Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Skladové hospodářství pomocí RFID a Raspberry Pi

Plzeň 2017 Jan Kohlíček

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 26. června 2017

Jan Kohlíček

Poděkování

Děkuji panu Ing. Lukášovi Svobodovi nejen za odborné vedení mé bakalářské práce, ale také za užitečné připomínky a za čas strávený konzultacemi.

Abstract

This thesis deals with possibilities of using a RFID for warehouse management. The part of the thesis is implementation of warehouse system using the RFID and the Raspberry Pi. The storage system consists of three parts of the server, the RFID reader and the mobile application. The server is built on Node.js and communication with other parts is ensured by the protocol MQTT and REST API. Information about warehouse is stored in database MongoDB. The RFID reader is used to add and remove warehouse items. A wiring diagram is available to build a reader, programming was done in Python. Mobile apps for Android are used to manage malt system, and NFC has been used to quickly retrieve warehouse item information.

Abstrakt

Práce se zabývá možnostmi využití RFID pro skladové hospodářství. Součástí práce je implementace skladového systému s užitím RFID a Raspberry Pi. Skladový systém se skládá ze tří částí: serveru, čtečky RFID a mobilní aplikace. Server je postavený na Node.js a komunikaci s dalšími částmi zajišťuje protokol MQTT a REST API. Informace o skladu jsou ukládány do NoSQL databáze MongoDB. Čtečka RFID slouží k přidávání a odebírání skladových položek. Pro sestrojení čtečky je k dispozici schéma zapojení a programování probíhalo v jazyce Python. Mobilní aplikace pro Android slouží ke správě skladového systému, NFC se využívá k rychlému načtení informací o položce skladu.

Obsah

1 Úvod

Nástupem čtvrté průmyslové revoluce neboli průmyslu 4.0 se očekává plná automatizace řídících systémů co nejvíce nezávislých na lidské obsluze.

Na průmyslové úrovni má jít o nahrazení manuální lidské práce robotizací, současné „manuální“ zadávání dat a postupů má být nahrazeno automatickým předáváním informací mezi materiály, polotvary a jednotlivými stroji, sklady atd. prostřednictvím zapisovatelných RFID čipů umístěných na každém materiálu a produktu. Ve vnitropodnikovém prostředí má být lidská síla v přepravě komponent a materiálů v rámci výrobního procesu nahrazena automatizovanými dopravními prostředky od skladů jednotlivých materiálů až po sklad finálních výrobků.

V oblasti mezifiremní přepravy a dopravy dojde k automatickému řízení logistiky opět na základě sdílení dat pomocí RFID čipů materiálů a produktů, k předávání potřebných dat bude docházet mezi databázovými systémy běžícími na různých internetem propojených serverech. Všechny tyto změny budou hlavně iniciovány touhou ušetřit náklady a ještě více zefektivnit a zrychlit výrobu.

Cílem této práce je zjistit využití technologie RFID pro skladové hospodářství a vytvořit skladový systém s RFID a Raspberry Pi. Skladový systém se bude skládat ze serveru, klientské části s uživatelským rozhraním a samotného RFID zařízení.

2 Skladové hospodářství

Skladové hospodářství je nedílnou součástí logistického systému. Sklady mají za úkol přijímat zásoby, uchovávat a vydávat je a provádět potřebné skladové manipulace. Skladové hospodářství má v podnicích návaznost na téměř všechny ostatní úseky. Hlavním úkolem skladu je ekonomické sladění rozdílně dimenzovaných toků v podniku, a to tak, aby bylo dosaženo synergického efektu.[11]

2.1 Současný skladový systém

Pod tímto pojmem si můžeme představit systém pro evidenci skladu resp. skladovaných zásob (příjem a výdej materiálu), systém pro účtování zásob a jejich objednávek nebo systém pro inventarizaci. Můžeme tedy říci, že skladový systém je systém pro správu ekonomických operací nad skladem.[4]

2.2 Skladový systém s RFID

2.2.1 Rychlé načtení údajů

RFID čip má oproti etiketě s čárovým kódem dvě hlavní výhody - rychlost čtení a nepřímou viditelnost čtecího zařízení na čip. Současné standardy UHF RFID čipů umožňují načíst najednou až 1000 čipů za sekundu, tato hodnota se však s příchodem novějších a výkonnějších zařízení bude zvyšovat. RFID čtecí zařízení nepotřebuje mít přímou viditelnost na jednotlivé čipy, čtení i zápis probíhá bezdrátově a to do vzdálenosti cca 15 m u pasivních čipů a až 100 m u aktivních čipů.[2]

Například paletový přepravník tak může projet celým RFID čtecím portálem a v jeden čas dojde k současnému načtení všech čipů na paletě, tím se dosáhne zrychlení procesu příjmu, výdeje, přesunu a inventarizace produktu.[2]

2.2.2 Odstranění chyb obsluhy

RFID čipy společně se čtecím zařízením vylučují možnost vzniku chyby obsluhy, které vzniknou například tím, že obsluha načte pouze část čárových kódů na paletě.[2]

2.2.3 Zápis údajů o zboží během celého logistického pohybu

RFID čip má oproti etiketě s čárovým kódem hlavní výhodu v tom, že do čipu lze informace i zapisovat a nejenom číst, jak je to v případě čárového kódu. Tato vlastnost bude v budoucnosti klíčová a rozhodne v mnoha odvětvích pro úplnou náhradu čárového kódu RFID čipem. Do čipu lze navíc informace zapisovat a měnit opakovaně, lze takto do každého produktu zapsat datum výroby a poté také připsat jednotlivé logistické zápisy, které vznikají po celou dobu cesty produktu.[2]

2.2.4 Přesná evidence spotřebitelských jednotek

V současnosti při samotném logistickém procesu obsluha načte čárový kód palety, ale již není schopna ověřit, zda je na paletě správný počet kartónů a správný počet produktů. Jediným řešením by bylo paletu rozebrat a postupně načíst všechny čárové kódy. RFID čtecí portál však načte najednou všechny RFID čipy na paletě nalezené. Navíc dle typu čipu dokáže vyhodnotit počet RFID čipů kartónů i počet RFID čipů samotných produktů.[2]

2.2.5 Odolnost RFID čipů

Etiketa s čárovým kódem podléhá teplotním a povětrnostním vlivům a následně dochází k poškození etikety. Je tomu hlavně proto, že je nutné etikety s čárovým kódem umisťovat tak, aby je bylo možné načíst čtecím zařízením a tudíž zvenku. RFID čip je umístěn uvnitř produktu nebo balení a tím je odolný jak proti teplotě, vodě i povětrnosti. V současné době na trhu již existují RFID čipy, které navíc mohou obsahovat čidla - například pro měření vlhkosti nebo teploty. [2]

2.2.6 Optimalizace skladových zásob

Představme si tak obvyklou záležitost, jakou je příjem materiálu (zboží) a jeho naskladnění. Tato operace dnes probíhá v mnoha společnostech po jednotlivých kusech (logistických jednotkách), a tak se také informace dostávají do informačního systému (se zpožděním). Celá došlá zásilka je načtena RFID čtečkami během několika sekund a tato informace (násobně větší objem) se přenáší do informačního systému, který dosud nebyl na tento způsob zpracování informací připraven. Výsledkem tohoto naskladnění je okamžitá informace o stavu našeho skladu, a dramatické zrychlení jejího získání. Vybavíme-li stejnou technologií také výrobu ve společnosti, získáme

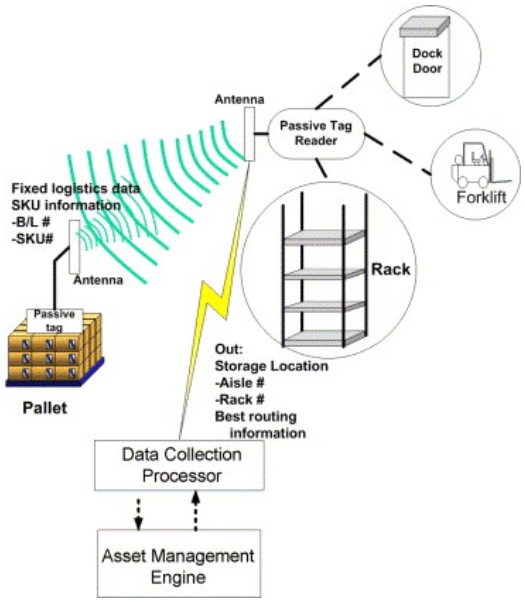
tak stejné informace z jednotlivých částí výroby (stav výroby, průběžné zásoby na pracovišti). Dramatické zrychlení sběru informací v rámci logistiky (výroby) umožňuje daleko lepší plánování zásob společnosti. Pokud máme plán výroby a stavy zásob, tak zásoby, které máme na skladě můžeme odpovědně řídit ne z hlediska `Q (množství)`, ale z hlediska `T (času)`. Tím se zcela zásadním způsobem zjednodušuje systém predikce objednávek a otevírá se velký prostor pro optimalizaci skladových zásob - úspory na vázaném kapitálu.[2]

2.3 Popis systému RFID-RMS

Systém `RFID-RMS` se používá pro poskytovatele logistických služeb ke zlepšení skladových operací pomocí sledování a optimalizace využitých zdrojů. `RFID-RMS` využívá mobilní technologie k přesnému určení lokalizace, sledování a řízení zdrojů v prostředí skladu. Architektura systému `RFID-RMS` se skládá ze dvou částí, které přispívají k rozhodovacímu procesu v oblasti řízení zdrojů. První část je `frontend`, obsahující dva moduly pro sběr dat, a to modul pevných logistických dat a modul proměnných logistických dat. Tyto dva moduly obsahují dva různé typy RFID tagů pro ulehčení přenosu a ukládání logistických dat. Druhá část je `backend`, který obsahuje modul sledování zdrojů, modul řízení zdrojů a datového úložiště. Primární funkcí této části je provádět řízení zdrojů, sledování zdrojů, hodnotit využití zdrojů, výběr nejvhodnější trasy, údržba a kontrola provozu.[1]

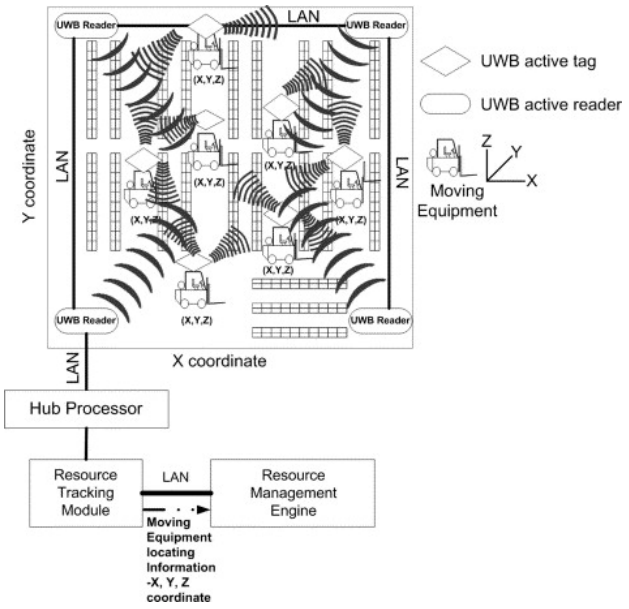
2.3.1 Sběr dat

`Modul pevných logistických dat` používá nízkoúrovňové `radiové` signály pro výměnu dat mezi pasivními tagy a čtečkou. Pasivní tag se skládá z `integrovaného obvodu` pro uložení identity položek a dalších informací. Jak je znázorněno na obrázku 2.1, pasivní značka je připojena na položky, jako jsou palety, obaly atd. Čtečky tagů s pevnou pozicí antén jsou namontovány do konstrukce jako jsou dveře, vstupní brána nebo integrované do vysokozdvižných vozíků a ostatního vybavení. Obsahuje-li více antén, dokáží v rámci svého rozsahu najednou rozpoznat a číst stovky tagů. Dále budou přijaté signály dekódovány na data a dál poslány pomocí datového připojení k serveru, na kterém běží obchodní logika. Výměna dat mezi čtečkou a řízením zdrojů je zprostředkována prostřednictvím bezdrátové sítě `LAN`.[1]



Obrázek 2.1: Modul proměnných logistických dat[1]

K `modulu proměnných logistických dat` se vztahují ultra-širokopásmové technologie `(UWB)`, které definují přenos proměnných logistických dat mezi čtečkou a tagem, jak je znázorněno na obrázku 2.2. Tento modul obsahuje kolekci aktivních tagů, čtyři čtečky `UWB` a `Hub procesor` pro sledování polohy zdrojů. `UWB` aktivní tagy se skládají z interní baterie a krátkého `impulsového` vysílače umožňujícího vysílat na mnohem delší vzdálenost. Tag vydá krátkodobý `pulzní` signál několikrát za sekundu. `UWB` čtečky přijímají signály a posílají je do Hub procesoru. S logickým nastavením `triangulace` rozdílného čtení `UWB` čteček lze vypočítat přesné (x, y, z) souřadnice aktivního tagu. Přitom je koordinace zdrojů ve skladu přesně vypočítaná. Takové koordinace zdrojů budou předány `sledovacímu modulu zdroje` pro následné zpracování.[1]



Obrázek 2.2: Modul proměnných logistických dat[1]

2.3.2 Sledování

`Modul sledování zdrojů` je server, který uloží všechny údaje aktivních tagů a poskytuje výkonné prostředí pro manipulaci s daty, filtruje a předává veškeré užitečné údaje řízení zdrojů pro formulování obchodní logiky a rozhodování. To je základem pro řízení zdrojů pro provádění činností včetně řízení zdrojů, plánování zdrojů, využívání a měření výkonnosti. Kromě výše uvedených funkcí, modul sledování je schopný převést data `UWB` na (x, y, z) souřadnice, které pak poskytují v reálném čase vizuální zobrazení přesného umístění zdroje na mapě skladu.[1]

2.3.3 Řízení

`Modul řízení zdrojů` zpracovává data z aktivních tagů v reálném čase. Na základě těchto informací hledá optimální trasu a délku cesty k vyzvednutí zdrojů každým potenciálním manipulačním zařízením. V důsledku toho je vybráno zařízení nejvhodnější pro provedení úkolu.[1]

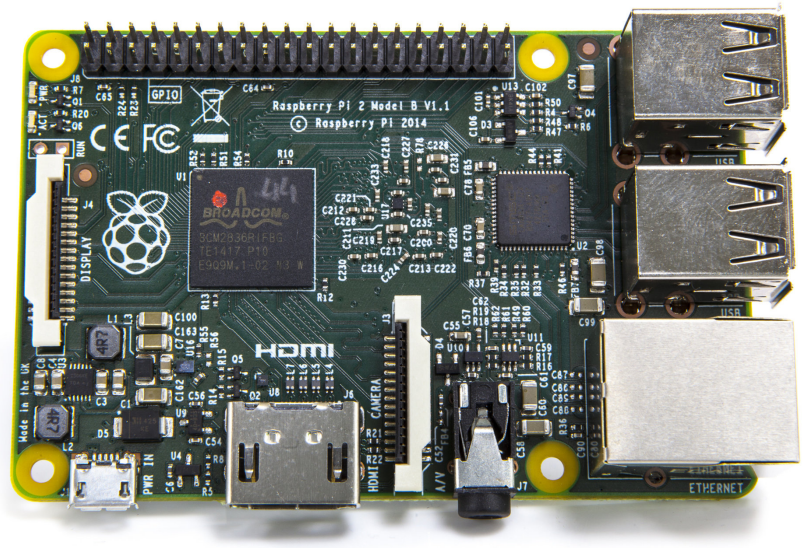
3 Popis zařízení

3.1 Raspberry Pi

Raspberry Pi (RPi) je řada malých jednodeskových počítačů, vyvíjená ve Velké Británii. Pro tento projekt se použije Raspberry Pi 2 Model B (viz obrázek 3.1), číslovka v názvu určuje generaci a model B je osazen Ethernetovým portem na rozdíl od modelu A. RPi má procesor Broadcom BCM2836 ARM Cortex-A7 Quad Core 700 MHz s možností přetaktovat na 900 MHz, 1GB RAM, 4x USB 2.0, HDMI, 4-pólový jack a již zmiňovaný 10/100 Ethernet. Přenosová rychlost ethernetu je velmi omezená, protože je napojený na USB řadič. V slotu MicroSDHC musí být microSD karta, protože z ní bootuje operační systém.

Na desce jsou ještě umístěny 2 řady pinů, takzvané GPIO (viz obrázek 3.2).

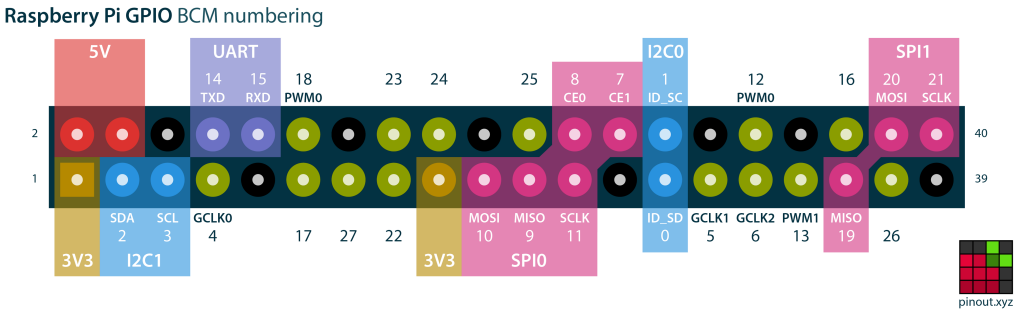
PRi neobsahuje RTC (Real Time Clock), získávání aktuálního času se řeší pomocí NTP (Network Time Protocol).



Obrázek 3.1: Raspberry Pi 2 Model B

3.1.1 GPIO

`GPIO (General Purpose Input/Output)` je `40 pinů` umístěných na desce ve dvouch řadách. Číslování pinů je dvojího typu `BCM` a `BOARD`, u `BCM` se počítají jen nastavitelné piny, kdežto u `BOARD` se počítají všechny. Mezi nenastavitelné piny patří napájení `5V` (**červená**), napájení `3V3` (**oranžová**) a `zem` (**černá**) (viz obrázek 3.2).



Obrázek 3.2: Schéma `GPIO`

3.1.2 Operační systémy

Raspbian

`Raspbian` je operační systém založený na linuxové distribuci `Debian` a optimalizován pro hardware `Raspberry Pi`. Systém je vydáván ve dvou verzích Pixel a Lite, verze Pixel obsahuje navíc `GUI` a balíčky pro vývoj, což se promítlo na celkové velikosti 4 GB oproti Lite verzi s 1,5 GB. Pro většinu hlavních programujících jazyků je k dispozici knihovna pro ovládání `GPIO`.

Windows 10 IoT Core

Verze `Windows 10 IoT Core` je určena pro minipočítače typu `Raspberry Pi`. Systém je v raném vývoji a je na stránkách Microsoftu k dispozici zdarma. Systém nemá uživatelské rozhraní. Spravuje se přes webové rozhraní, `Powershell` nebo na něm běží aplikace, která uživatelské rozhraní může mít. Spustit lze pouze `Universal Windows Platform (UWP)` aplikace, které je možné psát v `c#`, `c/c++`, `Pythonu`, `Node.js` a `javascriptu`. Pro vývoj a nahrání `UWP` aplikace je zapotřebí mít `Visual Studio 2015`.

3.2 RFID

RFID představuje technologii identifikace pomocí radiofrekvenčních vln. Jedná se v podstatě o generaci identifikátorů navržených (nejen) k identifikaci zboží, navazujících na systém čárových kódů. Identifikace a dohledatelnost je možná celosvětově, a to při dodržení standardu dat EPC Global a s využitím internetového rozhraní EPC Global Network. EPC o délce 96 bitů nabízí dostatečný číselný prostor 268 milionům výrobců produkujícím každý 16 milionů druhů výrobků (tříd), přičemž v každé třídě je prostor pro 68 miliard sériových čísel.[2]

3.2.1 Základní princip

Technologie RFID pracuje na principu radaru. Transpondéry (tagy) mohou být jak aktivní, tak pasivní. Čtečka nejprve vysílá na svém nosném kmitočtu elektromagnetickou vlnu, která je přijata anténou pasivního transpondéru. Indukované napětí vyvolá elektrický proud, který je usměrněn a nabíjí kondenzátor v transpondéru. Uložená energie je použita pro napájení logických a rádiových obvodů transpondéru. Když napětí na kondenzátoru dosáhne minimální potřebné úrovně, spustí se logický automat či mikroprocesor a transpondér začne odesílat odpověď čtečce. Vysílání transpondéru je realizováno zpravidla pomocí dvoustavové ASK (Amplitude Shifting Key) modulace, která je realizována změnou zakončovací impedance antény transpondéru. Odrazy, které vznikají změnou impedance antény, jsou detekovány čtečkou do binární podoby. Řízení komunikace a jednotlivých stavů komunikačního řetězce je definováno příslušnou ISO normou.[2]

3.2.2 Forma tagu

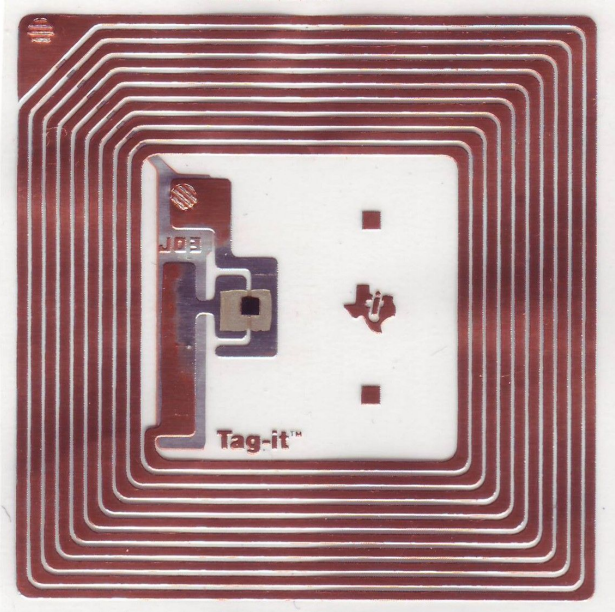
RFID tag - paměťový radiofrekvenční čip nesoucí datovou informaci, který komunikuje bezkontaktně a bez přímé viditelnosti se snímačem - nejčastěji v podobě etikety nebo štítku. Provedení (tvar, rozměry, materiál) se mohou velmi lišit dle požadavků aplikace. RFID tag se skládá z vlastního čipu, antény, propojení a zapouzdření, případně baterie. Čip definuje kapacitu a typ RFID tagu, anténa stanovuje kvalitu příjmu a odesílání radiofrekvenčního signálu, zapouzdření ovlivňuje možnost použití v různých prostředích a životnost tagu.[2]

RFID Smart label - čip je umístěn na potisknutelné etiketě s možností dalších informací (text, čárový kód) viz obrázek 3.3

RFID wristband - náramek na ruku obsahující RFID čip, využití ve zdravotnictví k identifikaci osob.

RFID karta - čip může být zapouzdřen do plastové karty nebo předmětu typu klíčenky – např. k využití v platebních a docházkových systémech.

RFID inlay - zabudování čipu přímo do produktu, v případě kovového výrobku možnost oddělující vrstvy kvůli rušení.



Obrázek 3.3: Pasivní HF RFID

3.2.3 Frekvenční pásma

Systémy RFID pracují s různými frekvencemi, přičemž frekvence ovlivňuje rychlost čtení a zápisu, dosah signálu a prostor pokrytí atd. Více informací v tabulce 3.1

Frekvence	**Dosah**	**Popis**
Nízká frekvence (LF) 125–134 kHz	0,5 m	krátký dosah, velká anténa, pouze pro čtení, nízká přenosová rychlost, kovy a kapaliny nevadí
Vysoká frekvence (HF) 13,56 MHz	1 m	krátký dosah, velká anténa, pouze pro čtení, kapaliny znesnadňují čtení
Velmi vysoká frekvence (UHF) 860-930 MHz	3 m	možnost číst i zapisovat, vysoká přenosová rychlost, nelze číst přes kapaliny
Mikrovlnná frekvence (MW) 2,54 a 5,8 GHz	10 m	možnost číst i zapisovat, vysoká přenosová rychlost, kapaliny a kovy příliš nevadí

Tabulka 3.1: Frekvenční pásma RFID

3.2.4 Zdroje napájení

Čipy (tagy) se dělí na aktivní a pasivní podle toho, zda je možné informace z nich nejen číst (pasivní), ale i do nich zapisovat (aktivní). Aktivní jednotky pak musí disponovat vlastním zdrojem energie, ten obstarává miniaturní baterie.[2]

Pasivní

Pasivní zdroje jsou nejrozšířenější, nemají vlastní baterii, napájeny jsou polem snímače. Ten periodicky vysílá pulsy prostřednictvím antény do prostoru, čip využije přijímaný signál k nabití svého napájecího `kondenzátoru` a vyšle odpověď. Pasivní tagy mají různou vzdálenost čtení od 0,5 m do 10 m, dlouhou životnost čipu a používají metodu `RTF (reader talk first)`. V současné době jsou nejvíce rozšířeny pasivní čipy a to zejména kvůli své nenáročnosti na obsluhu a odolnosti, velikost paměti 64-256 bitů.[2]

Aktivní

Aktivní zdroje mají vlastní baterii, jsou schopny vyslat svoji identifikaci. Používají se méně často než pasivní systém `RFID`. Jsou složitější, obsahují navíc i zdroj napájení a jsou schopny samostatně vysílat své identifikace - používají se proto pro aktivní lokalizaci. Aktivní čipy vysílají samy své údaje do okolí `TTF (tag talks first)`, to umožňuje vlastní miniaturní baterie umístěná v čipu, která vydrží cca 1-5 let. Tyto čipy však kvůli baterii mají menší odolnost na teplotu a je nutné provádět výměnu baterie. Aktivní čipy mají vzdálenost čtení až 100 m, velikost paměti na čipu může dosahovat až 100 kb.[2]

3.2.5 Čtečka

Snímače neboli čtečky `RFID` jsou zařízení, která dokáží zachytit vysílání aktivního nebo pasivního tagu. Čtečka nemusí pouze informace zachycovat, ale také může do tagu zapisovat. Čtečka používá pro vysílání a přijímání signálu anténu, která může být integrovaná nebo externí. Základním požadavkem na čtečku je schopnost zpracovat obrovské množství dat. Čtečky musí poznat již jednou přečtené tagy a odstranit odrazy signálů tagů od pevných překážek a musí zvládnout současně načíst velký počet tagů. S tím souvisí schopnost paralelně načítat tagy v relativně krátkém časovém intervalu.[2]

4 Technologie

4.1 Python

Python je interpretovaný, interaktivní, objektově orientovaný programovací jazyk. Poskytuje vysokoúrovňové datové struktury, jako jsou seznam a asociativní pole, dynamické zadávání a dynamické vazby, moduly, třídy, výjimky, automatické řízení paměti atd. Má pozoruhodně jednoduchou a elegantní syntaxi a přesto je univerzálním programovacím jazykem. Byl navržen Guido van Rossum v roce 1990. Stejně jako mnoho jiných skriptovacích jazyků je to zdarma, a to i pro komerční účely, a může být provozováno prakticky na jakémkoliv moderním počítači. Python program je automaticky překládán interpretem do nezávislého bajtového kódu platformy, který je pak interpretován.[9]

Python je modulární. Jádro je velmi malé a může být rozšířeno importováním rozšiřujících modulů. Distribuce Pythonu zahrnuje rozmanitou knihovnu standardních rozšíření (některé jsou napsány v jazyce Python, jiné v jazyce C nebo C ++) pro operace od manipulací s řetězci a regulárních výrazů atd. Tyto rozšiřující moduly jsou někdy označované jako „balíčky“ nebo komponenty.[9]

4.2 Java

Jazyk Java byl vytvořen vývojářským týmem firmy Sun Microsystems pod vedením Jamese Goslinga. Java je objektově orientovaný jazyk, jehož syntaxe je podobná C nebo C++. Java je interpretovaný jazyk - to znamená, že program se nepřekládá přímo do strojového kódu, ale do takzvaného bajtkódu, který je následně interpretován virtuálním strojem Java. To zabezpečuje, že Java je na platformě nezávislý jazyk. O správu paměti se stará automatický Garbage Collector, jenž zabezpečuje, aby objekty, které se už nepoužívají, byly odstraněny. Problém přestavuje skutečnost, že nevíme, kdy bude spuštěn.[6]

4.3 Node.js

Node.js je JavaScript v serverovém prostředí. Je založen na implementaci V8 enginu od Googlu. V8 a Node.js jsou většinou implementovány v

C a C ++ se zaměřením na výkon a nízkou spotřebu paměti. Ale zatímco V8 podporuje hlavně JavaScript v prohlížeči, Node.js je zaměřen na podporu dlouhodobých serverových procesů. Na rozdíl od většiny ostatních moderních prostředí, proces Node.js se nespoléhá na multithreading, který podporuje souběžné provádění logiky. Je založen na asynchronním modelu události I/O. Přemýšlejte o procesu serveru Node.js jako o jednovláknovém démonu, který vloží do JavaScriptu podporu přizpůsobení. To se liší od většiny událostních systémů pro jiné programovací jazyky, které přicházejí ve formě knihoven. Node.js podporuje model události na jazykové úrovni. JavaScript je vhodný pro tento přístup, protože podporuje zpětná volání událostí. Například, když prohlížeč úplně načte dokument, uživatel klepne na tlačítko nebo je splněn AJAX požadavek, událost vyvolá zpětné volání. Funkčnost přirozeného jazyka JavaScript usnadňuje vytváření anonymních funkcí, které si můžete zaregistrovat jako obslužné události.[10]

4.4 JSON

JSON je navržen tak, aby byl jazykově nezávislý, čitelný pro lidi a pro počítače snadno zpracovatelný. JSON je přímo podporován v jazyce JavaScript a je nejvhodnější pro aplikace v JavaScriptu. Tím je zajištěno značné zvýšení výkonu o proti XML, které vyžaduje další knihovny, aby načetly data z objektu DOM (Document Object Model). Odhaduje se, že JSON v moderních prohlížečích zpracovává až stokrát rychleji než XML. Argumenty proti JSONu zahrnují nedostatek podpory jmenného prostoru, nedostatek ověřování vstupů a nedostatky rozšiřitelnosti. Ověření vstupů je odpovědností jednotlivých aplikací a nedostatek rozšíření je řešen flexibilitou konstrukcí JSON. Syntaxe JSONu je člověku čitelná.[8]

Ukázka JSON formátu k zakódování jména a příjmení:

```
{firstname: 'Jan', lastname: 'Kohlíček'}
```

4.5 MQTT

MQTT je jednoduchý a nenáročný protokol pro předávání zpráv mezi klienty prostřednictvím centrálního bodu – brokeru. Díky této nenáročnosti a jednoduchosti je snadno implementovatelný. Navržen byl v IBM, dnes za ním stojí Eclipse foundation.[7]

U protokolu MQTT probíhá přenos pomocí TCP a používá návrhový vzor publisher – subscriber. Tedy existuje jeden centrální bod (MQTT broker),

který se stará o výměnu zpráv. Zprávy jsou tříděny do tzv. témat (topic) a zařízení buď publikuje v daném tématu (publish), to znamená, že posílá data brokeru, který je ukládá a distribuuje dalším zařízením, nebo je přihlášeno k odběru tématu či témat (subscribe), a broker pak všechny zprávy s daným tématem posílá do zařízení. Jedno zařízení samozřejmě může najednou být v některých tématech publisher, v jiných subscriber.[7]

Samotný obsah zpráv není nijak daný ani vyžadovaný, MQTT je payload agnostic. Obsah zprávy jsou nějaká binární data, která jsou přenesena. Nejčastěji se používá JSON, BSON, textové zprávy. Velikost zprávy je v aktuální verzi protokolu omezena na necelých 256 MB.[7]

MQTT minimalizuje množství servisních dat. Zavádí tři úrovně QoS:

QoS 0 - Nejnižší úroveň znamená, že zpráva je odeslána bez potvrzení a není zaručeno její doručení (At most once)

QoS 1 - Prostřední úroveň říká, že zpráva je doručena alespoň jednou (At least once)

QoS 2 - Nejvyšší úroveň znamená, že každá zpráva je doručena právě jednou (Exactly once)

Pro zajištění zabezpečení může MQTT broker vyžadovat ověřování pomocí uživatelského jména a hesla. Šifrování komunikace může být řešeno pomocí SSL. Standardní port pro MQTT je 1883, pro komunikaci MQTT přes SSL je výchozím portem 8883.[5]

4.6 REST API

REST není skutečným protokolem, ale architekturou pro API. Poprvé tuto architekturu představil Roy Fielding v roce 2000, a od té doby je široce používána.[5]

REST v protokolu HTTP používá metody GET, POST, PUT a DELETE poskytující systém pro zasílání zpráv orientovaný na zdroje, kde lze všechny akce provádět jednoduše pomocí asynchronních příkazů HTTP požadavek/odpověď. Používá vestavěnou hlavičku protokolu HTTP, která označuje formát dat v obsahu. Typ obsahu může být XML nebo JSON a závisí na serveru HTTP a jeho konfiguraci. REST je již důležitou součástí IoT, protože je podporována všemi komerčními cloudovými platformami M2M. Navíc může být snadno implementována v mobilních aplikacích, protože vyžaduje pouze knihovnu HTTP,

která je dostupná pro všechny distribuce operačních systémů. Funkce HTTP mohou být plně využity v architektuře REST včetně cachování, autentizace a požadování typu obsahu.[5]

RESTful služby využívají bezpečný a spolehlivý protokol HTTP, který může využívat protokol TLS/SSL pro zabezpečení. Dnes však většina komerčních M2M platforem nepodporuje požadavky HTTPS. Namísto toho poskytují jedinečné ověřovací klíče, které musí být v záhlaví každé žádosti, aby bylo dosaženo jisté úrovně zabezpečení.[5]

I když je REST již v komerčních platformách M2M široce využíván, je nepravděpodobné, že se stane dominantním protokolem kvůli tomu, že nebude snadno implementovatelný. Používá protokol HTTP, což znamená, že není kompatibilní s omezenými komunikačními zařízeními. Toto ponechává jeho použití pro konečné aplikace.[5]

Vzhledem k současné tendenci používání REST na mobilních aplikacích, ovlivňuje dodatečná režie spojená s protokolem požadavek/odpověď výdrž baterie. Režie se stane zbytečnou při nepřetržitém volání a dlouhém dotazování na hodnoty, zejména v případě, že nejsou k dispozici nové aktualizace.[5]

4.7 MongoDB

MongoDB je dokumentově orientovaná databáze. Dokument je v MongoDB základní datovou jednotkou, srovnatelnou s řádkem v relačních databázích. Jedná se o semistrukturované dokumenty vybavené indexy. Dokumenty jsou seskupovány do kolekcí, které se podobají tabulkám relačních databází, ale nemají pevně dané schéma. Protože kolekce neomezují schéma, mohou být seskupovány libovolné dokumenty v jedné kolekci. Dokumenty v kolekci by ale měly být podobné, aby bylo možné efektivní indexování. Kolekce se seskupují do databází, které jsou uloženy jako soubory v operačním systému. Jedna instance MongoDB může spravovat několik databází, které jsou zcela nezávislé.[3]

Dokumenty jsou ukládány ve formátu BSON. BSON je binární reprezentace JSON formátu. BSON formát je bohatší než JSON formát a podporuje další datové typy, jako regulární výrazy, binární data nebo datum. Každý dokument má unikátní identifikátor, který je zadán uživatelem při vytváření dokumentu nebo je vytvořen MongoDB.[3]

Vývoj MongoDB je zaměřen na čtyři hlavní oblasti:

Flexibilita - MongoDB ukládá data jako dokumenty ve formátu JSON, který poskytuje datový model, jež se snadno mapuje na datové typy pro-

gramovacích jazyků. Jelikož se jedná o datový model bez schématu, je jeho použití daleko snadnější než u relačních databází.[3]

Funkcionalita - `MongoDB` poskytuje mnoho z funkcionalit relačních databází, jako jsou sekundární indexy, třídění, agregace apod.[3]

Rychlost - Automatické dělení do fragmentů a ukládání souvisejících dat pohromadě umožňuje snadné horizontální škálování databáze a tím zvyšování výkonu bez nutnosti přerušení provozu.[3]

Snadné použití -`MongoDB` je vyvíjena pro snadnou instalaci, konfiguraci, údržbu a provoz. Obsahuje jen málo konfiguračních parametrů a kdekoli je to možné, jsou nastavení prováděna automaticky.[3]

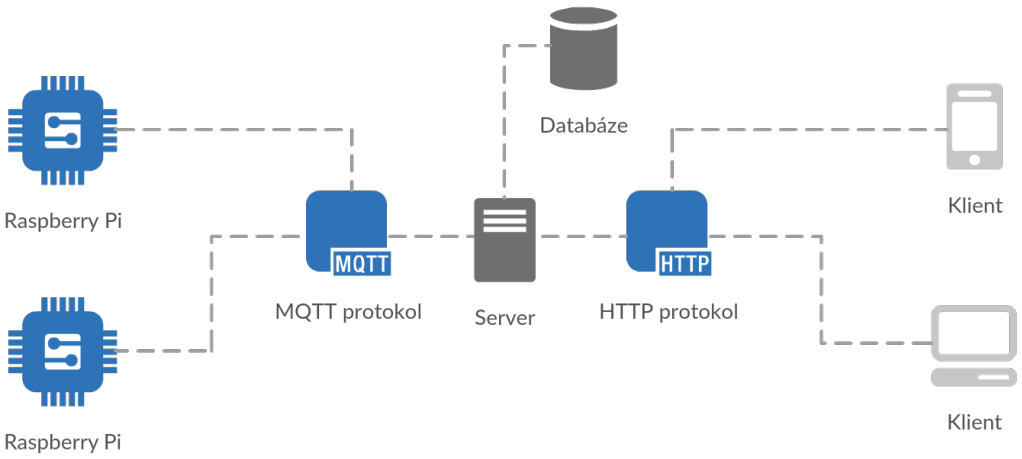
5 Návrh řešení

Skladový sytém se skládá ze tří propojených částí. První z nich je server zajišťující komunikaci mezi ostatními částmi a databází. Další částí je čtečka `RFID`, jejímž úkolem je umožnit skladníkům přidat a odebrat položky ze skladu. Poslední částí je klient umožňující uživatelům správu skladu.

5.1 Server

Ze dvou možností jsem zvolil architekturu `client-server`. Hlavní výhodu vidím v tom, že na rozdíl od `peer-to-peer`, je veškerá logika soustředěná na jednom místě. Také při výskytu chyby stačí aktualizovat jen server, než aktualizovat velké množství zařízení. Dalším přínosem je možnost server přesunout na `cloud`.

Server bude postavený na serverovém frameworku `Node.js`, jehož hlavní výhodou je jednoduchá práce s daty ve formátu `JSON`, také dokáže rychle zpracovat ohromné množství požadavků. Formát `JSON` se hodí pro tvorbu `REST API` nebo komunikaci s `NoSQL` databází. Komunita kolem `Node.js` vytváří spoustu užitečných modulů, které jsou k dispozici na `npmjs.org`. Tento framework má již rozsáhlou podporu mezi komerčními `cloudovými platformami`.



Obrázek 5.1: Návrh komunikace

5.1.1 Komunikace s čtečkou RFID

Protokol MQTT je komunikačním standardem pro IoT. Jeho nenáročnost se projevuje v několika směrech. Jedním z nich je prodloužená výdrž baterie, což se využije tehdy, pokud bychom chtěli napájet čtečku RFID z baterie. Tím druhým je, že při zavádění velkého množství skladových položek se nepřetíží počítačová síť. Protokol MQTT je podporován výrobci různých zařízení, která by mohla být přínosem pro skladový systém. Takto bude skladový systém připraven na případné rozšíření. Všechny tyto důvody vedly k tomu, že jsem si tento protokol vybral pro komunikaci se čtečkou RFID.

5.1.2 Komunikace s klientem

Nejlepší volbou je otevřené REST API, jehož široká podpora umožní vznik různým aplikacím. Také v budoucnu se API stane přínosem při importu a exportu dat skladového sytému. Nelze vyloučit ani přístup jiných systémů, které budou potřebovat k provozu skladový sytém.

5.2 Čtečka RFID

Pro Raspberry Pi jsem zvolil operační systém Raspbian Jessie Lite, protože je přímo vyvíjen výrobcem. Tím je zajištěna nejlepší kompatibilita a stabilita. Systém má širokou komunitu, při vyskytnutí problému bude s velkou pravděpodobností již existovat řešení.
Aplikace je vyvíjena v programovacím jazyku Python, protože je součástí systému spolu s balíčkem RPi.GPIO. Tento balík obsahuje třídy pro řízení GPIO.

5.2.1 Přidat a odebrat položku

Přidávání a odebírání položek skladu lze vyřešit dvěma způsoby:

- Prvním řešením je využít dvě čtečky RFID-RC522. Jedna čtečka by sloužila pro přidávání a druhá pro odebírání položek skladu. Společně s UID tagu by se posílal i typ akce na server. Nevýhodou tohoto řešení je zvýšení ceny celkového zařízení a zbytečná logika na zařízení. Pozitivum lze vidět v paralelním přidávání a odebírání položek.

- Druhým řešením je mít jen jednu čtečku RFID-RC522 a přepínat mezi režimy přidat a odebrat. To otevírá několik otázek: Kde ukládat aktuální režim? Jak přepínat režim? Jak dát najevo aktuální režim?

Úložiště režimu - První, co se nabízí, je ukládat režim na zařízení a serveru posílat `UID` tagu s aktuálním režimem. Mínusem je zbytečně přidaná logika na zařízení, které se v případě nutnosti bude obtížně aktualizovat. Dalším úložištěm režimu se logicky jeví server, což by přineslo možnost vzdáleně režim měnit. Nevýhodou by se stala nutnost posílat aktuální režim zařízení.

Přepnutí režimu - Přepínání by bylo možné pomocí tlačítka na zařízení, ale nevýhodou tohoto řešení je hygiena a výdrž mechanických částí. Druhou možností je vyhradit si tag, který by sloužil k přepínání režimu. V neprospěch tohoto řešení je nutnost mít pro každého skladníka tag pro přepínání. Poslední způsob je přepínat vzdáleně pomocí klienta, což přináší více negativ než pozitiv, hlavně z těchto důvodů: nutnost mít klienta pro každého skladníka, časová náročnost na přepnutí a možné konflikty spojené se vzájemným přepínáním režimu.

Zobrazení režimu - Aktuální režim jde zobrazit třemi způsoby: `LED diodou`, bzučákem, nebo `LCD` displejem s rozlišením 16x2 znaků. Jejich samostatné použití je velmi nedostatečné a pro skladníka nepřívětivé.

Ideální by byla kombinace všech těchto komponentů. Zde je příklad jejich využití: Skladník má tag, který přiloží ke čtečce, ozve se pípnutí (bzučák), což ho upozorní, že tag byl načten. Dioda upozorní zablikáním na příchod odpovědi ze serveru, která se zobrazí na displeji. Na displeji se na prvním řádku objeví druh režimu a na druhém informace o odpovědi serveru.

Došel jsem k závěru, že pro přidání a odebrání skladových položek bude využita jen jedna čtečka `RFID-RC522` s přepínáním mezi režimy. Jelikož nemám k dispozici displej ani bzučák, bude zobrazení režimu obstarávat jen `RGB LED dioda`. Čtečka načte `UID` tagu a odešle ho na server. Tam se zjistí typ tagu a podle toho se rozhodne, jaká bude akce. Podle typu akce se objeví barevná signalizace.
Akce podle typu tagu:

položka - Tag reprezentuje položku ve skladu. K vybrané položce se podle aktuálního režimu přičte nebo odečte kus. Při úspěšné akci se pošle příkaz k zablikání zelenou barvou.

režim - Tag změní režim z přidávat na odebírat a naopak. Reakce bude

podle změny: pro přidávání příkaz konstantní svícení zelenou a odebírání červenou.

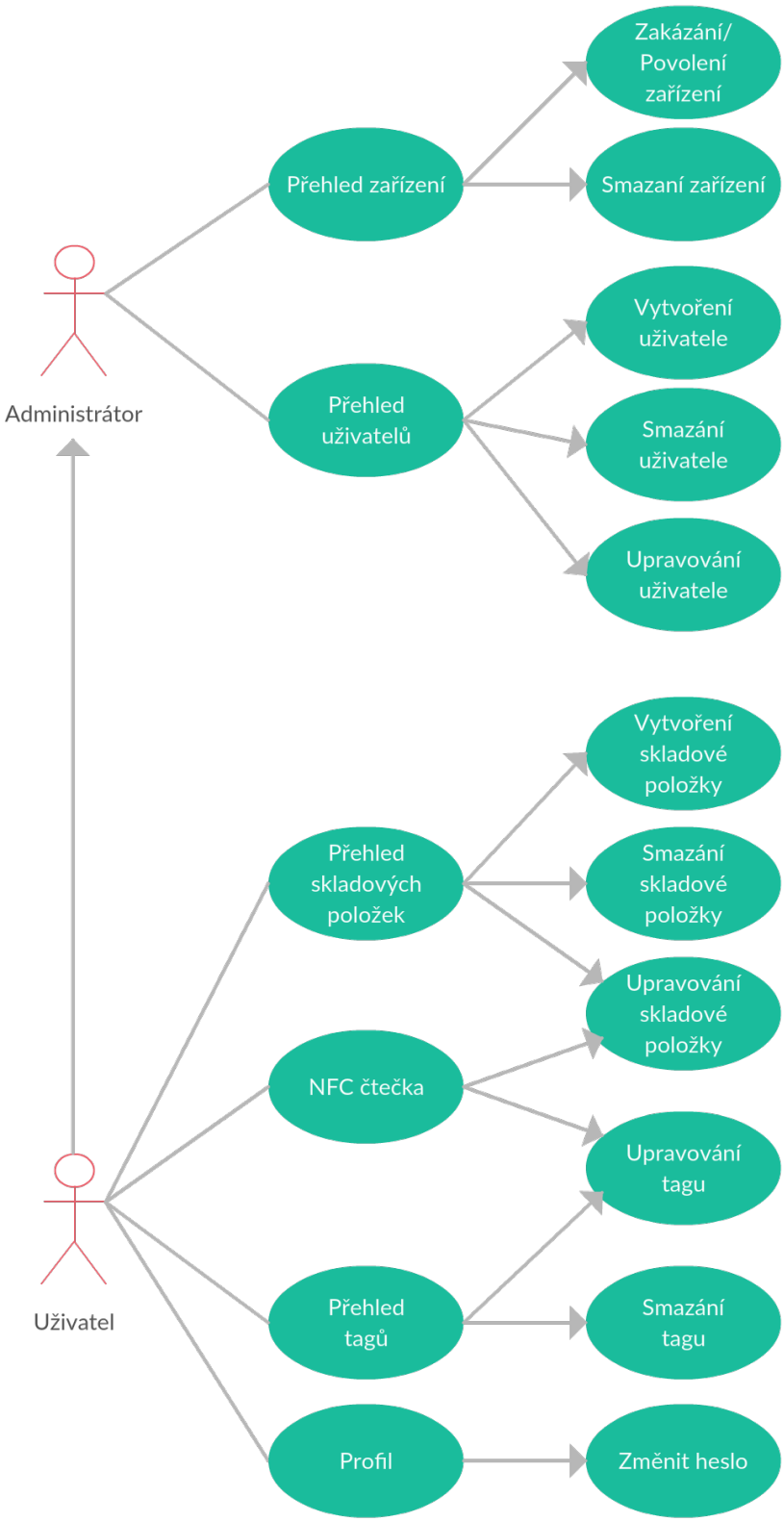
neznámý - Tento typ tagu nemá žádnou funkci. Odpověď serveru bude příkaz pro modré zablikání.

Při jakékoliv chybě se odešle příkaz pro zablikání červenou barvou.

5.3 Klient

Pro klienta správy skladu je důležité mít kompletní přehled. Týká se to hlavně obsluhy daného skladu, která zajišťuje příjem, výdej a přesun zboží. K usnadnění činnosti skladníka bude sloužit tato aplikace. V aplikaci samozřejmě nesmí chybět přidávání a upravování položek skladu. Mazaní položky bude možné jen v případě, pokud její počet kusů bude roven nule. Je to opatření proti schodku na skladě a i předcházení nepřesnostem při inventarizaci. Tag se do systému zaeviduje pouze načtením čtečky. Manuální zadání tagu nebude možné. Úprava tagu se bude týkat jen jeho typu. V případě typu položka je nutné přiřadit i konkrétní skladovou položku.

Administrátor bude mít navíc k dispozici správu uživatelů a zařízení. Nové uživatele bude moci přidat jen administrátor. Registrace uživatelů nebude možná, protože skladový systém je určený jen skladníkům. Podrobný přehled všech možných akcí pro konkrétní roli viz diagram 5.2.



Obrázek 5.2: Diagram užití mobilní aplikace

5.3.1 Platforma

Vybrat platformu je důležité pro uživatelskou přívětivost a použitelnost celého systému. K dispozici jsou tyto tři typy:

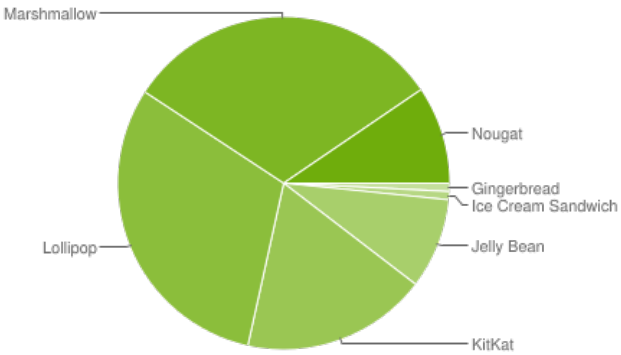
Mobilní aplikace - mobilní zařízení přináší možnost mít správu skladu neustále u sebe. Skoro každé zařízení dnes má `NFC`. Jelikož je `NFC` nástupcem `RFID`, existuje díky tomu zpětná kompatibilita, což poskytuje schopnost načíst `UID` tagu. Přiložením mobilu k tagu by skladník měl okamžitě veškeré informace o tagu případně i o položce skladu.

Desktopová aplikace - se hodí pro zobrazení velkého množství dat. Bohužel správa tagů by byla velmi obtížná a komplikovaná, protože vyhledání konkretního tagu lze jen podle `UID`.

Webová aplikace - obrovskou výhodou je možnost spustit aplikaci na jakémkoliv zařízení, které má webový prohlížeč. Nevýhody jsou stejné jako u desktopové aplikace.

Do prostředí skladu bude pro jeho správu ideální volba mobilní aplikace. `NFC` a přenositelnost jsou důležité aspekty. Jelikož ve skladu probíhá spousta změn, nebudou využity možnosti notifikací, aby skladník nebyl přehlcen informacemi. Z mobilních platforem jsem zvolil `Android`, protože umožňuje vývojářům plný přístup `NFC`. Zbývající platformy jako je `Windows Phone` nebo `iOS` nejsou vhodné. `Windows Phone` je oficiálně nepoužívaná platforma, jelikož její prodej se zastavil. `iOS` je velmi omezený systém, na kterém lze `NFC` využít jen k placení a párování s `Bluetooth` sluchátky. Pro skladový systém je nevhodný.

Vývoj bude cílen na `Androidu 6.0 Marshmallow`, protože je podle statistik nejrozšířenější verzí systému `Android`. Ke dni 5.6.2017 je na trhu zastoupen 31,2%. Aplikace bude nativní, jelikož bude určená jen pro `Android`, hybridní nemá význam. Nativní vývoj probíhá v jazyku `Java`.

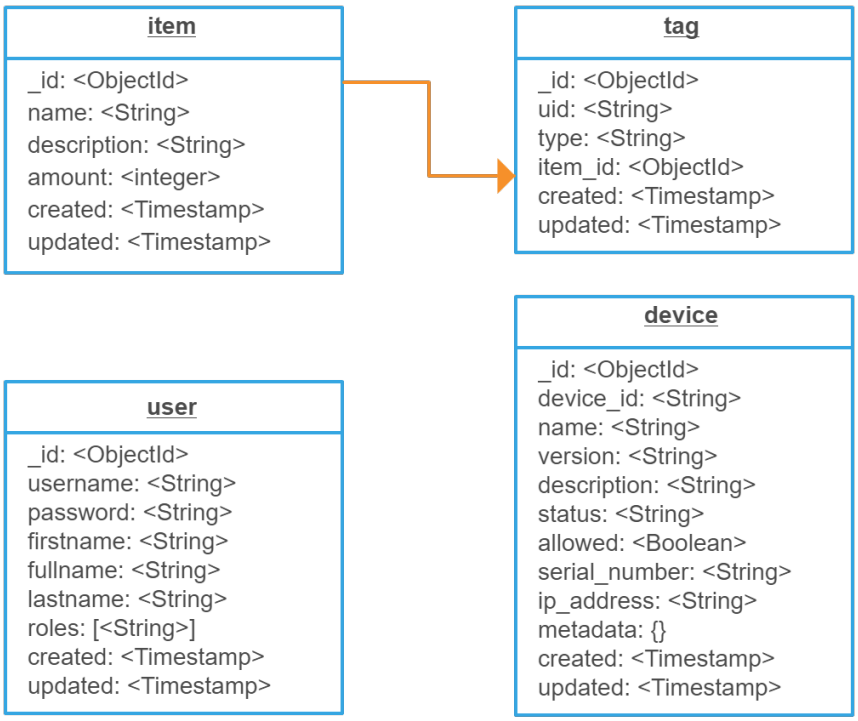


Obrázek 5.3: Zastoupení jednotlivých verzí systému Android ke dni 5.6.2017
(zdroj: `https://developer.android.com/about/dashboards/index.html`)

5.4 Databáze

Jako databázi jsem vybral `MongoDB`, protože díky vzdálenému přístupu umožňuje být umístěna jinde než server. Dalším přínosem je dokumentová orientace, což dovoluje vkládat data různého formátu. Tento typ databází má velmi rychlý zápis, který se využije u `MQTT` brokeru, jež do databáze ukládá příchozí zprávy. `JSON` format dat velmi urychlí vývoj `REST API`. Databáze `MongoDB` patří mezi nejpoužívanější databáze a její komunita bude mít řešení na jakýkoliv problém.

Návrh databáze bude obsahovat jen nejnutnější údaje, aby se dala prezentovat funkčnost skladového systému. Z návrhu lze vidět, že položka má jen název, popisek a množství. Ostatní údaje jsou aktuálně nedůležité. Tag obsahuje `UID` fyzického tagu, tak i `_id` generované databází, protože `UID` nemá standardizovaný formát. Návrh zařízení je velmi obecný, aby do budoucna vyhovoval každému zařízení. Důležitý atribut `metadata` může obsahovat jakýkoliv objekt, který dané zařízení bude potřebovat.



Obrázek 5.4: Návrh databáze

6 Server

Server je kombinací MQTT brokeru a HTTP serveru. Byl napsán v Node.js ve verzi 7.8. Pro správu balíčku se používá NPM. Jeho konfiguraci naleznete v souboru package.json, který se nachází v kořenovém adresáři tohoto projektu. Tento soubor obsahuje všechny potřebné moduly a informace o projektu.

6.1 Komunikace

6.1.1 MQTT

MQTT broker/server řeší modul Mosca. Je to jediný modul k dispozici pro Node.js. Modul Mosca potřebuje úložiště pro zprávy, podporovány jsou MongoDB, Redis, Mosquitto. Vybral jsem možnost MongoDB, protože je tato databáze již používaná k ukládání dat. Modul mi dává velké pole působnosti. Všechny části jako je autorizace, publikace a subscribe lze zcela přepsat, což u jiných brokerů nelze.

Standardně si klient generuje náhodné client_id a jelikož z uživatelského hlediska jsou takováto ID nepřívětivá, tak server umožní připojení jen klientům, kteří splňují tento formát. Každý klient, který se chce připojit k brokeru, musí jako client_id splňovat tento formát [app_name]/[device_id]. ID klienta musí být jedinečné, aby se předešlo konfliktu při připojování k MQTT brokeru. Název aplikace v client_id umožňuje, aby se mohly připojit různé aplikace z jednoho zařízení.

Zařízení se po připojení automaticky zaeviduje do databáze, přitom dojde k rozdělení client_id na název aplikace a ID zařízení, které se uloží spolu s IP adresou. Připojené zařízení informuje server o své verzi softwaru a sériovém čísle, tyto informace pomůžou správci v rozhodnutí, které zařízení aktualizovat a nalezení konkrétního zařízení.

Autorizace

Standardně se přístup k MQTT brokeru řeší přihlašujícími údaji. Buď se pro všechna zařízení určí společné přihlašovací údaje, které si každé zařízení bude pamatovat, nebo každému zařízení přidělí vlastní přihlašovací údaje, což zkomplikuje správu zařízení. V případě hacknutí se u první možnosti musí měnit hesla na všech zařízeních, u druhé jen na hacknutém zařízení.

Z tohoto důvodu jsem zvolil jiný způsob řešení. Zařízení se při připojení automaticky zaregistruje do databáze. Pokud již v databázi existuje, tak se zkontroluje povolení k přístupu, které přiděluje administrátor. V případě, že povolení nemá, je odpojen. Toto je elegantní řešení, které umožní administrátorům mít přehled o všech připojeních k `MQTT` serveru. Bezpečností riziko může vzniknout ukradením `client_id`. Tento problém lze řešit zakázáním přístupu v případě změny `IP` adresy.

6.1.2 REST API

HTTP server obstarává modul `Restify` framework, který slouží k tvorbě `REST API`. Díky přizpůsobení frameworku pro `API` neobsahuje šablonovací ani renderovací systém, což je znát na rychlosti zpracování požadavku. Také obsahuje užitečnou funkci `throttle`, při překročení 50 dotazů za sekundu dojde k zablokování zdroje žádostí.

`REST API` je umístěno na adrese `/api/v1/`, verzování standardně v `URL`. Data jsou k dispozici jen ve formátu `JSON`.

Popis všech dostupných `URL` adres v `API`:

Metoda	URL	Popis
GET	/api/v1/devices	Vrátí všechna zařízení
GET	/api/v1/devices/{id}	Vrací jedno zařízení
PUT	/api/v1/devices/{id}	Aktualizuje jedno zařízení
DELETE	/api/v1/devices/{id}	Odstraní jedno zařízení
GET	/api/v1/items	Vrátí všechny položky
POST	/api/v1/items	Vytvoří novou položku
GET	/api/v1/items/{id}	Vrací jedinou položku
PUT	/api/v1/items/{id}	Aktualizuje jednu položku
DELETE	/api/v1/items/{id}	Odstraní jednu položku
GET	/api/v1/tags	Vrátí všechny tagy
GET	/api/v1/tags/{id}	Vrátí jednu značku
PUT	/api/v1/tags/{id}	Aktualizuje jednu značku
DELETE	/api/v1/tags/{id}	Odstraní jednu značku
GET	/api/v1/tags/uid/{uid}	Vrátí jednu značku
GET	/api/v1/users	Vrátí všechny uživatele
POST	/api/v1/users	Vytvoří nového uživatele
GET	/api/v1/users/{id}	Vrátí jednoho uživatele
PUT	/api/v1/users/{id}	Aktualizuje jednoho uživatele
DELETE	/api/v1/users/{id}	Odstraní jednoho uživatele
GET	/api/v1/account/	Vrátí přihlášeného uživatele
PUT	/api/v1/account/password/	Změní heslo přihlášeného uživatele

Tabulka 6.1: REST API v1

Stavové kódy HTTP

`HTTP` umožňuje použít celou škálu kódů s jasnou sémantikou. V `REST API` využijeme jen několik z nich:

200 OK - Zpracování požadavku proběhlo bez problému.

201 Created - Zpracování požadavku proběhlo bez problému. Záznam byl vytvořen.

204 No Content - Zpracování požadavku proběhlo bez problému. Záznam byl smazán.

400 Bad Request - Požadavek nebyl zpracován. Důvodem může být serverem neznámý požadavek nebo špatná vstupní data.

401 Unauthorized - Požadavek nebyl zpracován. Žadatel nebyl autorizován.

403 Forbidden - Požadavek nebyl zpracován. Žadatel nemá potřebná práva.

404 Not Found - Požadovaný záznam nebyl nalezen.

500 Internal Server Error - Požadavek nebyl zpracován. Nastala nečekaná chyba.

Autentizace

Autentizace je proces ověření pravosti přihlašovacích údajů, tento proces řeší modul `Passport`, jehož jediným účelem je ověřování požadavků, a který to dělá přes rozšiřitelnou sadu zásuvných modulů, známých jako `strategie`. Existují `strategie` pro `autentizaci` pomocí `OAuth`, `Facebook`, `Twitter`, `OpenID` a spoustu dalších. Vybral jsem strategii `HTTP Basic` hlavně kvůli rychlé implementaci a jisté podpoře klienta. Použité `HTTP Basic auth` funguje velmi jednoduše: jméno a heslo je posláno jako textový řetězec, který se skládá ze jména, dvojtečka a heslo, pak se celé zakóduje `Base64`, poté vložíme do `HTTP` hlavičky. Příklad přidané autentizace do hlavičky `HTTP`:

```
Authorization:Basic a29obDpoZXNsbzM=
```

Aby byla zajištěna bezpečnost, počítá se s šifrovanou komunikací, jinak lze heslo lehce odposlechnout.

Autorizace

Autorizace je proces ověření potřebných oprávnění pro přístup uživatele k datům. Oprávnění je možné získat s přidělenou rolí.

Každý uživatel musí mít alespoň jednu roli. Možné role jsou `admin` a `user`. Uživatel s rolí `user` má přístup jen ke správě skladových položek a správě tagů. Zato administrátor má navíc k dispozici správu zařízení a správu uživatelů s rolí `user`.

Od spuštění serveru existuje účet s uživatelským jménem „admin“. Tento účet se vyváří automaticky a nelze smazat. Dalším specifikem účtu jsou rozšířená práva pro správu uživatele s rolí `admin`.

Dokumentace

Dokumentace je řešena frameworkem `Swagger` nejrozšířenějším nástrojem pro vývojáře `API`, který umožňuje vývoj napříč životním cyklem `API` od

návrhu či dokumentace až po testování či zavádění. Modul `swagger-jsdoc` umožňuje z komentářů vygenerovat `Swagger specifikaci` ve formátu `JSON`, která se předá `Swagger UI`. Zvolil jsem generování z komentářů, abych měl popis a implementaci na jednom místě a zabránilo se nekonzistenci. `Swagger UI` je webová stránka, která se automaticky generuje ze `Swagger specifikace`. Umožní komukoliv si prohlédnout a vyzkoušet `API` rozhraní, aniž by byla použita logika implementace.

Dokumentace je dostupná na adrese `/api/docs/`

6.2 Databáze

Pro jednodušší přístup k dokumentům v `MongoDB` databázi se použije modulu `Mongoose`. Nabízí podobnou funkcionalitu jako různé `ORM` frameworky. `Mongoose` nabízí nástroje pro `CRUD` operace (vytváření, čtení, editaci a mazání) nebo třeba umožňuje provádět validaci před vložením dokumentu do kolekce. V `Mongoose` všechno začíná u schématu. Každé schéma mapuje kolekci `MongoDB` a definuje strukturu dokumentů v rámci této kolekce. Každý klíč ve schématu definuje vlastnost v dokumentech, kterému se musí přiřadit odpovídající datový typ. Dále lze klíči nastavit max/min hodnota, max/min délka řetězce, `trim`, `index`, povinný údaj, výchozí hodnotu atd. Schéma se musí převést na model, s nímž pak lze pracovat.

Všem schématům jsem nastavil striktní dodržování struktury a přidal časová razítka vytvoření a úpravy dokumentu. Verzování dokumentů jsem vypnul, protože jsem ho nepotřeboval a naopak jsem povolil virtuální vlastnosti, které nejsou uloženy v databázi, ale dopočítávají se.

Položka

Položce jsem určil tyto vlastnosti: `name`, `description` a `amount`. Vlastnost `amount` má výchozí hodnotu 0, protože množství položky lze zadat jen při editaci položky. Schéma položky:

```
{
    name: { type: String, trim: true, minlength: 3,
            maxlength: 200, required: true },
    description: { type: String, trim: true, maxlength: 2000 },
    amount: { type: Number, default: 0, min: 0, required: true }
}
```

Tag

Tagu jsem přidělil tyto vlastnosti: `uid`, `type` a `item`. Vlastnost `uid` má omezení na 8 znaků, aby se nevkládalo `uid` z jiných typů tagů, které nejsou podporovány čtečkou `RFID`. Vlastnost `item` může obsahovat `_id` položky, `autopopulate` je proces automatického nahrazení specifikované cesty v dokumentu s dokumentem z jiné kolekce. Schéma tagu:

```
{
    uid: { type: String, lowercase: true, trim: true,
            minlength: 8, maxlength: 8, required: true,
            unique: true },
    type: { type: String, enum: ['unknown', 'mode', 'item'],
            default: 'unknown', required: true },
    item: { type: Schema.Types.ObjectId, ref: 'Item',
            index: true, autopopulate: true }
}
```

Uživatel

Uživateli jsem stanovil tyto vlastnosti: `username`, `password`, `firstname`, `lastname` a `roles`. Pro lepší práci s uživatelem jsem přidal virtuální vlastnost `fullname`, která se skládá z `firstname` a `lastname`.
Vlastnost `roles` musí obsahovat pole práv, může nabývat práv `admin` a `user`. Schéma uživatele:

```
{
    username: { type: String, lowercase: true, trim: true,
            minlength: 3, maxlength: 20, required: true,
            unique: true },
    password: { type: String, default: '', required: true,
            select: false },
    firstname: { type: String, default: '', trim: true,
            maxlength: 30 },
    lastname: { type: String, default: '', trim: true,
            maxlength: 30 },
    roles: { type: [{ type: String, enum: ['admin', 'user'] }],
            required: true }
}
```

Zařízení

Zařízení jsem určil tyto vlastnosti: `device_id`, `name`, `version`, `description`, `status`, `allowed`, `serial_number`, `ip_address` a `metadata`. Vlastnost metadata je typu `Mixed`, což znamená, že může obsahovat cokoliv. Důvodem jsou četná zařízení, která potřebují uložit různá data. Také obsahuje virtuální vlastnost `client_id`, která vzniká spojením `name`, lomítka a `device_id`. Vlastnost `status` může obsahovat jen hodnoty z `enum`, je to obdoba číselníků z relační databáze. Celé schéma zařízení:

```
{
    device_id: { type: String, trim: true, required: true },
    name: { type: String, trim: true, required: true },
    version: { type: String, trim: true},
    description: { type: String, trim: true },
    status: { type: String,
            enum: ['active', 'inactive', 'error'],
            default: 'inactive', required: true },
    allowed: { type: Boolean, default: false },
    serial_number: { type: String, trim: true },
    ip_address: { type: String, trim: true },
    metadata: { type: Schema.Types.Mixed }
}
```

6.3 Konfigurace

Konfiguraci obstarává modul `config`, který načte konfigurační soubory ze složky `./config` typicky umístěné v kořenovém adresáři aplikace. Modul podporuje velké množství konfiguračních souborů: `json`, `js`, `xml`, `toml`, `yaml` a další. Ze souboru `config/default.[format]` dědí všechny ostatní konfigurační soubory. Formát konfiguračního souboru byl vybrán `js`, pro možnost načíst data ze souboru `package.json`, který slouží jako konfigurační soubor pro správce modulů `NPM`.

V `Node.js` lze při spuštění nastavit parametr `NODE_ENV`, který určuje v jakém prostředí běží. Výchozí hodnotou je `development`. Některé moduly se přizpůsobují prostředí, např. podle `NODE_ENV` změní level logování. Modul `config` vybírá konfigurační soubor podle `NODE_ENV`, např. při `NODE_ENV=test` se vybere soubor `config/test.js`.

Ukázka konfiguračního souboru `config/default.js`:

```
{
    name: 'BPINI',
    version: package.version,
    host: '127.0.0.1',
    port: {
        http: 80,
        mqtt: 1883
    },
    logger: {
        level: 'info',
        path: './logs/bpini.log'
    },
    mongodb: {
        uri: 'mongodb://127.0.0.1:27017/warehouse',
        options: {}
    },
    api: true
}
```

V konfiguračním souboru lze nastavit `host` a `porty` pro `MQTT` a `HTTP`.

Další nastavení se týká umístění a level logování.

Pro připojení databáze musí do parametru `uri` zadat `Connection string` podle standardního formátu `MongoDB URI`. Dodatečná nastavení jako `ssl`, `connectTimeoutMS` atd. se nastavují přes parametr `options`.

Spolu se servem se spouští i `API`, automatické spuštění můžeme zakázat změněním parametrů `api` na false. `API` lze také spustit zcela samostatně.

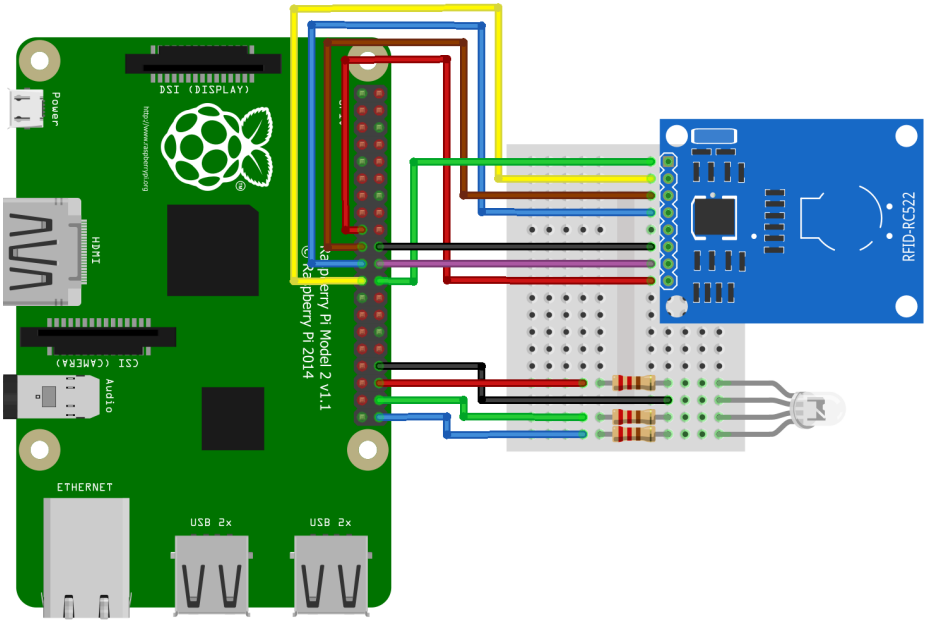
6.4 Logování

Logování zajišťuje modul `Bunyan`. Velmi často se používá a má podporu v ostatních modulech např. `Restify`. Typ logu je `rotating-file`, znamená to, že každý den se vytvoří nový soubor logu, který se uchová po dobu jednoho týdne. `Bunyan` je charakteristický logováním ve formátu `JSON`, díky tomu je lepší práce s logy např. vyhledávání. Propojení s `Restify` zajišťuje logování všech požadavků a odpovědí serveru.

7 Čtečka RFID

7.1 Sestrojení

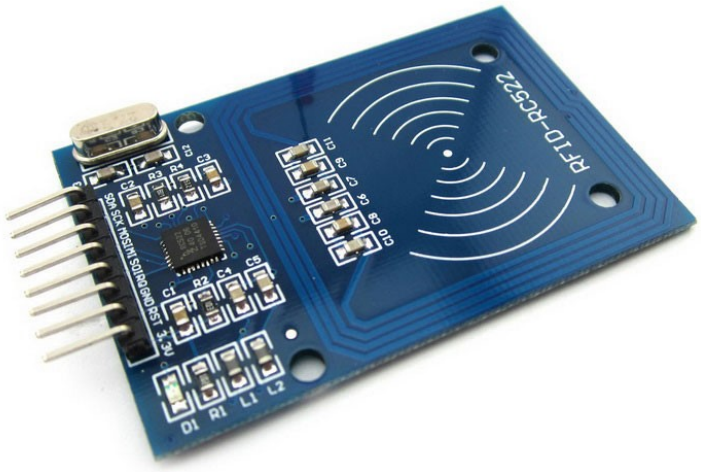
Pro sestrojení čtečky `RFID` jsem použil tyto hlavní komponenty: `Raspberry Pi 2`, `RFID-RC522` a `RGB LED diodu`. Kompletní model zapojení viz obrázek 7.1, schéma je k dispozici v manuálu.



Obrázek 7.1: Model zapojení `RFID-RC522` a `RGB LED` do `GPIO`

7.2 Zapojení RFID-RC522

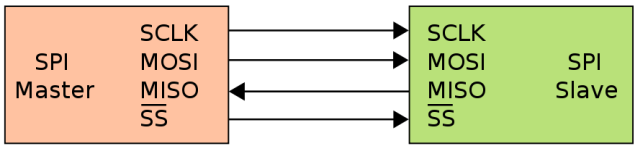
Čtečka `RFID-RC522` (viz obrázek 7.2) se k `Raspberry Pi` zapojí přes GPIO.



Obrázek 7.2: Čtečka `RFID-RC522`

Komunikace probíhá přes `SPI`, což je synchronní sériový datový protokol používaný pro komunikaci mezi mikroprocesory nebo s periferními zařízeními. Při komunikaci přes `SPI` je vždy jedno zařízení hlavní, které řídí periferní zařízení. Komunikaci `SPI` se vždy skládá z těchto pinů (viz obrázek 7.3):

- `MISO (Master Slave Out)` - `Slave` linka pro odesílání dat do `Master`
- `MOSI (Master Out Slave In)` - `Master` linka pro odesílání dat do periferií
- `SCK (Serial ClocK)` - hodinové pulsy, které synchronizují přenos dat



Obrázek 7.3: Zapojení `sběrnice SPI`: řídicí (`master`) a podřízené (`slave`) zařízení

`GPIO` má vyhrazené piny pro `SPI` (viz obrázek 3.2). `RFID-RC522` potřebuje navíc napájení `3V3`, `GND` a `pin 22` (podle BOARD) pro `RST` (Reset). Kompletní zapojení je popsáno na schématu (viz obrázek 7.1).

7.2.1 Zapojení RGB LED diody

Pro rozsvícení `RGB LED diody` potřebujeme 4 piny (viz obrázek 7.4). Nejdelší pin je `GND`, nalevo od `GND` je pin pro červenou, napravo pro zelenou a modrou barvu.

Obrázek 7.4: `RGB LED dioda`

Parametry diody pro provoz jsou: proud až 20 mA, napětí v propustném směru až 1,7 V. Před piny určující barvu je potřeba zapojit rezistory, protože `GPIO` dodává napětí 3,3 V. Rezistory se používají na ochranu diody před poškozením. Pro výpočet potřebného odporu rezistoru se používá Ohmův zákon. Udává vztah mezi napětím (U), proudem (I) a odporem (R) v obvodu. Matematicky formulovaný je takto:

$$U = R * I$$

Z toho vyplývá také vztah:

$$R = U/I$$

$$80 = (3.3 - 1.7)/0.02$$

Rezistor s odporem 80 Ohm je těžko dostupný, proto jsem použil nejčastěji používaný rezistor s 200 Ohm. Zapojení je vidět na schématu (viz obrázek 7.1).

7.3 Aplikace

Aplikace je napsaná v jazyce `Python` ve verzi 2.7, abych zajistil správnou funkčnost všech balíčků. Všechny potřebné balíčky pro spuštění aplikace jsou umístěny v souboru `requirements.txt`, který je umístěný v kořenovém adresáři.

7.3.1 Načtení UID tagu

Pro načtení UID je zapotřebí řídit GPIO na Raspberry Pi, to zajišťuje balíček RPi.GPIO. Vybrán byl tento balíček, protože je to oficiální knihovna od tvůrců Raspberry Pi a je součástí operačního systému. Balíček spidev umožňuje komunikovat se zařízením SPI s Raspberry Pi jako hlavním zařízením. Dalším balíčkem je rc522. Jeho hlavní účel je ovládat přes SPI součástku RFID-RC522. Tento balíček byl využit hlavně pro načtení UID, které jsem musel poupravit, jelikož docházelo k neustálému načítání tagu. Pro další načtení UID je nutné mít za sebou dva neúspěšné pokusy o načtení tagu, což znamená, že tag je odebrán z dosahu čtečky a nedochází tak k vícenásobnému načtení, než je potřeba.

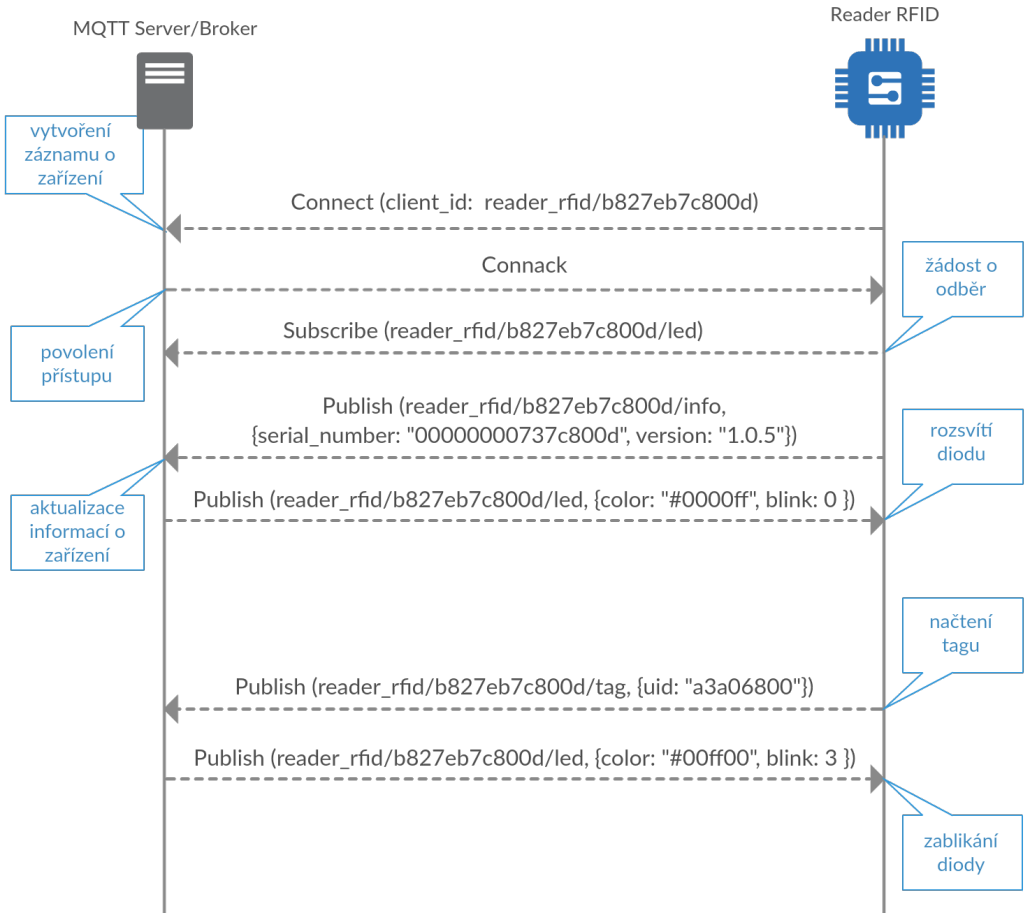
7.3.2 Komunikace

Pro komunikaci s MQTT brokerem jsem použil oficiální balíček od tvůrců MQTT. Balíček Paho vyžaduje před připojením určit klient ID. Klient ID je vygenerováno z názvu aplikace a ID zařízení. Jako ID se bere MAC adresa zařízení, tím se zajistí jedinečnost. MAC adresu má každé zařízení se síťovou kartou.

Ukázka klient ID pro čtečku RFID:

```
reader_rfid/b827eb7c800d
```

Klient ID se používá v tématu (topicku) zpráv, aby se identifikovalo od koho a komu je zpráva určená. Zde je ukázka komunikace pro připojení čtečky RFID k MQTT serveru/brokeru:



Obrázek 7.5: Komunikace mezi čtečkou RFID a MQTT server/broker

8 Mobilní aplikace

Vývoj mobilní aplikace probíhal nativně v Javě pro Android 6.0. Pro zrychlení vývoje jsem použil knihovnu Butter Knife. Její hlavní účel je eliminovat používání metody findViewById pomocí property injection @BindView. @BindView do jeho parametru stačí zadat ID požadované komponenty a knihovna Butter Knife automaticky inicializuje atribut. Tímto způsobem jsem dosáhl větší přehlednosti v kódu.

Aby aplikace mohla správně fungovat potřebuje tato práva:

- INTERNET - umožňuje aplikacím otevřít síťové sokety.
- ACCESS_NETWORK_STATE - umožňuje přístup k informacím o síti.
- NFC - umožňuje provádět I/O operace přes NFC.

8.1 Grafické rozhraní

Do grafického rozhraní jsem zavedl několik vylepšení. Na procházení velkého množství dat byla použita elegantní technika infinite scroll, bez nutnosti čekání se načte další stránka. Princip je jednoduchý, zatímco uživatel skroluje, další obsah je automaticky načítán.

Tlačítka jako Uložit, Přihlásit, atd. jsou umístěna v komponentě Toolbar, což je z důvodu dlouhých formulářů, aby nemusel uživatel skrolovat. Takto je tlačítko stále k dispozici.

Pro znázornění loadingu byly použity komponenty ProgressDialog a ProgressBar. ProgressBar se zobrazí při čekání na data od serveru. ProgressDialog při odesílání dat nejde zrušit kvůli tomu, aby se zabránilo přerušení nebo opětovné operaci. Dialog se zavře, pokud se operace dokončí nebo skončí chybou.

Pro zobrazení dat v přehledech byla použita komponenta CardView pro možnost zobrazit velké množství informací o jednom záznamu. Tímto způsobem jsem se vyhnul prokliku na detaily.

8.2 Komunikace

Komunikaci s REST API serveru mi zjednodušuje knihovna Retrofit. Stačí vytvořit Java rozhraní se všemi možnými dotazy na server a Retrofit si

je sám implementuje. Požadavky lze volat jako `Java metody` a přijatá data formátu `JSON` se konvertují na `JAVA objekty` pomocí knihovny `Gson`. Pro editaci `HTTP hlavičky` jsem použil knihovnu `OkHttp`. Tato knihovna mi také umožní komunikovat přes `HTTPS`.

8.3 Lokalizace

Mobilní aplikace je plně přeložená do dvou jazyků češtiny a angličtiny. Jazyk se vybírá automaticky podle nastavení v systému a pokud se nenajde požadovaný jazyk, tak se zvolí angličtina.

Na straně serveru se pracuje s časem bez časových zón. Aby čas nebyl irelevantní k uživateli, musí se podle časové zóny, kterou má systém `Android` nastavenou, přepočítat čas. Toto přináší výhodu, ať je uživatel odkudkoliv, vždy se mu zobrazí správný čas.

9 Testování

Pro testování mé aplikace pro skladové hospodářství jsem si vybral 6 lidí různého věku a odborného vzdělání. Účastníky testu jsem stručně seznámil s funkcemi mé aplikace a zadal jsem jim úkoly, které měli plnit na mobilním zařízení. Prvním úkolem bylo přihlásit se do systému aplikace. Administrátorem byl uživatelům dopředu vytvořen účet. Poté měli přidat novou položku skladu. Dalším krokem bylo zaevidovat libovolný tag a přidělit mu právě vytvořenou položku. Testující použili pro zaevidování tagu různé bezkontaktní karty. Posledním úkolem bylo přes čtečku `RFID` přidat nové položce tři kusy a jeden odebrat.

Testování probíhalo na těchto mobilních zařízeních:

- Samsung Galaxy A3 2016 (Android 6.0)
- Sony Xperia E5 (Android 6.0)
- Huawei P8 Lite (Android 6.0)

9.1 Výsledky testování

Účastníci testování, ač byli seznámeni s aplikací jen stručně, si s mobilní aplikací většinou poradili. Nejčastějším problémem se ukázalo, že chtěli přidělit tag k položce, ačkoli to má být opačně. Většina hledala tuto možnost v úpravách položky místo v úpravách tagu. Opačným způsobem to nelze, protože při přidělování tagu k položce bychom nevěděli, jaký tag přidělujeme, což by způsobilo nekonzistentnost dat. Po důkladném vysvětlení se zorientovali a už to pro ně nebyl žádný problém. Častý dotaz testujících byl, jak je možné vyhledat konkrétní položku podle názvu. Viz Možnosti rozšíření Po objasnění vše velmi dobře zvládli. Nakonec ohodnotili aplikaci jako uživatelsky přívětivou. Testující si vyzkoušeli různé bezkontaktní karty včetně Plzeňské karty. Systém nepřijal karty načtené pomocí mobilní aplikace, jejichž `UID` bylo delší než 8 znaků, protože to je jiný typ tagu a není podporován čtečkou `RFID`. Vypozoroval jsem, že testujícím chybí některé popisky, nebo že je některé matou. Tyto popisky jsem změnil a např. „Host“ byl přejmenován na „Server“.

Mobilní aplikace fungovala na všech zařízeních bez problémů.

9.2 Možnosti rozšíření

Systém skladového hospodářství potřebuje hodně vylepšení, nejvíce asi aktuální autentizace pro `API`. Autentizace `Basic Auth` není dostatečně bezpečná, nahradil bych jí bezpečnější strategií `OAuth 2.0`, která je ovšem složitější na implementaci.

Dále v tomto systému chybí údaje o tom, kdo vytvořil a upravil záznam v databázi. Tento údaj by pomohl dohledat skladníka, kdy a jak se zbožím manipuloval.

Také bych rozšířil údaje o skladové položce, jelikož nyní obsahuje jen základní údaje: jméno, popisek a množství. Přidal bych např. tyto údaje: cena, expirace a šarže. S jiným typem tagu by se otevřela možnost, uložit některé informace do paměti tagu. Tyto informace by mohly být k dispozici i off-line.

Mobilní aplikaci by prospělo, pokud by v aplikaci bylo vyhledávání v položkách. Z výsledku testování vyplývá, že tato funkce je pro uživatele důležitá a neměla být vynechána. U tagu je vyhledávání zbytečné, jelikož by šlo vyhledávat jen pomocí `UID`, což by bylo uživatelsky nepřívětivé. Aktuálně k vyhledání tagu slouží čtečka v mobilní aplikaci.

Ke zlepšení čtečky `RFID` by velmi pomohl `LCD` displej s rozlišením 16x2 znaky. `RGB LED dioda` pro signalizaci stavu čtečky není dostatečná. Je složité pochopit význam její signalizace. Má omezený počet stavů např. pro odpojení a vypnutí čtečky je stejný stav - zhasnutá `LED dioda`. `LCD` displej by mohl sloužit nejen k zobrazení konkrétního stavu čtečky, ale také k zobrazení informací o tagu a případně o změnách skladové položky. Takovéto řešení by bylo více uživatelsky přívětivé a cena zařízení by se zvedla jen o 150 korun.

Poslední rozšíření se věnuje posílání informací o zařízení serveru, bylo by vhodné pokud by se posílaly i chybové hlášky. Hlášky by sloužily správci, aby mohl konkrétní zařízení zkontrolovat a případně opravit.

10 Závěr

Výsledkem mé práce je skladový systém postavený na technologii RFID. Systém se skládá ze tří částí: serveru, čtečky RFID a mobilní aplikace. Server je postavený na Node.js. Komunikaci s dalšími částmi zajišťuje protokol MQTT a REST API. Informace o skladu jsou ukládány do NoSQL databáze MongoDB. Čtečka RFID slouží k přidávání a odebírání skladových položek. Mobilní aplikace pro Android slouží ke správě skladového systému, NFC se využívá k rychlému načtení informací o položce skladu.

Pro nasazení skladového systému k praktickému použití by bylo zapotřebí změnit RFID. Aktuálně se používá RFID s nízkou frekvencí. Tento typ RFID je určen např. pro autentizaci. Pro potřeby skladu je vhodné RFID s velmi vysokou frekvencí, tento typ umožňuje ukládat data na tag. UID tagu by mělo reprezentovat jeden fyzický kus ve skladě a informace o jakou položku se jedná uložit na tag. Další výhodou by byla možnost hromadného načítání.

V rámci této práce jsem vyzkoušel hodně rozmanitých činností, jako je pájení pinů, navrhování elektrických obvodů, práci s NoSQL databází a mobilní vývoj atd. Také jsem poznal druhou stranu mince nových technologií, jejichž vývoj postupuje velmi rychle. Během ročního vývoje skladového systému jsem musel kvůli změnám v knihovnách měnit již hotové části práce. Vždy to byl krok k lepšímu, ale zpomalovalo to celkový vývoj.

Přehled zkratek

API Application Programming Interface

BSON Binary JSON

CRUD Create, Read, Update, Delete

DOM Document Object Model

GND GrouND

GPIO General-Purpose Input/Output

GUI Graphical User Interface

HTTP Hypertext Transfer Protocol

I/O Input/Output

ID Identification

IoT Internet of Things

IP Internet Protocol

JSON JavaScript Object Notation

LAN Local Area Network

LCD Liquid-crystal display

LED Light-Emitting Diode

MAC Media Access Control

NFC Near Field Communication

NPM Node Package Manager

ORM Object-relational mapping

PyPI Python Package Index

QoS Quality of Service

REST Representational State Transfer

RFID Radio Frequency Identification

RGB Red, Green, Blue

RPi Raspberry Pi

SPI Serial Peripheral Interface

SSH Secure Shell

SSL Secure Sockets Layer

TCP Transmission Control Protocol

TLS Transport Layer Security

UID Unique Identification

UID Unique Identification

URL Uniform Resource Locator

UWB Ultra-Wideband

UWP Universal Windows Platform

XML eXtensible Markup Language

Literatura

[1] Chow, H. K. et al. Design of a RFID case-based resource management system for warehouse operations. *Expert systems with applications*. 2006, 30, 4, s. 561–576.

[2] Doleček, J. *Identifikace v informačních systémech (RFID)-radiofrekvenční identifikace*. PhD thesis, Bankovní institut vysoká škola, 2010.

[3] HOUŽVIČKA, T. *Aplikace NoSQL databází*. PhD thesis, Bankovní institut vysoká škola, 2012.

[4] Hron, M. Skladový systém pro obalovny živičných směsí. Master's thesis, Západočeská univerzita v Plzni, 2014.

[5] Karagiannis, V. et al. A survey on application layer protocols for the internet of things. *Transaction on IoT and Cloud Computing*. 2015, 3, 1, s. 11–17.

[6] Lacko, L. *Vývoj aplikací pro Android*. Computer Press, Albatros Media as, 2017.

[7] Malý, M. Protokol MQTT: komunikační standard pro IoT, 2016. Dostupné z: `https://www.root.cz/clanky/protokol-mqtt-komunikacni-standard-pro-iot/`.

[8] Nurseitov, N. et al. Comparison of JSON and XML Data Interchange Formats: A Case Study. *Caine*. 2009, 2009, s. 157–162.

[9] Sanner, M. F. – others. Python: a programming language for software integration and development. *J Mol Graph Model*. 1999, 17, 1, s. 57–61.

[10] Tilkov, S. – Vinoski, S. Node. js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*. 2010, 14, 6, s. 80–83.

[11] VÍTEK, M. *Skladové hospodářství konkrétního podniku*. PhD thesis, Masarykova univerzita, Ekonomicko-správní fakulta, 2007.

A Postup nasazení

Projekt naleznete na přiloženém DVD nebo je ke stažení na:
`https://github.com/kohlicekjan/BPINI`

A.1 Server

A.1.1 Instalace

Pro spuštění serveru je potřeba nainstalovat `Node.js`. Na oficiálních stránkách je k dispozici podrobný postup:
`https://nodejs.org/en/download/package-manager/`

Pro nainstalování potřebných modulů, spusťte ve složce `/src/Server/` tento příkaz:

```
npm install
```

Nainstalujte také databázi `MongoDB`, postup naleznete zde:
`https://docs.mongodb.com/manual/installation/`

A.1.2 Konfigurace

Ve složce projektu `/src/Server/config` jsou konfigurační soubory. V souboru `default.js` zadejte do `uri` adresu spuštěné databáze s názvem databáze, kterou chcete vytvořit. Dále také můžete nastavit adresu a porty serveru. Ukázka konfigurace:

```
host: '127.0.0.1',
port: {
    http: 80,
    mqtt: 1883
},
mongodb: {
    uri: 'mongodb://127.0.0.1:27017/warehouse',
    options: {}
}
```

A.1.3 Spuštění

Server spustíte tímto příkazem:

```
node server.js
```

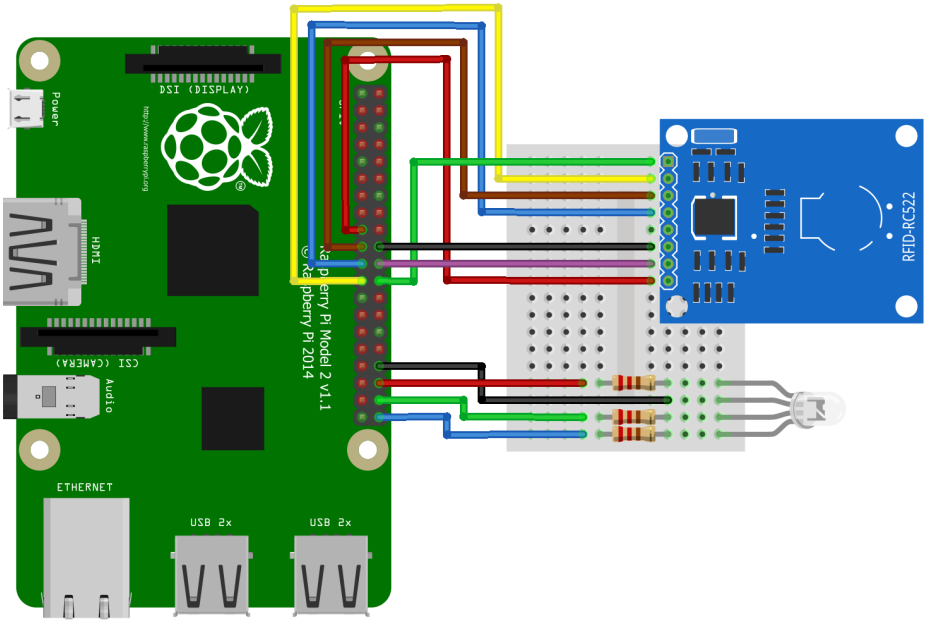
A.2 Čtečka RFID

A.2.1 Seznam součástek

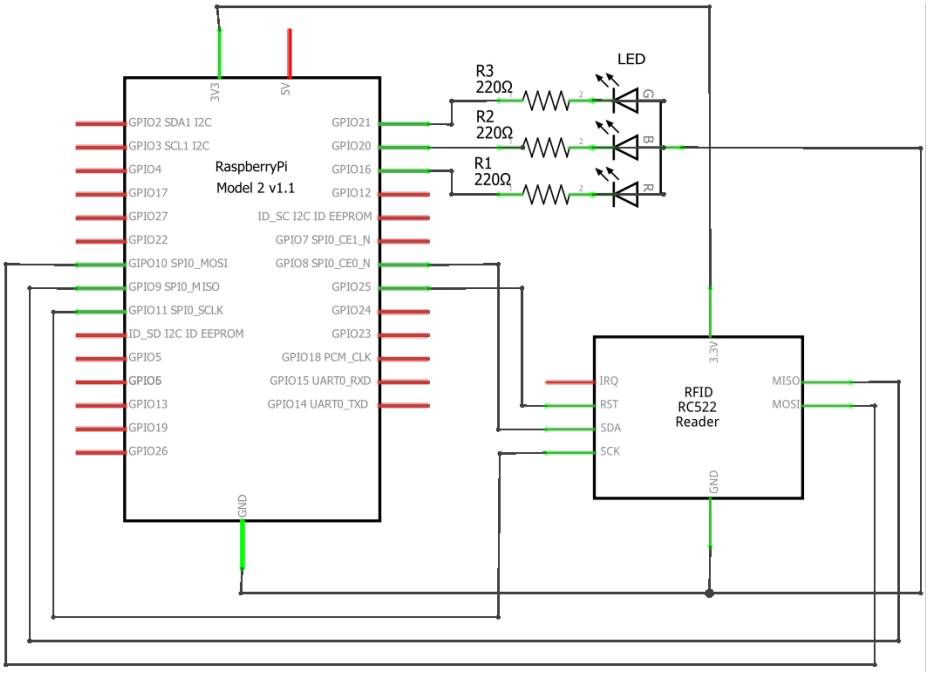
- 1x Raspberry Pi 2 Model B
- 1x micro SD karta
- 1x RFID-RC522
- 1x RGB LED
- 3x Resistor 220 Ohm
- 11x M-F Kabely samec samice
- 1x Nepájivé pole

A.2.2 Zapojení součástek

Součástky zapojte podle schématu (viz obrázky A.1 a A.2).



Obrázek A.1: Model čtečky RFID



Obrázek A.2: Schéma čtečky RFID

A.2.3 Instalace

Návod na instalaci sytému `Raspbian Jessie` naleznete na oficiálních stránkách:
`https://www.raspberrypi.org/documentation/installation/installing-images/README.md`

`SSH` protokol je z důvodu bezpečnosti ve výchozím stavu zakázán. Návod pro povolení naleznete na stránce:
`https://www.raspberrypi.org/documentation/remote-access/ssh/`

V konfiguraci `Raspberry Pi` povolte tyto položky:

- `Internationalisation Options` -> `SPI`
- `Advanced Options` -> `GPIO`

Příkaz pro otevření konfigurace:

```
sudo raspi-config
```

`Python` je součástí `Raspbianu`, potřebné balíčky se budou instalovat z `Python`

Package Index (PyPI). K tomu slouží nástroj pip. Tento nástroj je standardně nainstalován v Raspbian Jessie (ale ne Jessie Lite). Můžete jej nainstalovat pomocí příkazu:

```
sudo apt-get install python-pip
```

Následující příkaz nainstaluje všechny potřebné balíčky:

```
sudo pip install -r requirements.txt
```

A.2.4 Spuštění

Příklad spuštění s nastavením adresy serveru:

```
python ./reader_rfid/ -H 10.10.90.26
```

Volitelné parametry:

- -h ... vypíše nápovědu
- -v ... vypíše verzi
- -d ... zapne logování levelu debug
- -H ... adresa serveru
- -p ... port pro připojení k serveru

A.3 Mobilní aplikace

Aplikace je určena pro Android 6.0 a vyšší. Soubor BPINI-1.4.2.apk nahrajte do mobilního zařízení a spusťte instalaci. Při instalaci bude potřeba dočasně povolit instalaci z neznámých zdrojů. Po dokončení najdete aplikaci v menu mezi ostatními aplikacemi.

B Uživatelský manuál

B.1 Čtečka RFID

Čtečka svůj stav signalizuje pomocí `LED diody` její stavy (viz tabulka B.1).

LED dioda	**Popis**
vypnuta	není navázáno připojení
svítí zelená	režim přidávání položek
svítí červeně	režim odebírání položek
bliká zeleně	akce proběhla úspěšně
bliká červeně	nastala nečekaná chyba
bliká modře	tag nemá nastavenou žádnou funkci

Tabulka B.1: Stavy LED diody

Tag načtený čtečkou se automaticky zaeviduje. Funkčnost tagu potom můžeme nastavit v mobilní aplikaci. Je-li čtečka v režimu přidávání položek a načte-li tag reprezentující položku, přičte se k položce +1 množství. K přepínání režimů slouží speciální tag typu „režim“.

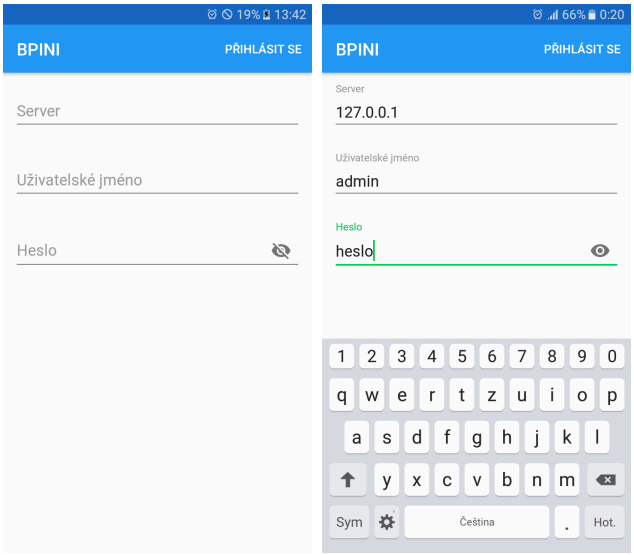
B.2 Mobilní aplikace

B.2.1 Přihlášení

Při prvním spuštění aplikace se zobrazí přihlašovací formulář (viz obrázek B.1a). Zadejte adresu serveru, pak následuje uživatelské jméno a heslo (viz obrázek B.1b).

Výchozí přihlašovací údaje administrátora systému jsou:

- uživatelské jméno: admin
- heslo: heslo

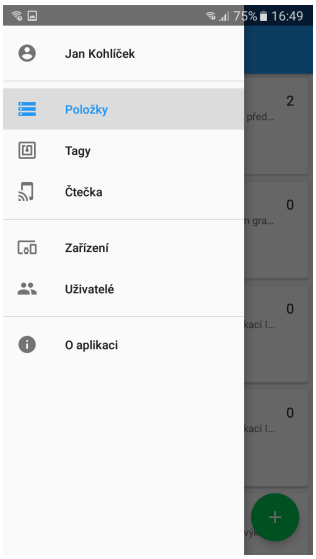


(a) Prázdný (b) Vyplněný

Obrázek B.1: Přihlašovací formulář

B.2.2 Menu

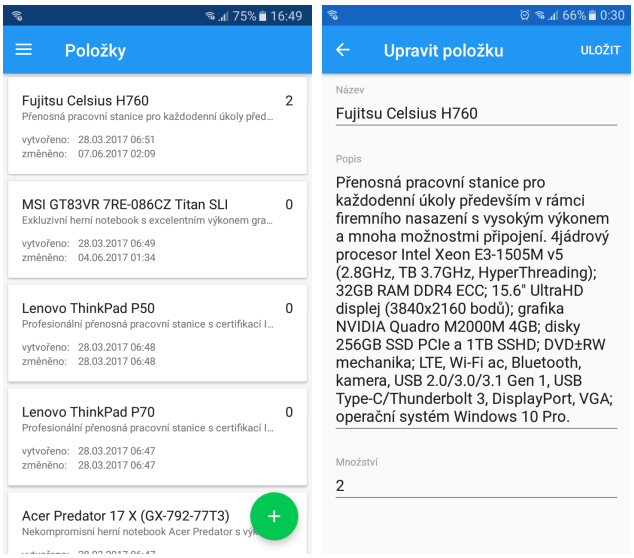
Menu se zobrazí stisknutím hamburger menu nebo vysunutím zpoza okraje (viz obrázek B.2). První položka menu otevírá detail přihlášeného uživatele. Na řádce jsou výše zmíněné „Položky“, pak následují „Tagy“. Další položkou v menu je „Čtečka“, ta je dostupná jen pro mobilní zařízení s `NFC`, ostatním se nezobrazí. Poté následují „Zařízení“ a „Uživatelé“, které jsou přístupné jen pro administrátora. Poslední je „O aplikaci“, zobrazí verzi, popis a autora aplikace.



Obrázek B.2: Menu aplikace

B.2.3 Položky

Po spuštění aplikace se zobrazí seznam položek skladu (viz obrázek B.3a). Položky můžete přidat pomocí zeleného plus v pravém dolním rohu a kliknutím na danou položku editovat. Položku lze smazat jen tehdy, když se její počet rovná nule.



(a) Seznam položek (b) Úprava položky

Obrázek B.3: Položky

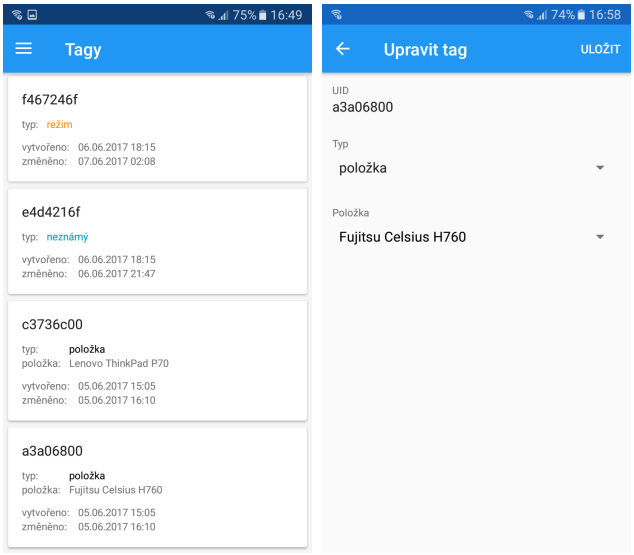
B.2.4 Tagy

Tagy nelze přidávat ručně, jen pomocí čtečky. Kliknutím na tag se zobrazí editace, která nabízí změnu typu. Tag může být tří typů:

neznámý - tag nemá nastavenou žádnou funkci

režim - tag umožňuje čtečce RFID přepínat režimy přidat/odebrat položku

položka - tag reprezentuje položku ve skladu

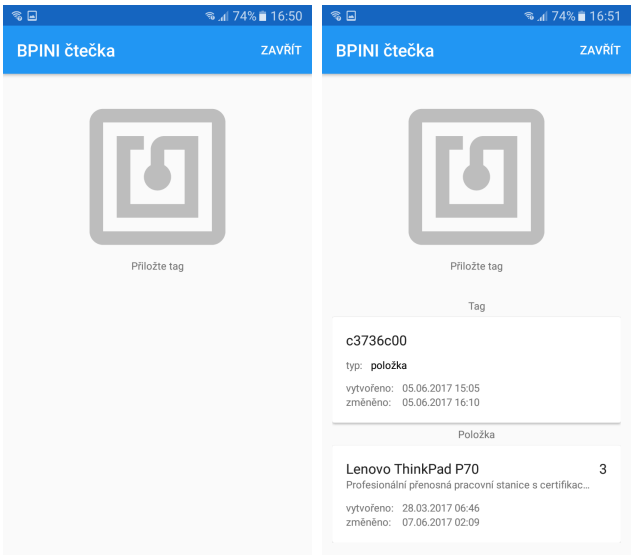


(a) Seznam tagů (b) Úprava tagu

Obrázek B.4: Tagy

B.2.5 Čtečka

Čtečka čekající na přiložení tagu (viz obrázek B.5a). Po přiložení tagu se načtou detailní informace (viz obrázek B.5b), ale pokud není v systému zaevidován, pak je vytvořen tag typu „neznámý“.

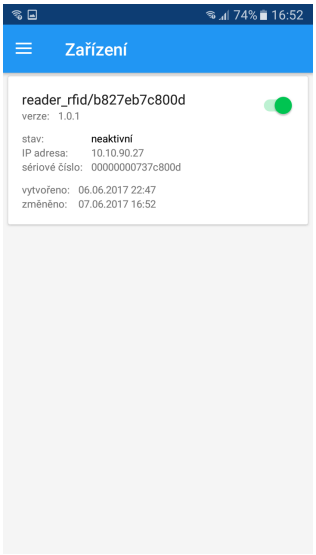


(a) Připravená čtečka (b) Načtený tag

Obrázek B.5: Čtečka

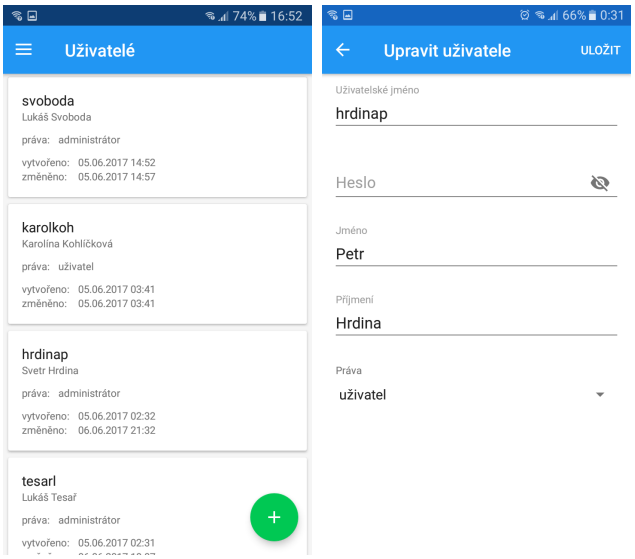
B.2.6 Administrace

Seznam zařízení se ukazuje jen administrátorům (viz obrázek B.6). Zařízení se evidují automaticky při jakémkoli pokusu o připojení k serveru. Aby se zařízení připojilo, je nutné, aby konkrétnímu zařízení byl povolen přístup. Ten se mění pomocí přepínače.



Obrázek B.6: Zařízení

Jen administrátor má přístup ke správě uživatelů (viz obrázek B.7a). Dostat se na vytvoření nového uživatele je možné přes zelené plus v pravém dolním rohu a kliknutím na uživatele editovat. Uživatele s rolí „administrátor“ může vytvářet a editovat jen výchozí administrátor.



(a) Seznam uživatelů (b) Úprava uživatele

Obrázek B.7: Uživatelé

C Obsah přiloženého CD