6. Je spočtena cena za parkování.

Konkrétní testovací situace jsou popsány v části Varianty testů.

Předpoklady: Musí být nastavený čas v patriot-gateway (například na 10:28) a auto musí být zaregistrované.

Očekávané výsledky: Při zaparkování auta na zarezervované místo se uloží čas příjezdu a čeká se na odjezd auta (simulaci odjezdu auta). Nakonec se spočítá cena za parkování.

Varianty testů:

Varianta 1: carComesAndLeavesSoon

1. Nasimulujeme, že je brána otevřená.
2. Kontrola, že je brána po 30 sekundách opět zavřená.
3. Nasimulujeme, že auto zaparkovalo na svém místě.
4. Kontrola, že se uložil čas příjezdu.
5. Čekání 30 sekund.
6. Nasimulujeme, že auto odjíždí.
7. Kontrola, že se správně spočítala cena za parkování.

Varianta 2: carComesAndLeavesLater

1. Nasimulujeme, že je brána otevřená.
2. Kontrola, že je brána po 30 sekundách opět zavřená.
3. Nasimulujeme, že auto zaparkovalo na svém místě.
4. Kontrola, že se uložil čas příjezdu.
5. Čekání 120 sekund.
6. Nasimulujeme, že auto odjíždí.

7. Kontrola, že se správně spočítala cena za parkování.

Varianta 3: carComesAndLeavesSoLater

1. Nasimulujeme, že je brána otevřená.
2. Kontrola, že je brána po 30 sekundách opět zavřená.
3. Nasimulujeme, že auto zaparkovalo na svém místě.
4. Kontrola, že se uložil čas příjezdu.
5. Čekání 200 sekund.
6. Nasimulujeme, že auto odjíždí.
7. Kontrola, že se správně spočítala cena za parkování.

Auto odjíždí

Tento testovací případ ověřuje integraci mezi patriot-gateway a smart-parking ve chvíli, kdy auto odjíždí z parkoviště a kontroluje se, zda auto zaplatilo.

1. Auto odjíždí z parkovacího místa.
2. Spočítá se cena za parkování.
3. Pokud není parkovné zaplaceno, tak auto nemůže odjet.
4. Pokud je vše vyrovnáno, tak se otevře závora a auto může odjet.

Konkrétní testovací situace jsou popsány v části Varianty testů.

Předpoklady: Musí být nastavený čas v patriot-gateway (například na 10:28) a auto musí být zaregistrované a připravené k odjezdu.

Očekávané výsledky: Při odjezdu z parkovacího místa se spočítá cena parkovného a auto ho musí zaplatit. Pokud je parkovné zaplaceno, tak se otevře závora a pustí auto ven. Pokud není, tak zůstane závora zavřená.

Varianty testů:

Varianta 1: carLeavesPaid

1. Nasimulujeme, auto odjíždí ze svého parkovacího místa.
2. Kontrola, že se správně spočítala cena za parkování.

3. Kontrola, že je závora stále zavřená.
4. Kontrola, že je parkovné zaplacené.
5. Čekání 5 sekund.
6. Kontrola, že je závora otevřená.
7. Kontrola, že je parkovací místo označené jako „volné“.

Varianta 2: carLeavesWithoutPaying

1. Nasimulujeme, auto odjíždí ze svého parkovacího místa.
2. Kontrola, že se správně spočítala cena za parkování.
3. Kontrola, že je závora stále zavřená.
4. Kontrola, že je parkovné nezaplacené.
5. Kontrola, že je závora zavřená.
6. Kontrola, že je parkovací místo označené jako „obsazené“.
7. Čekání 30 sekund.
8. Kontrola, že je parkovné nezaplacené.
9. Kontrola, že je závora zavřená.

Kapitola 5
Implementační část

Tato kapitola se zaměřuje na popis zařízení, akčních členů a senzorů, a způsob jejich implementace včetně popisu a ukázek u daného fiktivního systému.

V každém modulu jsou různé typy akčních členů:

- BinaryStateActuator - má dva stavy (0-1, otevřeno-zavřeno, zapnuto-vypnuto)
- MultiStateActuator - lze si nadefinovat libovolný počet stavů

5.1 Chytrý domov

Následují ukázky některých z implementovaných komponent systému Chytrý domov.

5.1.1 Akční člen (Actuator)

Nově implementovaným akčním členem v domečku je:

- SoundSystem - rádio

SoundSystem

Akční člen SoundSystem je ve své podstatě rádio, které umožňuje zapínání a vypínání a také změnu kanálu. V SoundSystem routes jsou již přednastavené oblíbené kanály jako: News, Romantic nebo Rock.

Při zaslání dotazu k nastavení Romantic kanálu se zároveň zapne rádio jako takové, pokud bylo vypnuté.

Ukázku rádia (SoundSystem) můžete vidět níže.

Ukázka

```
public class SoundSystem extends MultiStateActuator {

    // zadny vychozi kanal
   private String channel = "none";

    // konstruktor
   public SoundSystem(String label) {
    super(label);
   }
   public String getChannel() { // getter
        return this.channel;
   }
   public void setChannel(String channel) {     // setter
        this.channel = channel;
   }
```

Logika tohoto akčního členu je co nejjednodušší. Rozšiřuje MultiStateActuator, protože má několik stavů (v našem případě kanálů nebo stanic), které můžeme přepínat. Na které kanály můžeme přepínat je definováno v modulu PatrIoT Gateway v io.patriot.framework.commands.SoundSystemCommand.

Měli bychom také mít trasy (routes) na straně domu, abychom mohli komunikovat s bránou. Z tohoto důvodu jsou trasy pro SoundSystem vytvářeny v io.patriot_framework.samples.smarthome.smart_home_virtual.routes.SoundSystemRouteBuilder.

```
public class SoundSystemRouteBuilder
    extends IntelligentHomeRouteBuilder { // trasa

 //hlavni metoda
 @Override
 public void configure() throws Exception {
 final HouseBean houseBean = HouseBean.getInstance();

 from("direct:soundsystem-romantic")     // interni from krok
     // hlavicka
 .setHeader("soundsystem", simple("/root/romantic.mp3"))
    // bean metoda
 .bean(HouseBean.getInstance(), "configureSoundSystem");

 // pristupna trasa
 from(restBaseUri() + "/radio/romantic?httpMethodRestrict=GET")
  .to("direct:soundsystem-romantic") // interni .to
  .to("direct:soundsystem");             // interni .to
```

```
// pristupna route
from(restBaseUri() + "/radio?httpMethodRestrict=GET")
// interni.to
  .to("direct:soundsystem");

from("direct:soundsystem")                    // interni from
  .setProperty("type", simple(SoundSystem.class.getSimpleName()))
  .bean(houseBean, "objInfo")     // o jake jde zarizeni
  // hlavicka
  .setHeader("content-type", constant("application/json"))
  .marshal().json(JsonLibrary.Jackson); // vraci JSON
```

Výše popsané trasy se používají pro práci Gateway s Rádiem. Chceme-li přidat nový akční člen, musíme postupovat podle následujících 3 kroků:

- přidat akční člen do souboru house.yaml
- zaregistrovat ho do io.patriot_framework.samples.smarthome.smart_home _virtual.house.Device
- přidat bean metodu[1] a novou část metody objInfo s konfigurací zprávy do io.patriot_framework.samples.smarthome.smart_home_

  virtual.house.HouseBean

Chceme-li přidat komponentu jako SoundSystem, musíme ji přidat do souboru house.yaml:
- name: soundsystem -> jméno si zvolíme libovolně (slouží nám pro lepší identifikaci)
type: SoundSystem -> typem je jméno třídy (v tomto případě třída SoundSystem.java)

Registrace Rádia do Device.java se provádí přidáním tohoto kódu do metody:

```
private static Device performCreation(String type,
  String name, Object[] args) throws
   IllegalArgumentException {

   if ("SoundSystem".equalsIgnoreCase(type)) {
        return new SoundSystem(name); }
```

Posledním krokem je přidání bean metody s konfigurací zprávy. Mělo by to vypadat takto:

[1]Bean metoda - třídy (class), k jejichž obsahu se přistupuje pomocí referencí

```
public void configureSoundSystem(Message msg) {

    String media = msg.getHeader("soundsystem", "none",
        String.class);
  // ID soundsystem je nami zvolene jmeno z house.yaml
  // souboru
  house.getDeviceWithId("soundsystem", SoundSystem.class)
    .setChannel(media);
}
```

A přidejte novou část metody objInfo:

```
public void objInfo(Exchange exchange) {
   //   dalsi kod
   } else if (exchange.getProperty("type")
     .equals(SoundSystem.class.getSimpleName())) {

         object = house.getDeviceWithId("soundsystem",
         SoundSystem.class);
```

5.1.2 Zařízení (Device)

Mezi nově implementovaná zařízení v domečku patří:

- CoffeeMaker
- EntranceGate
- Sunblinds

CoffeeMaker

Zařízení CoffeeMaker simuluje chování běžného kávovaru. Umožňuje kávovar zapnout/vypnout, nastavit si požadované množství vody a sílu kávy. Kávovar se umí také v případě nedostatku vody nebo kávy zastavit a počkat na doplnění. Využívá se zde StateMachine (stavový automat) definující jednotlivé stavy kávovaru.

Kávovar má také vlastní CoffeeMakerActuator (vznikl rozšířením AbstractActuator z patriot-data-generator), který zahrnuje již zmiňovanou definici stavového automatu.

EntranceGate

Vstupní brána (neboli EntranceGate) je zařízení simulující branku u vjezdu k domu. Tato vstupní brána využívá DoorActuator (stavový automat umožňující otevírání a zavírání brány).

Sunblinds

Sunblinds, tedy žaluzie, patří také mezi zařízení. Umožňují pohyb nahoru a dolů a do určitých výškových pozic.

Ukázka

Zde můžete vidět příklad kávovaru (CoffeeMaker) definovaného v io.patriot_framework.samples.smarthome.smart_home_virtual.house.CoffeeMaker.

```
public class CoffeeMaker extends Device {        // zarizeni
   // aktuator bude popsan nize
  private CoffeeMakerActuator act;

   // konstruktor
  public CoffeeMaker(String name, int waterForCoffee, int strength) {
        super(name);
        // nastaveni
    act = new CoffeeMakerActuator(name, waterForCoffee, strength);
}
  public CoffeeMaker(String name) { // konstruktor
        super(name);
        act = new CoffeeMakerActuator(name, 150, 7);
}  //vychozi hodnoty

  public void setDefaultWater(int defaultWater) {
         act.setHowMuchWaterForCoffee(defaultWater);
}    //mnozstvi vody

  public void setDefaultCoffee(int defaultCoffee) {
         act.setCoffeeStrength(defaultCoffee);
}    // mnozstvi kavy

  public void turnOfCoffeeMaker() {       // prikaz vypnuti
     act.turnOfCoffeeMaker();
}

 public void turnOnCoffeeMaker() {       // prikaz zapnuti
     act.turnOnCoffeeMaker();
}
```

```
 public void makeCoffee() {      // prikaz pripravuj kavu
     act.makeCoffee();
}
```

Třída io.patriot_framework.samples.smarthome.smart_home_virtual.house .CoffeeMakerActuator rozšiřuje AbstractActuator. Hlavním účelem tohoto akčního členu je řešit nastavení výchozího množství vody a kávy, které máme, a ovládat StateMachine. K dispozici je také definice StateMachine pro CoffeeMaker včetně nastavení podmínek pro některé stavy.

Podmínku lze nastavit a použít tímto způsobem:

```
// stavovy automat kavovaru
StateMachine machine = new StateMachine.Builder()
      .from("Ready to make coffee", 3000) // from krok
          .to("Checking water", "check")  // to krok a akce
          .to("Off", "off")          // to krok and akce

    .from("Checking water", 3000) // from krok
          .to("Fill water")             // to krok
          .to("Enough water")        // to krok
          // podminka s parametry
          .condition(checkWaterCondition, waterForCoffee)

      .build();

this.setStateMachine(machine);
}

   // condition for checking water amount
Condition checkWaterCondition = new Condition() {

 @Override
 public String con(int data) {
        boolean enoughWater=setHowMuchWaterForCoffee(data);

        if (!enoughWater) {
        // nazev kroku, kam budeme pokracovat...
                return "Fill water";
                }
        // nazev kroku, kam budeme pokracovat...
        return "Enough water";
        }
   };
```

StateMachine tedy rozhodne, do kterého kroku .to() půjdeme - v závislosti na výsledku podmínky checkWaterCondition. Je potřeba si uvědomit, že je třeba definovat všechny kroky a v názvu kroku nesmí být překlep.

Pokud chcete kávovar přesunout z jednoho stavu do dalšího, máte dvě možnosti:

- zavolat this.controlSignal (event) - událost má v definici kroku akci (jako „off“ nebo „check“)
- splnit podmínku -> výsledek podmínky rozhodne, do kterého stavu se přejde a StateMachine se automaticky přepne do dalšího stavu (kroku).

5.1.3 Senzor (Sensor)

Další komponentou v domečku je senzor. Nově máme v domečku přidány tyto senzory:

- COSensor
- CO2Sensor
- MotionSensor
- LightLevelSensor
- FireSensor

COSensor

Tento senzor měří koncentraci CO (oxidu uhelnatého) ve vzduchu. Jeho jednotkou je PPM (parts per million) – odpovídá počtu částic na jednu miliontinu.

Generované hodnoty jsou v rozmezí od 40 do 80 PPM s využitím normální distribuce (NormalDistVariateDataFeed) z patriot-data-generator modulu. Bezpečné hodnoty jsou do 70 PPM [10].

Pro potřeby testů disponuje senzor metodami, které přenastavují generované hodnoty. Metoda přenastavení vypadá takto:

```
public void setConstant50DataFeed(){
// generovani konstantni hodnoty 50
    DataFeed newFeed = new ConstantDataFeed(50);
    sensor.addDataFeed(newFeed);
}
```

CO2Sensor

CO2Sensor měří koncentraci CO_2 (oxidu uhličitého) v ovzduší. Jeho jednotkou je PPM (parts per million) – odpovídá počtu částic na jednu miliontinu. Generované hodnoty jsou v rozmezí od 700 do 1100 PPM s využitím normální distribuce (NormalDistVariateDataFeed) z patriot-data-generator modulu. Bezpečné hodnoty jsou do 1000 PPM [10].

MotionSensor

Senzor pohybu (neboli motion sensor) je senzor sloužící k detekci pohybu – například pro zapínání/vypínání osvětlení.

Jeho generované hodnoty jsou 1 (pohyb) nebo 0 (žádný pohyb) s využitím patriot-data-generator modulu.

Pro potřeby testů disponuje senzor metodami, které přenastavují generované hodnoty na konstantní z 0 na 1 a naopak.

LightLevelSensor

Senzor úrovně osvětlení (light level sensor) je senzor generující hodnoty v jednotkách lm (lumen), což je jednotka světelného toku.

Generované hodnoty jsou v rozmezí od 800 do 1800 lm.

Pro potřeby testů disponuje senzor metodami, které přenastavují generované hodnoty na konstantních 800 nebo 1800 lm [10].

FireSensor

Tento senzor slouží k detekci požáru. Pro zjištění požáru využívá hodnoty ze senzoru CO a senzoru teploty.

Generované hodnoty jsou 1 (hoří) nebo 0 (nehoří) a získávají se na základě vyhodnocení PPM z CO senzoru a také na základě teploty.

Ukázka

Na příkladu CO senzoru si lze ukázat, jak vytvořit základní senzor.

Nejprve musíme vědět, který senzor bychom chtěli vytvořit. Také bychom měli vědět, které jednotky má měřit. V našem případě stupně PPM.

```
public static final String DEFAULT_UNIT = "PPM";
// parts per million
```

Vzhledem k tomu, že CO senzor bude uvnitř domu, očekávané hodnoty jsou někde mezi 40-80 (v celočíselných hodnotách). Proto zvolíme NormalDistributionDataFeed.

```
private DataFeed dataFeed = new NormalDistVariateDataFeed(60, 20);
```

Poté vytvoříme konstruktor a metody jako get a set a nakonec vytvoříme novou třídu:

```
public class COSensor extends Sensor
  implements SimpleValueSensor<Float> {

    public static final String DEFAULT_UNIT = "PPM";      // jednotka
    private String unit;

    private DataFeed dataFeed = new NormalDistVariateDataFeed(60, 20);
    private io.patriot_framework.generator.device.Device device =
    = new io.patriot_framework.generator.device.impl.
    basicSensors. COSensor (getLabel(), dataFeed);

    public COSensor (String label) {              //konstruktor
      this(label, DEFAULT_UNIT);
    }

    @Override
    public Integer getValue() {   // vygenerovana hodnota
      return device.requestData().get(0).get(Double.class).intValue();
    }

    @Override
    public String getUnit() {             // jednotka
      return this.unit;
    }
```

Kombinace zařízení

V projektu jsou kromě akčních členů, zařízení a senzorů možné také kombinace výše zmíněných komponent.

Příkladem je FireSensor - je to zařízení využívající data ze senzorů (CO a teplota). Jedná se tedy o kombinaci dvou senzorů. Vypadá takto:

```
public class FireSensor extends Device {

    private Fire device;

    public FireSensor (String label) {
     super(label);
     // vygenerovana data
     DataFeed co = new NormalDistVariateDataFeed(60, 20);
```

```
 DataFeed temp = new NormalDistVariateDataFeed(25, 3);
 device = new Fire("FireSensor",temp, co);
}

public Float getTemperature() {
 return getValue(0);
}

public Float getCO() {
 return getValue(1);
}

private Float getValue(int index) {
 List<Data> data = device.requestData(null);
 return data.get(0).get(Double.class).floatValue();
}

public String buildMessage() { // vytvareni zpravy
 List<Double> values = device.requestData(null)
 .stream().map(it -> it.get(Double.class))
 .collect(Collectors.toList());

 return "{\"temperature\" : " + values.get(0).intValue()
 + ", \"co\" : " + values.get(1).intValue() +
 ", \"label\" : \"" + getLabel() + "\"}";
     }
}
```

Trasy (routes) jsou definovány stejným způsobem jako u zařízení typu Actuator – tedy definice v house.yaml, registrace ve třídě Device a HouseBean jsou stále stejné.

5.2 Sklad

Následují ukázky některých z implementovaných komponent systému Sklad.

5.2.1 Akční člen (Actuator)

V rámci systému Sklad nejsou implementovány žádné nové akční členy.

5.2.2 Zařízení (Device)

Mezi nově implementovaná zařízení ve skladu patří:

- DispensingPointPackage
- Shop
- VanTruck
- WarehousePackage

DispensingPointPackage

DispensingPointPackage - tedy výdejní místo je třída držící si stav balíčků, které do něj doveze dodávka. Umožňuje načítání nových balíčků a například i jejich změnu stavu.

Shop

Shop je obchod, který také přijímá balíčky, ale zároveň má i svou maximální kapacitu a může balíčky odmítnout.

VanTruck

VanTruck je třída, která se stará o povolení k vjezdu do objektu skladu. Můžeme přidávat nové dodávky nebo nákladní automobily a přidávat jim povolení k vjezdu.

WarehousePackage

WarehousePackage reprezentuje sklad jako takový, do kterého mají povolení k vjezdu jen určití dodavatelé (resp. dodávky a nákladní automobily). Tento sklad přijímá balíčky a určuje místo k jejich uložení.

Ukázka

Ukázkou je část z Shop - konrétně jeho vytvoření včetně inicializace.

```
public class Shop extends Device {
    // abstraktni akcni clen majici
    // na starosti prepinani stavu (kapacity)
    private ShopActuator act;

    // aktualni pocet baliku
    private int actual = 0;
```

```
// maximalni pocet baliku
private int maximum = 5;
List<String[]> packages;

// konstruktor
public Shop(String label) {
    super(label);
    act = new ShopActuator(label);
    packages = new ArrayList<String[]>();
    actual = 0;
}
// dalsi kod
```

5.2.3 Senzor (Sensor)

V rámci systému Sklad nejsou implementovány žádné nové senzory.

5.3 Chytré parkování

Následují ukázky některých z implementovaných komponent systému Chytré parkování.

5.3.1 Akční člen (Actuator)

V rámci systému Chytrého parkování nejsou implementovány žádné nové akční členy.

5.3.2 Zařízení (Device)

Mezi nově implementovaná zařízení v Chytrém parkování patří:

- Cars
- ParkingPlace

Cars

Třída Cars reprezentuje seznam zaregistrovaných a zaparkovaných automobilů. Cars dále hlídá dobu parkování, vybrané místo stání a také drží stav aktuálně zaparkovaných automobilů.

ParkingPlace

ParkingPlace udržuje stav parkovacích míst, zda jsou volná nebo obsazená a obsahuje v sobě i informační tabuli ukazující aktuální volnou kapacitu k parkování.

Ukázka

Níže je zobrazena ukázka části Cars.

```
public class Cars extends Device {
    // seznam automobilu
    List<String[]> cars;

    public Cars(String label) {
        super(label);
        cars = new ArrayList<String[]>();
    }
    // aktualni pocet aut
    public int getActual() {
        return cars.size();
    }
    // pridani registrace auta - cas prijezdu resi gateway
    public void addCar(String spz, String time, String place,
      String reservation) {
        String arr[]=new String[5];
        arr[0]=spz;
        // cas muze byt nulovy, pokud auto jeste nedorazilo
        arr[1]=time;
        arr[2]=place;
        // na zacatku neni zaplaceno
        arr[3]="no"; // placeno, neplaceno
        arr[4]=reservation;
        cars.add(arr);
    }

    // dalsi kod
```

5.3.3 Senzor (Sensor)

V rámci systému Chytrého parkování nejsou implementovány žádné nové senzory.

5.4 Gateway (brána)

Gateway je modul v rámci PatrIoT framework napsaný z části v Drools[2] a z části v Apache Camel, který spojuje virtuální dům a jeho trasy s cestami „vnějšího světa“ za účelem popisu pravidel komunikace. To znamená, že můžeme simulovaný systém ovládat pomocí jednoduchých příkazů.

Jedním z cílů této diplomové práce bylo přepsat modul patriot-gateway, aby využíval jen Apache Camel a odstranit Drools.

5.4.1 Struktura

Patriot-gateway má pouze jedno přímé připojení (s Chytrým domovem nebo Skladem nebo Chytrým parkováním) a po přepsání do Apache Camel je jeho struktura následující:

- App - hlavní aplikace a definice tras
- Commands - třídy pro udržování stavu jednotlivých senzorů, zařízení a akčních členů
- Processors – zpracovávají a řeší výměnu zpráv (požadavků a odpovědí)

App část

V části App existují dvě třídy:

- PatrIoTGatewayApplication - nastavení hlavní třídy pro spuštění jako SpringApplication[3]
- PatrIoTGatewayRouter - třída, která rozšiřuje RouteBuilder z Apache Camel a kde jsou připraveny trasy (routes) k fiktivnímu systému

[2] Drools - technologie pro vytváření business pravidel fungování systému - viz WWW: <https://www.baeldung.com/drools>

[3] SpringApplication - aplikace běžící pomocí Spring

Commands část

Tato část (balíček) obsahuje třídy, které zpracovávají stav senzorů/zařízení/akčních členů. Tyto třídy jsou ve skutečnosti objekty s předdefinovanými settery/gettry. Ukázka této třídy je uvedena níže v podkapitole Funkcionalita.

Processors část

Procesory z pohledu Apache Camel jsou třídy, kde jsou zpracovávány objekty Exchange[4]. Například zde můžeme nastavit hlavičku (header) a tělo (body) požadavku/odpovědi pomocí objektů z Commands. Třídy jsou podrobněji popsány v následující kapitole.

Funkcionalita

Začněme příkladem jedné z třídy Commands. Zvolená ukázka je z patriot-gateway pro Chytrý domov a jmenuje se SoundSystemCommand.java.

```
public class SoundSystemCommand extends Command {

    public enum Sound {          // enum se stavy Radia

        OFF, ROMANCE, NEWS, ROCK, ON     // stavy
    }

    private Sound sound = Sound.OFF;     // prvotni stav

    public SoundSystemCommand(final Sound sound) {
        this.sound = sound;

    }
    public Sound getSound() {    // getter
        return sound;
    }
    // override metody
}
```

Tato třída tedy obsahuje stav SoundSystem (Rádia), který lze použít v trasách (routes) nebo procesorech (processors).

[4]Exchange - Exchange objekty - v těchto objektech se posílají zprávy mezi jednotlivými komponentami

Trasy lze definovat v PatrIotGatewayRouter, který rozšiřuje RouteBuilder z Apache Camel.

```
@Component
public class PatrIotGatewayRouter extends RouteBuilder {
   // definice routes

        // hlavni metody
   @Override
   public void configure() throws Exception {

    // implementace metod viz nize
      configureSoundSystemRoutes();
          configureMoodActionRoutes();
   }
```

Implementace tras pro SoundSystem pak vypadají takto:

```
private void configureSoundSystemRoutes() throws Exception {

 from("direct:radio").routeId("io.Radio") // hlavni route
    .choice() // vice moznosti, jako IF-ELSE
    .when().simple("${header.sound} == '" +
    + SoundSystemCommand.Sound.OFF + "'")
        .to("direct:radioOff")  // kam poslat zpravu

    .when().simple("${header.sound} == '" +
    + SoundSystemCommand.Sound.ON + "'")
        .to("direct:radioOn")

    .when().simple("${header.sound} == '" +
    + SoundSystemCommand.Sound.NEWS + "'")
        .to("direct:radioNews")

    .when().simple("${header.sound} == '" +
    + SoundSystemCommand.Sound.COFFEE + "'")
        .to("direct:radioCoffee")

    .otherwise()                    // jako ELSE
        .to("direct:radioRomantic");

 from("direct:radioRomantic")    // dalsi route
    // text zpravy, ktery chceme poslat
    .setBody().constant("radio")
```

```
        .removeHeaders(Exchange.HTTP_QUERY) // smazeme hlavicky
        // Exchange metoda
        .setHeader(Exchange.HTTP_METHOD, constant("GET"))

    // URL kam chceme poslat zpravu, musi existovat v systemu
    //   jinak dostaneme chybu behem kompilace

    .to("http://" + iotHost + "/radio/romantic?bridgeEndpoint=true");
```

Nutno zmínit, že je třeba definovat všechny přímé trasy, které jsme použili s .to(), a všechny adresy URL, které jsme použili musí existovat v domku, jinak se při kompilaci Gateway zobrazí chyba.

Můžeme také definovat některá pravidla, která lze spustit pomocí konkrétní adresy URL, a použít více cest .to() v jednom. Například:

```
// pravidlo
private void configureMoodActionRoutes() throws Exception {

    // processors budou ukazany pozdeji
    MoodActionEveningProcessor moodActionEveningProcessor=
    =new MoodActionEveningProcessor();

    from(url+"/MoodAction").routeId("rule.MoodAction")
    // co zalogovat do konzole
        .log("Received MoodAction")
        .choice()
        .when().simple("${header.mood} == 'evening'")
            .process(moodActionEveningProcessor)
            .to("direct:fire")
            .to("direct:led_ALL")
            .to("direct:ledBatchMoodAction")
            .to("direct:media")
            .to("log:rule.MoodAction")

    // ostatni routes
}
```

Po spuštění hlavní trasy pravidla MoodAction se bude pokračovat na „večerní“ trasu. Na „večerní“ trase se spouští .process() a všechny trasy .to() -> spouští jednu po druhé a loguje ji do konzole. Zavolá se tedy krb, všechna světla a media (rádio).

Nyní se zaměříme na metodu .processor() s příkladem MoodActionEveningProcessor.

```
// metoda procesoru, ktera implementuje Processor z Apache Camel
```

```
public class MoodActionEveningProcessor implements Processor {

    private Logger logger = logger.
      .getLogger(MoodActionEveningProcessor.class.getName());

    // tato metoda se spusti automaticky zavolanim
    // .processor() z route
    public void process(Exchange exchange) {

          // ulozeny text/hlavicka ze zpravy
          Message in = exchange.getIn();

    // nastaveni hlavicky
          exchange.getOut().setHeader("processor",
          , "moodActionEveningProcessor");

          // nastaveni stavu krbu
    FireplaceCommand fireplaceCommand=
     =new FireplaceCommand(FireplaceCommand.Fire.HEAT);
     // hlavicky
          exchange.getOut().setHeader("fire",
          ,simple(fireplaceCommand.getFire().toString()));

          // nastaveni statusu vsech svetel a hlavicek
          LightCommand lightCommand=
          =new LightCommand(LightCommand.Place.ALL, LedState.ON);
          exchange.getOut().setHeader("light_ALL",
          ,lightCommand.getPlace().getLed());
          exchange.getOut().setHeader("light_ALL_State",
          ,lightCommand.getState());

          // dalsi kod
}
     // metoda pro ovladani vice svetel najednou
    public String getLightCommands(){

     // nastaveni
     BatchLightCommand batchLightCommand=new BatchLightCommand();

   // nastaveni mista (o jake svetlo se jedna a stav)
     batchLightCommand.addCommand(new LightCommand(
     LightCommand.Place.LIVINGROOM_FIREPLACE,
          new LedState(10,10,10)));
     batchLightCommand.addCommand(new LightCommand(
     LightCommand.Place.LIVINGROOM_LIBRARY,
          new LedState(10,10,10)));
```

```
        batchLightCommand.addCommand(new LightCommand(
        LightCommand.Place.LIVINGROOM_COUCH,
            new LedState(10,10,10)));

        return batchLightCommand.getBatch();
            }
}
```

Metodu getLightCommands(), která vrací řetězec (String), lze použít v definici trasy pro nastavení hlavičky/textu. Pro lepší pochopení viz níže uvedený příklad.

```
// route definice
from("direct:ledBatchMoodAction").routeId("io.BatchMoodAction")
            // nastaveni typu
    .setHeader(Exchange.CONTENT_TYPE, constant("application/json"))
     // http metoda
    .setHeader(Exchange.HTTP_METHOD, constant("POST"))
    .choice()

     // pokud hlavicka odpovida, tak se nastavi
     // MoodActionEveningProcessor
    .when().simple("${header.processor} == '" +
      + "moodActionEveningProcessor" + "'")

    // nastaveni zpravy (body) s pomoci retezce,
    // ktery nam vrati metoda
    .setBody(simple(moodActionEveningProcessor.getLightCommands()))
    .to("http://" + iotHost + "/led/batch?bridgeEndpoint=true")

// dalsi kod
```

Vhodným pomocníkem při práci s patriot-gateway a nastavováním tras je zapnutí logování přijatých zpráv, viz:

```
.streamCaching()  // povoleni stream caching
.convertBodyTo(String.class) // prevedeni zpravy do String
.log("${body}")  // vypis zpravy (zalogovani do konzole)
```

5.4.2 Implementace testovacích případů

Pro spuštění integračních testů jsou připraveny cesty (routes) pro testovací případy (TC). Testovací případy jsou spuštěny odesláním požadavku na konkrétní adresu URL nebo pomocí nasimulování konkrétní akce (například simulací pohybu pomocí poslání nastavovacího dotazu do patriot-data-generator).

Příkladem je kód testovacího případu „Ranní rutina“ z Chytrého domova. Předpoklady testu - musí být nastaven čas v patriot-gateway a čas budíku.

Na začátku tedy existuje trasa (route) čekající na požadavek s nastavením času a času budíku. Když získá informace, nastaví hodnotu hodin. Je implementována tímto způsobem:

```
// cas na budiku se nastavi jako
// hlavicka zpravy
from(url+"/alarmclock").routeId("rule.AlarmClock")

  .log("Received alarm clock settings")
  .process(new Processor() {
  @Override
  public void process(Exchange exchange) throws Exception {

   int h=Integer.parseInt(exchange.getIn().getHeader("h")
    .toString());
   int m=Integer.parseInt(exchange.getIn().getHeader("m")
    .toString());
   timeAndDayCommand.setAlarmTime(h,m);
    }
    });
```

Poté nekonečná smyčka zkontroluje, zda přichází správný čas (čas alarmu) nebo ne.

```
 // nekonecna smycka se spousti kazdou vterinu
from("timer:alarmChecker?period=1000")
.log("motionSensorChecker")
.process(new Processor() {
  @Override
  public void process(Exchange exchange) throws Exception {

    exchange.getOut().setHeader("alarmTime", "false");
    // probuzeni za 1 minutu
    if(timeAndDayCommand.getAlarmTime()==
        ==timeAndDayCommand.getTime()+60){

        exchange.getOut().setHeader("alarmTime", "true");
        doorCommand.setWindowsOpenBool(true);
```

```
        }

    else if(doorCommand.windowsAreOpen()==true) {

        int timeLast = doorCommand.getTime();

        //List<Integer> list=calculateTime(timeActual, true, timeLast);
        // dosazen limit, zhasinaji se svetla
        if (timeAndDayCommand.getTime()-timeLast>=
            doorCommand.getTimeLimit()){

            doorCommand.setWindowsOpenBool(false);
            exchange.getOut().setHeader("closeWindow", true);

        }
    }

        }
    })
    .choice()
    // neco se bude updatovat
    .when().simple("${header.alarmTime} == 'true'")
        .to("direct:sunblindsUp")
        .to("direct:wakingUp")

    // zacina morning TC
    .when().simple("${header.closeWindow} == 'true'")
        .to("direct:windowClose")
        .log("closing windows....")
    .otherwise()
        .log("nothing");
        .to("direct:coffeeOn");

    from("direct:wakingUp").routeId("rule.TestCasesWakingUp")
    .log("Received TestCasesWakingUp rule")
    .streamCaching()
    .to("http://" + iotHost + "/hygrometer?bridgeEndpoint=true")
    .bean(hygrometerCommand, "setHumidity")
    .to("http://" + iotHost + "/co?bridgeEndpoint=true")
    .bean(co2Command, "setCo2Amount")

    .process(new Processor() {
        @Override
        public void process(Exchange exchange) throws Exception {

            // je vysoka vlhkost, pak otevri okna
```

```
            if(hygrometerCommand.getHumidity() >= 60 ||
                co2Command.getCO2Amount() > 1000){

                exchange.getOut().setHeader("openWindows", true);
            }

            else {
                exchange.getOut().setHeader("openWindows", false);

            }
        }
});

        .choice()
        .when().simple("${header.openWindows} == 'true'")
        .to("direct:windowOpen") // jen na urcity cas

        .otherwise()
        .log("no window opening");
```

Nejdříve se tedy nastaví čas budíku a ten je neustále kontrolován. Ve chvíli, kdy nastane správný čas (1 minuta do zvonění budíku), tak se provedou akce jako vytažení žaluzií, kontrola vlhkosti a CO_2, a zapnutí kávovaru.

Veškeré testovací případy jsou napsány s obdobnou mechanikou - nejdříve se čeká na nějaký impulz (například pohyb, příjezd dodávky, čas buzení). Poté začne nekonečná smyčka kontrolovat splnění (i nesplnění) podmínek a na jejich základě a nastavení reagovat a provádět určité akce. Tento princip je aplikován u všech tří simulací fiktivních systémů.

Kapitola 6

Testovací část

Základním stavebním kamenem správně fungující aplikace (systému) je testování. Abychom mohli říci, že systémy správně interagují (spolupracují), je potřeba provést jejich otestování. Nutno zmínit, že testování IoT nebývá bezproblémové (viz kapitola 2.2.3)

Pro otestování tří fiktivních systémů (Chytrý domov, Sklad, Chytré parkování) byly navrženy integrační E2E testy využívající RestAssured technologii.

Cílem tohoto testování je zjistit, zda se všechny příkazy na patriot-gateway dostanou do simulovaného systému a zda na ně simulovaný systém správně zareaguje.

6.1 Integrační testy

Každý testovací případ má vlastní testovací soubor (třídu), ve kterém jsou definovány varianty testů. Navržené varianty testů byly popsány v kapitole 4.

Následuje ukázka testu WakeUpAtNightFastReturn z modulu patriot-virtual-smart-home. Před každým spuštěním testu se nejdříve provedou kroky metody označené @BeforeEach (to je metoda obsahující kroky nastavující čas v patriot-gateway). V další části se spustí samotný test.

```
@Test
public void wakeUpAtNightFastReturn() {

    // nastaveni url pro domek a pro gateway
    RestAssured.defaultParser = Parser.JSON;
    RequestSpecification home = getRequest("smarthome",
      "SmartHome", 8282);
    RequestSpecification gateway = getRequest("gateway",
      "GatewayNetwork", 8283);
    // nasimulovani pohybu
    gateway.basePath("/setmotion").header("move","1").when()
      .get().then().statusCode(200);
    waitForSec(1000);
    // ukonceni simulovani pohybu
```

```
gateway.basePath("/setmotion").header("move","0").when()
   .get().then().statusCode(200);
waitForSec(1000);
// kontrola svetel, zda se rozsvitila
home.basePath("/led")
        .when()
        .get()
        .then()
        .statusCode(200)
        .and()
        .body("[12].label", equalTo("colorful-light-8"))
// pokracovani testu
```

Na obdobném principu byly implementovány všechny testy ověřující tři fiktivní systémy.

6.2 Výsledek testování

Před spuštěním testů v Docker[1] je nejdříve nasazena a spuštěna část simulovaného systému a poté část gateway. Po vytvoření kontejnerů pro běh testů jsou spuštěny automatizované integrační testy. Po spuštění začínají posílat dotazy na gateway a simulovaný systém a po skončení hlásí výsledky.

Po průchodu výsledků testů (u každého simulovaného systému zvlášť) je možné konstatovat, že všechny testy skončily úspěchem. Z dokončeného testování lze učinit závěr, že sada příkladů tří fiktivních systémů funguje.

Poznámka: Návod k instalaci a spuštění testů je k dispozici v kapitole 6.3.

6.3 Instalace a spuštění

Každý z implementovaných systémů má celkem 4 části (resp. moduly), které je nutné před spuštěním zkompilovat a správně nastavit. Tento konkrétní postup je ukázán na systému Chytrého domova. Postup je pro všechny systémy stejný, tudíž si ho lze pro další 2 systémy analogicky odvodit.

Budeme nastavovat tyto části:

- patriot-data-generator
- patriot-virtual-smart-home
- patriot-gateway

[1] Docker - software, který poskytuje jednotné rozhraní pro izolaci aplikací do kontejnerů

- patriot-integration-tests

(Poznámka: Může se stát, že Maven nebude mít dostatečná práva pro instalaci závislostí. V tom případě se doporučuje kompilovat jako administrátor – resp. s přidáním „sudo“ před „mvn“ příkazy.)

Je důležité všechny moduly kompilovat a spouštět pomocí Java ve verzi 8. Použití jiné verze není zatím ve frameworku podporováno a není tedy garance, že bude vše fungovat správně.

6.3.1 Patriot-data-generator

Tímto modulem je nutno začít, protože jsou na něm další z částí závislé a bez něho nemohou fungovat.

Nejprve se tedy přepneme do jeho složky příkazem:

```
cd patriot-data-generator
```

Dále ho necháme zkompilovat a nainstalovat všechny závislosti Maven pomocí:

```
mvn clean install
```

První běh kompilace trvá delší dobu. Poté se vrátíme o jednu složku nazpět:

```
cd ..
```

6.3.2 Patriot-virtual-smart-home

Pokud máme generátor připravený, tak můžeme pokračovat na modul samotného domku. Začneme přechodem do složky a kompilací:

```
cd patriot-virtual-smart-home
mvn clean package
```

Opět si chvilku počkáme, protože si musí Maven stáhnout všechny potřebné závislosti.

Následuje vytvoření Docker Image[2] s názvem:

[2]Docker Image - šablona obsahující instrukce pro vytváření kontejnerů, které mohou běžet na Dockeru, viz WWW: <https://jfrog.com/knowledge-base/a-beginners-guide-to-understanding-and-building-docker-images/>

```
docker build . --tag patriotframework/virtual-home
```

V případě, že máme Docker[3] nastavení jen s oprávnění pro administrátora, tak zavoláme:

```
sudo docker build . --tag patriotframework/virtual-home
```

a vrátíme se zpátky o úroveň výš:

```
cd ..
```

Po dokončení Docker Image můžeme pokračovat na část patriot-gateway.

6.3.3 Patriot-gateway

Nyní přejdeme do složky modulu patriot-gateway a spustíme kompilaci:

```
cd patriot-gateway/gateway
mvn clean package
```

V dalším kroku si vytvoříme pojmenovaný Docker Image pro modul gateway (případně jako „sudo“):

```
docker build . --tag patriotframework/smart-gateway
```

A vrátíme se zpět do hlavní složky:

```
cd ..
```

6.3.4 Patriot-integration-tests

Jsou-li všechny předchozí kroky úspěšné, pak se můžeme pustit do přípravy modulu s integračními testy. Tato část je velmi důležitá a v případě jakéhokoli problému v průběhu kompilace a nastavování je nutné začít od jeho začátku.

Přejdeme do složky testů:

```
cd patriot-integration-tests
```

[3]Docker - software, který poskytuje jednotné rozhraní pro izolaci aplikací do kontejnerů

Nejdříve si stáhneme Image routeru:

```
docker pull patriotframework/patriot-router:latest
```

Poté spustíme monitorovací prostředí:

```
docker-compose up -d
```

Monitorovací prostředí nám vrátí několik řádek – nejdůležitější z nich je ten první, který nám prozradí název sítě (network), který si musíme poznamenat. Zavoláme:

```
docker network inspect ${ZAZNAMENANY_NAZEV_SITE}
```

Zde nahradíme $ ZAZNAMENANY_NAZEV_SITE za uloženou hodnotu z předchozího příkazu. Název bude nejspíše „integration-tests_default“, ale může být i trochu jiný.

V dalším kroku si musíme zjistit, pod jakou IP adresou nám poběží graylog[4] kontejner s názvem v pravděpodobné podobě „integration-tests_graylog_1“. Projdeme tedy výpis a hledáme hodnotu uloženou pod:

```
"Name": " integration-tests_graylog_1",
// dalsi kod
"IPv4Address": "172.18.0.5/16"
```

Tuto IP adresu si zkopírujeme a zapíšeme do souboru „src/test/resources/-patriot.properties“ jako hodnotu k „io.patriot_framework.monitoring.addr“.
A na závěr spustíme testy:

```
mvn clean test
```

Tímto je všechno nastavené a testy se začínají spouštět.

6.3.5 Problémy a řešení

V případě, že se nějaký z předchozích kroků nepovedl, tak musíme vše vypnout a smazat a celý proces pro patriot-integration-tests modul zopakovat.

[4] Graylog - software umožňující správu logů

Není možné začít s nastavováním od poloviny.

Pokud se něco nezdařilo, tak zastavíme monitoring:

```
docker-compose down
```

Následně smažeme vytvořené kontejnery a sítě:

```
docker rm -f smarthome Internal1 External1 gateway
docker network rm SmartHome GatewayNetwork
```

Po vykonání těchto příkazů můžeme opět začít s nastavováním patriot-integration-tests od začátku.

Kapitola 7

Pokračování projektu do budoucna

Vzhledem k faktu, že je PatrIoT Framework stále předmětem vývoje, je neustále vylepšován a upravován, dovoluji si závěrem přidat několik bodů v podobě doporučení do budoucna.

7.1 Zabezpečení

V aktuální verzi neobsahuje PatrIoT Framework žádné zabezpečení. Jelikož je tento framework zatím zamýšlen jako ukázka možného řešení (simulace IoT systému), včetně jeho testování, tak není důvod k jeho zabezpečování.

V případě, že by se měl framework stát předmětem reálného testování IoT systémů, pak by bylo vhodné ho zabezpečit. Nyní je možné, jen s pouhou znalostí správné URL, nechat si otevřít okno nebo dveře bez jakéhokoli ověření.

7.2 Sjednocení technologií

Každý z modulů frameworku byl implementován jiným člověkem a každý čerpal ze svých znalostí různých technologií, proto je možné v modulech najít různé technologie a přeneseně řečeno i „podpis“ daného vývojáře.

Například v přepisovaném modulu patriot-gateway byly k nalezení technologie Drools[1] pro nastavování cest do domku a MQTT[2] pro posílání zpráv, které byly v rámci této diplomové práce přepsány a nahrazeny pomocí Apache Camel.

V případě patriot-virtual-smart-home se nabízí odstranit SilverWare[3] používaný pro monitorování mikroslužeb[4], který je již přes 3 roky bez aktualizovaných verzí a stává se tak zastaralou technologií, která práci s frameworkem

[1] Drools - technologie pro vytváření business pravidel fungování systému - viz WWW: <https://www.baeldung.com/drools>

[2] MQTT - protokol, který umožňuje přenos zpráv mezi zařízeními

[3] SilverWare - platforma, viz WWW: <https://github.com/SilverThings/SilverWare>

[4] Mikroslužby - procesy, které pro dosažení svého cíle komunikují po síti

neusnadňuje ba spíše naopak.

7.3 Další rozšíření

Posledním, ale ne méně důležitým doporučením, je možnost dalšího rozšíření zařízení/senzorů/akčních členů v jednotlivých systémech.

Nabízejí se zařízení jako chytrá chladnička, mrazák, domácí květiny se samozavlažováním nebo třeba IoT akvárium. Takovéto simulace by mohly ještě umocnit robustnost systémů. Jedná se tedy spíše o otázku dalšího vývoje a směřování projektu.

Kapitola 8

Závěr

Cílem diplomové práce bylo navrhnout sadu příkladů dokumentujících použití frameworku PatrIoT pro automatizované testování IoT systémů a definovat tři fiktivní IoT systémy. Dalším z cílů bylo vytvořit moduly s komponentami a pomocí nich simulovat tři fiktivní IoT systémy. Následně vytvořené moduly využít při sestavování příkladů automatizovaných integračních testů a fiktivní systémy s jejich pomocí otestovat.

Výsledkem práce je tedy navržená sada příkladů popisující tři fiktivní systémy - Chytrý domov, Sklad a Chytré parkování, přičemž jsou vytvořeny moduly pro tyto tři systémy, dále jsou implementovány integrační testy a fiktivní systémy jsou otestovány.

Pro účely snadnějšího pochopení celé problematiky a názorného uvedení do této problematiky byly přiblíženy použité technologie, popsány akční členy, zařízení a senzory a přidány demonstrace kódů v implementační části.

Využitými technologiemi pro zpracování diplomové práce jsou Java ve verzi 8, Apache Maven, Apache Camel a RestAssured.

V rámci zpracování mé diplomové práce byly převzaty a rozšířeny výsledky mého semestrálního projektu. Rozšířením je rešeršní a implementační část a oddíly zaobírající se doporučením do budoucna, instalací a testováním projektu.

Na závěr lze konstatovat, že všech zmíněných cílů bylo dosaženo, a to prostřednictvím vyhotovení sady příkladů a jejich popsáním a otestováním. Rovněž práce splňuje veškeré funkční a nefunkční požadavky včetně požadavků ze zadání diplomové práce.

Literatura

[1] Hanes D. et. al. *IoT Fundamentals: Networking technologies, protocols and use cases for Internet of Things.* Cisco Press 2017.

[2] B. S. Ahmed, M. Bures, K. Frajtak and T. Cerny. *Aspects of Quality in Internet of Things (IoT) Solutions: A Systematic Mapping Study* in IEEE Access, vol. 7, pp. 13758-13780 [online]. 2019 [cit. 2020-12-08]. Dostupné z WWW: <https://ieeexplore.ieee.org/abstract/document/8618341>.

[3] Bureš, Miroslav and Renda, Miroslav and Doležel, Michal and others. *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu.* Grada Publishing as 2016.

[4] P. Giménez, B. Molina, C. E. Palau, and M. Esteve, *SWE Simulation and Testing for the IoT* in IEEE International Conference on Systems, Man, and Cybernetics (SMC), IEEE, 2013, pp. 356–361.[online]. 2013 [cit. 2020-12-18]. Dostupné z WWW: <https://www.researchgate.net/publication/289469345_A_Smart_Home_Network_Simulation_Testbed_for_Cybersecurity_Experimentation>.

[5] H. H. Lee, J. H. Kwon, and E. J. Kim. *FS-IIoTSim: a flexible and scalable simulation framework for performance evaluation of industrial Internet of things systems*The Journal of Supercomputing, pp. 1–18. [online]. 2016 [cit. 2020-12-05]. Dostupné z WWW: <https://www.researchgate.net/publication/310517735_FS-IIoTSim_a_flexible_and_scalable___simulation_framework_for_performance_evaluation_of_industrial_Internet_of_things_systems>.

[6] Kent C. *Static vs Unit vs Integration vs E2E Testing for Frontend Apps.* [online]. 2020 [cit. 2020-12-01]. Dostupné z WWW: <https://kentcdodds.com/blog/unit-vs-integration-vs-e2e-tests>.

[7] PATRIOT - IoT Testing Framework. *PATRIOT - IoT Testing Framework.* [online]. 2018 [cit. 2020-12-10]. Dostupné z WWW: <https://patriot-framework.io/>.

[8] Maven – Introduction to the POM. *Maven – Welcome to Apache Maven.* [online]. 2008 [cit. 2020-12-12]. Dostupné z WWW:

<https://maven.apache.org/guides/introduction/introduction-to-the-pom.html>.

[9] Apache Camel. *Home - Apache Camel.* [online]. Copyright © 2004 [cit. 2020-11-10]. Dostupné z WWW: <https://camel.apache.org/>.

[10] Vývoj koncentrace CO2 v atmosféře. *Fakta o klimatu.* [online]. Copyright © 2020 [cit. 2020-12-12]. Dostupné z WWW: <https://faktaoklimatu.cz/infografiky/koncentrace-co2>.

[11] Rouse, M. *DEFINITION: internet of things (IoT).* [online]. Copyright © 2020 [cit. 2020-12-10]. Dostupné z WWW: <https://internetofthingsagenda. techtarget.com/definition/Internet-of-Things-IoT>.

[12] Statista, Inc. *Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025.* [online]. Copyright © 2019 [cit. 2020-12-05]. Dostupné z WWW: <https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/>.

[13] Stephens M., Rosenberg D. *Testování softwaru řízené návrhem.* Computer Press, a. s. Brno 2011.

[14] Khan, J.Y., Yuce, M.R. *Internet of Things (IoT): Systems and Applications. Jenny Stanford Publishing.* [online]. 2019. ISBN 9780429678059. [cit. 2020-12-01]. Dostupné z WWW: <https://books.google.sk/books?id=DRGwDwAAQBAJ>.

[15] The Apache Software Foundation *HttpClient Overview.* [online]. 2019.[cit. 2020-12-20]. Dostupné z WWW: <https://hc.apache.org/httpcomponents-client-4.5.x/index.html>.

[16] Bera, A. *How many IoT devices are there?.* [online]. 2019.[cit. 2020-12-28]. Dostupné z WWW: <https://safeatlast.co/blog/howmany-iot-devices-are-there>.

[17] SETHI, Pallavi; SARANGI, Smruti R. *Internet of Things: Architectures, Protocols, and Applications.* Journal of Electrical and Computer Engineering. [online]. 2017.[cit. 2020-12-30]. Dostupné z WWW: <https://www.hindawi.com/journals/jece/2017/9324035>.

[18] Internet věcí - IoT. Alza.cz. *Alza.cz – nakupujte bezpečně z pohodlí domova.*| Alza.cz [online]. 2018 [cit. 2020-12-01] Dostupné z: <https://www.alza.cz/internet-veci-iot/18858762.htm>

[19] Pohanka P. *Data Distribution Service IoT - Pavel Pohanka .* [online]. Copyright © 2021 Pavel Pohanka. [cit. 2020-12-31]. Dostupné z WWW: <https://pavelpohanka.cz/internet-of-things/>.

[20] REST Assured Tutorial. *How to test API with Example. Meet Guru99 - Free Training Tutorials & Video for IT Courses.* [online]. Copyright © 2020. [cit. 2020-01-2]. Dostupné z WWW: <https://www.guru99.com/rest-assured.html>.

[21] RAY, Brian. *What is M2M?* [online]. In: LinkLabs. [cit. 2020-01-2]. Dostupné z WWW: <https://www.link-labs.com/blog/what-is-m2m>.

[22] *The Internet of Things (IoT) in Supply Chain and Logistics.* [online]. In: eft, eyefortransport. [cit. 2020-01-1]. Dostupné z WWW: <https://www.eft.com/content/internet-things-iot-supply-chain-and-logistics>.

[23] *ČEZ testuje drony pro kontrolu energetických zařízení.* [online]. [cit. 2020-01-1]. Dostupné z WWW: <http://www.solarninovinky.cz/?zpravy/2017081402/cez-testujedrony-pro-kontrolu-energetickych-zarizeni>.

[24] *DHL a Huawei spolupracují na internetu věcí .* [online]. [cit. 2020-01-2]. 2017 Dostupné z WWW: <https://logistika.ihned.cz/c1-65630510-dhl-a-huawei-spolupracuji-na-internetu-veci>.

[25] Studie IDC. *Výdaje za internet věcí v České republice přesáhnou v roce 2018 miliardu dolarů.* [online]. 2015 In: FOCUS agency. [cit. 2020-01-1]. Dostupné z WWW: <http://www.m-journal.cz/cs/studie-idc–vydaje-za-internet-veci-v-ceskerepublice-presahnou-v-roce-2018-miliardu-dolaru___s288x11436.html>.

[26] Boj o Evropu věcí *Která IoT síť nakonec zvítězí? - Světchytře.cz.* [online]. Copyright © 2018 SocialBooster s.r.o. [cit. 2020-01-2]. Dostupné z WWW: < https://www.svetchytre.cz/a/pUxud/boj-o-evropu-veci-ktera-iot-sit-nakonec-zvitezi>.

[27] Bures, M. (2014). *Automated testing in the Czech Republic: the current situation and issues.* In Proceedings of the 15th International Conference on Computer Systems and Technologies (pp. 294-301). ACM.

Příloha A

Seznam zkratek

API Application Programming Interface

E2E End-to-End

HTTP Hypertext Transfer Protocol

IIoT Industrial Internet of Things

IoT Internet of Things

JSON JavaScript Object Notation

MQTT Message Queuing Telemetry Transport

NFC Near Field Communication

POM Project Object Model

URL Uniform Resource Locator