VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

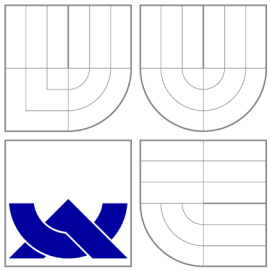
APLIKACE PRO ŘÍZENÍ A VZDÁLENOU SPRÁVU PLATEBNÍCH TERMINÁLŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

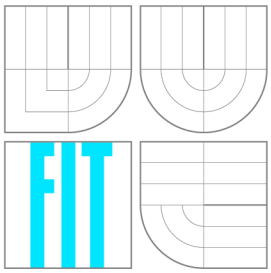
AUTOR PRÁCE
AUTHOR

Bc. FRANTIŠEK GAJDŮŠEK

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

APLIKACE PRO ŘÍZENÍ A VZDÁLENOU SPRÁVU PLATEBNÍCH TERMINÁLŮ

APPLICATION FOR PAYMENT TERMINALS CONTROL AND REMOTE MANAGEMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. FRANTIŠEK GAJDŮŠEK

VEDOUCÍ PRÁCE
SUPERVISOR

doc. RNDr. JITKA KRESLÍKOVÁ, CSc.

BRNO 2014

Abstrakt

Cílem práce je vytvořit ovládací software pro autonomní platební zařízení. Účelem tohoto softwaru je usnadnit práci s obsluhou těchto zařízení, a minimalizovat nutné servisní výjezdy k zařízení. Popsány jsou jednotlivé technologie použité při implementaci i protokoly použité pro komunikaci. Dále text práce popisuje postup práce od analýzy, přes návrh, implementaci a testování až po postup jak software zprovoznit. V zhodnocení projektu je popsán stav do jakého se software dostal a jak byl v praxi využit.

Abstract

The goal of this thesis is to create control software for stand-alone payment machines. The purpose of this software is minimize maintenance cost and usability as much as possible. Description of technologies and protocols used in implementation are in the text. Next, there is process of work beginning with analysis, continues with design, construction, testing and installation manual. Projekt status and product utilization at the end of project is in chapter about evaluation.

Klíčová slova

automatizace, sítě, Python, Django, uživatelské rozhraní, Bootstrap, web, Ajax, databáze, MariaDB, internet věcí

Keywords

automatization, networking, Python, Django, user interface, Bootstrap, web, Ajax, database, MariaDB, internet of things

Citace

František Gajdůšek: Aplikace pro řízení a vzdálenou správu platebních terminálů, diplomová práce, Brno, FIT VUT v Brně, 2014

Aplikace pro řízení a vzdálenou správu platebních terminálů

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením paní doc. RNDr. Jitky Kreslíkové, CSc.

.....
František Gajdůšek
27. května 2014

Poděkování

Tímto způsobem bych rád poděkoval všem lidem z firmy Siemens, kteří mě v průběhu práce poskytovali odbornou pomoc.

Obsah

Kapitola 1

Úvod

V dnešní době existuje stále více a více autonomně fungujících zařízení. Tato zařízení jsou buďto zcela automaticky fungující, nebo se jedná o samoobslužná zařízení. Oboje však mají jednu věc společnou. V zájmu provozovatele je jednak zajištění co nejsnazšího, tj. nejlevnějšího, servisu, a taktéž možnost okamžitého přehledu o využití daného zařízení.

Cílem práce je vytvořit software, který bude komunikovat s jednotlivými zařízeními a na jednom místě bude provozovateli těchto zařízení zobrazovat přehled o jednotlivých zařízeních. Díky tomuto software nebude nutné preventivně kontrolovat funkčnost zařízení technikem přímo na místě, zařízení budou ihned hlásit případné problémy na centrální místo. Okamžitým hlášením problémů se navíc výrazně zkrátí doba od vzniku poruchy po její odstranění, oproti pouze periodické kontrole. Do určité míry pak půjde zařízení vzdáleně ovládat, aby servisní technik vyjížděl pouze v případech kdy je fyzický zásah nezbytný, například v situaci mechanické poruchy nebo doplnění potřebných náplní.

Práce vychází ze semestrálního projektu. Semestrální projekt spočíval v návrhu tohoto softwaru, jeho součástí byly analýza a návrh, a to včetně výběru technologií a analýzy rizik. Součástí pouze diplomové práce je implementace a testování, na které navazují kapitoly s návodem na zprovoznění softwaru a kapitoly rozebírající zhodnocení a možnosti pokračování projektu.

Kapitola 2

Význam a využití projektu

Ještě před návrhem aplikace je potřeba objasnit důvody, proč tento software vytvářet a k čemu bude sloužit. Aby vůbec mohl být software kladně přijat uživateli, musí mít pro uživatele nějaký význam, musí uživateli v něčem pomoct, nějakým způsobem mu usnadnit práci. Nebo se může jednat o software určený pro zábavu, ale to není náš případ. Proto se dále zaměříme právě na přínosy uživateli k usnadnění práce a zlepšení informovanosti o využití zařízení pro usnadnění budoucího rozhodování.

2.1 Stovky až tisíce prodejních zařízení

Všechna tato zařízení jsou rozmístěna po velké ploše. Nemělo by význam mít více zařízení blízko sebe, naopak, rozmístěny jsou s rozestupy všude tam, kde je zákazníci očekávají. Proto není vůbec jednoduché všechna tato zařízení obsluhovat fyzicky přímo na místě. Technik přijede k jednomu zařízení, pak se musí dopravit k dalšímu, pravděpodobně automobilem, protože vzdálenost automatů není úplně zanedbatelná a pro případnou údržbu si musí převážet různé nástroje. Tato údržba by byla velmi drahá a nepraktická. Proto je nezbytné problém nějakým způsobem řešit.

2.2 Internet věcí

Moderní pojem, který však přesně vyjadřuje řešení nastíněného problému. Jedná se o pojem, který řeší samostatnou komunikaci strojů přes internet. Stroje komunikují za účelem plnění nějakého úkolu. Tento úkol je sice vyžadován lidmi, ale lidem jde pouze o výsledek činnosti. Nijak neřeší jednotlivé části komunikace ani se přímo nezajímají o jednotlivé posílané zprávy. Stroje komunikují samostatně a lidem poskytují pouze důležité informace, které je zajímají.

Důvod řešení samostatné komunikace je usnadnění práce při obsluze zařízení. Výjezdy technika k jednotlivým zařízením sice zcela neodstraní, ale výrazně zminimalizuje. Zařízení budou samostatně komunikovat prostřednictvím internetu, poskytovat o sobě různé informace. Díky tomu technik nemusí vyjíždět na pravidelné kontroly. Pokud zařízení pravidelně hlásí funkčnost, například informacemi o úspěšně prováděných transakcích, není nutný jakýkoliv servisní zásah. Jakmile v zařízení začne docházet nějaká náplň, zařízení pošle tuto informaci a technik by si měl v nejbližší době naplánovat servisní výjezd k tomuto automatu. Zařízení si navíc zvládne diagnostikovat některé poruchy, a v případě skutečného zjištění takové poruchy pak okamžitě odešle informaci tak, aby se co nejdříve dostala k technikovi.

Opomnět nelze ani mnohem kratší dobu nefunkčnosti zařízení. Tím, že zařízení okamžitě poskytne informaci o nefunkčnosti, se výrazně zkrátí doba mezi vznikem poruchy a opravou. Pokud by poruchy zjišťoval technik na místě, mohlo by trvat mnoho dní než by došlo k objevení vzniklé poruchy.

Mimo ušetřené náklady za výjezdy technika kvůli pravidelné údržbě je zde ještě jedna podstatná výhoda. Okamžitý přehled o využití automatu. Technik by tato data sice mohl stahovat ze zařízení během servisní návštěvy, ale díky online propojení jsou tato data dostupná ihned prakticky bez zpoždění. A na základě těchto dat lze na jednotlivých zařízeních, například na základě poptávky, měnit ceny. A to okamžitě a ne až s mnohedenním zpožděním.

2.2.1 Způsob propojení

Všechna prodejní zařízení jsou si rovnocená, co se týče funkčnosti pak úplně stejná. Proto by bylo velmi obtížné a nepraktické pouze ze zařízení vytvářet síť, která bude někde zajišťovat informování uživatelů. Pro zajištění komunikace jednak s ostatními zařízeními a jednak s uživatelem je potřeba přidat centrální bod – ovládací software. S tímto softwarem budou zařízení komunikovat, budou mu hlásit všechny poruchy a vzniklé události. A současně tento centrální bod bude moct tato zařízení řídit. A pro uživatele tento centrální bod zajistí možnost zjištění předávaných informací i možnost provádět změny nastavení jednotlivých zařízení.

2.3 Praktické uplatnění

Má se jednat o ovládací software pro velké množství zařízení, která mají stejnou funkcionalitu. Jeho účelem je jednak zajistit uživateli na jednom místě přehledně informace o těchto zařízeních, a pak také umožnit do určité míry tato zařízení ovládat, měnit nastavení, aktualizovat a podobně. O zařízeních lze zjišťovat poměrně velké množství údajů, jedná se především o stav zařízení – jestli všechno funguje tak jak má, nebo je s nějakým, a to konkrétně jakým, zařízením problém. Dále pak zařízení poskytují informace o platbách, z těchto informací pak lze vytvářet různé statistické přehledy.

Kapitola 3

Metody komunikace se zařízeními

Aby bylo možné se zařízeními komunikovat, je potřebné zvolit nějakou nebo nějaké metody komunikace, které budou umět jednak zařízení, a jednak ovládací software. Zařízení komunikují s ovládacím softwarem prostřednictvím internetu. Ke komunikaci je potřeba podpora základních internetových komunikačních protokolů. Dále pak speciálního firemního protokolu, který určuje strukturu a vlastnosti posílaných informací.

3.1 UDP (User Datagram Protocol)

Jeden ze základních internetových protokolů [3] třetí vrstvy TCP/IP (Transmission Control Protocol and the Internet Protocol), který může být použit pro komunikaci se zařízeními. Jedná se o nespolehlivý protokol. Přenáší jednotlivé zprávy – bloky dat, tzv. datagramy. Nezaručuje zachování pořadí doručených zpráv, ani doručení všech zpráv. Protokol je velmi jednoduchý, je bezestavový, to znamená že jednotlivé zprávy se zpracovávají nezávisle a ihned jak je to možné, nedochází k blokování při čekání na další blok dat. Implementace je velmi jednoduchá, není potřeba řešit použití více vláken ani jiné metody zpracování, a to právě díky bezestavovému chování.

Zmíněné výhody protokolu jsou současně i nevýhodami. Protože nelze zaručit doručení všech zpráv, musí si doručení zajistit vyšší čtvrtá aplikační vrstva, a to firemní komunikační protokol. Ten zajišťuje, že pokud zařízení nedostane potvrzení o doručení zprávy, danou zprávu odešle znovu. V případě že by byla původní zpráva už doručena, a došlo pouze ke ztrátě potvrzující zprávy, aplikační protokol to dokáže rozpoznat [2].

3.2 TCP (Transmission Control Protocol)

Další ze základních internetových protokolů [3] třetí vrstvy TCP/IP. Může být využíván pro komunikaci se zařízeními. Protokol je spolehlivý, stavový, přenáší proud dat v zaručeném pořadí a se zaručeným doručením zpráv. O nemožnosti zprávu doručit, například z důvodu výpadku v síti, je odesílatel informován.

Fakt, že je protokol stavový, však přináší jednu zásadní komplikaci. Komunikační program musí počítat se spojeními, a nějak je zpracovávat. Jedna z možností je pro každé spojení vytvořit speciální vlákno. Nebo je možné používat asynchronní zpracování, kdy jedno vlákno zpracovává více spojení. Vlákno pracuje v jeden okamžik pro jedno spojení, a to pro takové kde dojde k nějaké události, například jsou k dispozici další data ze sítě. Při asynchronním zpracování dochází jakoby k vícevláknovému zpracování, ale, a to i na

víceprocesorových počítačích, je vždycky zaručeno že pracuje současně pouze jedno vlákno. Pokud dorazí více událostí, jsou vždy zpracovávány postupně. Když dorazí další událost dříve než je předchozí zpracována, čeká.

3.3 Firemní komunikační protokol

Firemní komunikační protokol pracuje na čtvrté vrstvě modelu TCP/IP, nad protokoly UDP nebo TCP (podle nastavení zařízení). Funguje na principu posílání jednotlivých, speciálně naformátovaných zpráv [2]. Při použití protokolu UDP pošle jednu zprávu v jedné zprávě protokolu UDP, a jako odpověď očekává jednu potvrzující zprávu v jedné UDP zprávě. Ve chvíli, kdy protokol běží nad TCP, funguje podobně. Zařízení naváže TCP spojení s ovládacím serverem, odešle naformátovanou zprávu, vyčká na odpověď od serveru a ukončí spojení. Pro odeslání další zprávy znovu naváže spojení.

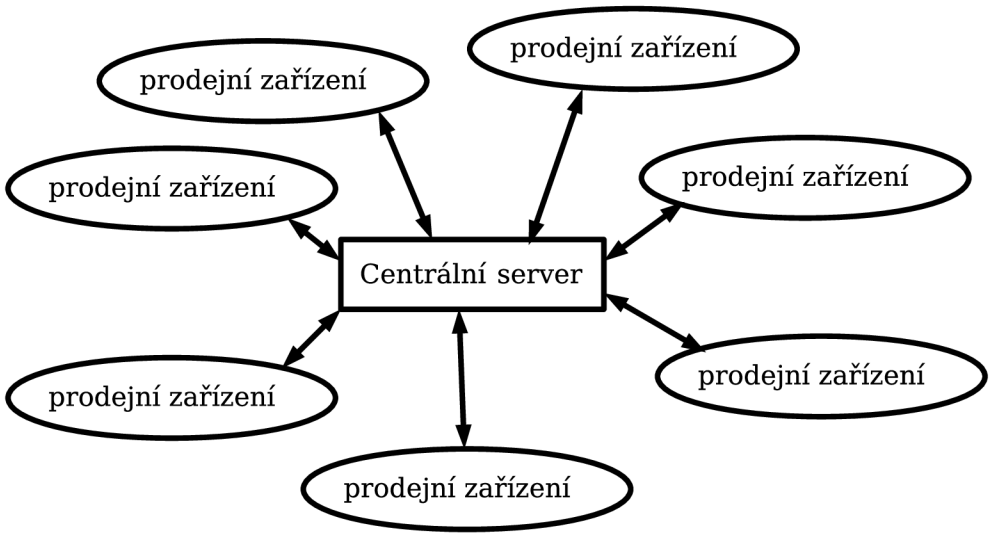
Všechny zprávy jsou číslované. V případě že zařízení nedostane na odeslanou zprávu odpověď, zprávu odesílá znovu. Server podle čísla pozná, jestli už zprávu zpracovával a ztratila se odpověď, nebo jestli danou zprávu ještě nezpracovával.

Kapitola 4

Základní architektura

Nejprve je nutné zjistit, co přesně je potřeba vytvořit. Následně pak rozhodnout jak to bude technicky fungovat.

Všechny prodejní terminály konkrétního provozovatele budou komunikovat s jedním centrálním ovládacím místem. Fyzicky budou komunikovat prostřednictvím sítě internet, logicky se pak jedná o topologii typu hvězda.



Obrázek 4.1: Základní architektura – logická hvězda [vlastní]

4.1 Současný stav

Úkolem této práce není vytvářet kompletní systém včetně zařízení, ale pouze centrální ovládací software. Koncová prodejní zařízení jsou daná, stejně tak je daný i protokol, kterým jsou tato zařízení schopna komunikovat s ovládacím softwarem. Z tohoto důvodu už se ve všech následujících kapitolách budeme zabývat pouze centrálním ovládacím software. Tento centrální software funguje v režimu server, kdy neustále naslouchá na požadavky klientů – autonomních prodejních zařízení.

4.2 Základní součásti

Vytvořit jeden program, který bude zajišťovat všechny schopnosti by bylo poměrně nepraktické. Lepší je program vhodně rozdělit podle úkolů, které má vykonávat. Proto si ihned definujeme rozdělení na součásti.

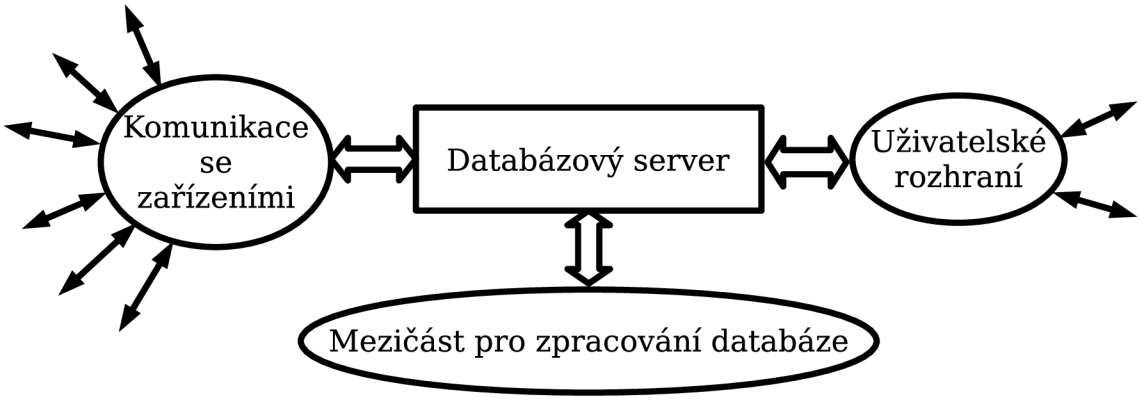
Centrální ovládací software zajišťuje dvě základní činnosti. Jednak musí neustále být na příjmu pro požadavky prodejních zařízení a být schopen s nimi komunikovat prostřednictvím daného protokolu. Jako druhá, neméně důležitá, činnost je zajištění uživatelského rozhraní aplikace. Oboje části navíc musí fungovat na principu klient-server, v obou případech v režimu server [3]. Část pro komunikaci s prodejními zařízeními musí být server z důvodů daných protokolem jakým zařízení fungují. Uživatelská část musí být taktéž server – bude přijímat požadavky od uživatele a odpovídat na ně. Tyto dvě části navíc spolu musí umět efektivně komunikovat.

4.3 Použití databáze

Protože už víme že software bude rozdělen na části, musíme definovat způsob jakým tyto části budou mezi sebou komunikovat. Hlavní činností aplikace je na straně komunikace s prodejními zařízeními sbírat a ukládat data, a na straně uživatelského rozhraní tato data smysluplně zobrazovat. Z tohoto důvodu jsem se rozhodl pro komunikaci mezi těmito částmi použít databázi. Část komunikující s prodejními zařízeními bude převážně ukládat přijatá data do databáze. Část zajišťující uživatelské rozhraní bude tato data převážně číst a zobrazovat uživateli. Pro zajištění dostatečné rychlosti uživatelského rozhraní je vhodné přidat ještě jednu část, která bude předzpracovávat data v databázi, například počítat denní statistiky a podobně.

Databáze navíc neslouží pouze jako komunikační prostředek, ale současně v ní zůstávají data uložena dlouhodobě. Tato data pak lze bezproblémově, za použití existujících nástrojů, zálohovat, a případně i přenášet do jiné instalace ovládacího softwaru.

Pro zefektivnění spolupráce částí bude potřeba data zpracovat a přepočítat ze způsobu jakým je ukládá komunikační část, do struktur vhodných pro uživatelské rozhraní.



Obrázek 4.2: Architektura ovládacího software [vlastní]

Kapitola 5

Použité technologie

Jako programovací jazyk jsem se rozhodl použít Python, a to pro všechny části. V tomto jazyce lze bez problémů vytvářet síťové aplikace [1]. Jako uživatelské rozhraní bude použita webová aplikace. Serverová část webové aplikace bude napsána v Pythonu, a to za použití webového rámce Django [6]. Ke zjednodušení tvorby klientské části webové aplikace bude použit klientský webový rámec Bootstrap [5]. Jako databázový server bude použit MariaDB [8], což je databázový server kompatibilní s MySQL, ale stále vyvíjený i ve volně šiřitelné variantě.

5.1 Python

Obrázek 5.1: Logo jazyka Python [https://www.python.org/]

Python [12] je skriptovací, objektově orientovaný, programovací jazyk s automatickým překladem do mezikódu. V roce 1991 jej navrhl Guido van Rossum, který je stále jedním z hlavních vývojářů. Je vyvíjen jako Open Source [11] projekt, který je zdarma dostupný pro většinu běžných platforem (Unix, Windows, Mac OS). Ve většině distribucí Linuxu je součástí základní instalace.

Python je dynamický interpretovaný jazyk. Byl navržen tak, aby umožňoval tvorbu rozsáhlých, plnohodnotných aplikací, včetně grafického uživatelského rozhraní. V současné době se používá i pro tvorbu webových aplikací. Kód programu je krátký a může být velmi dobře čitelný.

Velkou výhodou jazyka Python je snadnost učení. Je vhodný jak pro úplné programátorské začátečníky, tak pro programátory v jiných jazycích. Pro programátory v jiných jazycích zavádí pouze jednu značně neobvyklou vlastnost, a to používání odsazení místo blokových závorek. Ve většině jazyků je odsazování sice dobrou programátorskou zvyklostí, ale jazyk jej nevyžaduje. V Pythonu je odsazování součástí syntaxe jazyka. Jednoduchost jazyka ani rychlost učení však nejsou na překážku v používání, a to ani u rozsáhlejších projektů.

Přestože je Python skriptovací jazyk, velmi dbá na rychlost programů. Jazyk není čistě interpretovaný, ale interpret všechny programy automaticky překládá do mezikódu, čímž významně zrychlí běh programu. Zároveň však nepřidá negativní vlastnosti překládaných

jazyků. Tento koncept zajišťuje zachování všech výhod skriptovacích jazyků a přidává zrychlení oproti čistě interpretovaným jazykům. Určité zpomalení oproti překládaným jazykům patrné je, proto je velmi vhodný pro úkoly které při vykonávání nevyžadují provádění velkého množství výpočtů. Pro tvorbu různých aplikací, které komunikují přes síť, se používá velmi často. Důvodem je snadnost použití a díky kvalitním knihovnám pouze minimální snížení rychlosti oproti nízkoúrovňovým jazykům, jako je například C. Toto snížení rychlosti navíc nezpůsobuje téměř žádné zpomalení aplikace která převážně čeká na síťové operace, ale nanejvýš drobné zvýšení zátěže procesoru.

V současné době se používají dvě hlavní verze jazyka, a to 2.0 a 3.0, které nejsou zcela kompatibilní. Postupně se přechází na novější verzi 3.0, ale právě drobné odlišnosti neumožňují okamžitou změnu, starší projekty psané ve verzi 2.0 nelze beze změn spustit ve verzi novější.

Licencován je pod speciální Open Source licencí Python Software Foundation License.

5.2 Django

Obrázek 5.2: Logo rámce Django [https://www.djangoproject.com/community/logos/]

Django je jedním z nejstarších webových rámců v Pythonu, a je navíc asi nejvíce používaný. Původně byl vytvořen pro správu několika zpravodajských webů firmy The World Company v Lawrenci v Kansasu. Následné byl vydán, v červnu 2005, veřejně pod Open Source licencí BSD.

Jedná so o kompletní nástroj který usnadňuje tvorbu webových aplikací. Používá návrhový vzor model-view-controller. Model jsou nástroje pro přístup k databázi a celkově práce s daty. Usnadňuje přístup k databázi, často odstraní nutnost používat SQL příkazy. K datům se přistupuje jako k objektům z objektově orientovaného programování. Jedná se o takzvaný ORM (Object Relational Mapper) přistup. View je šablonovací systém. Umožňuje snadnou tvorbu vzhledu aplikace bez míchání logiky aplikace se vzhledem. Controller je samotná logika aplikace. Pro každé volání stránky se zavolá controller. Ten může pracovat s daty poskytnutými modelem a nakonec po zpracování odešle výsledek přes view a to přes šablonu kterou určí.

Tím, že Django obsahuje všechny části pro podporu programování serverové části aplikace, není nutné vybírat ještě další pomocné nástroje. A díky tomu že všechny nástroje patří pod jeden balík, lze mnohem snáze hledat i příčiny případných chyb v aplikaci – ostatní programátoři používají úplně stejnou sadu nástrojů. Obsahuje mnoho součástí. Systém pro formuláře, který překládá data mezi formulářem a databází. Zprostředkovatel mezi objektovým modelem v Pythonu a relační databází. Nástroje pro použití vyrovnávací paměti. Nástroje pro tvorbu šablon a možnost jejich rozšiřování.

Django je díky svojí zavedenosti mezi webovými programátory široce podporované, existuje kolem něj komunita která obvykle v případě potřeby poradí. Navíc díky velké mase uživatelů programátorů skoro každý problém už někdo někdy řešil. Navíc je Django stále intenzivně vyvíjené, takže nehrozí že by případná objevená chyba zůstala dlouho neopravena.

Výhodou týkající se komerčního prostředí je velmi mírná open source licence BSD. Licence není restriktivní vůči použití programu v komerčním prostředí, a dokonce umožňuje šíření upravené verze bez nutnosti poskytnutí zdrojových kódů. Úpravy samotného Djanga by sice neměly být potřeba. Ale, v případě že by tato potřeba v budoucnu nastala, nebude znemožněna licencí.

5.2.1 Spouštění webového serveru

Samotný rámec Django je potřeba nějakým způsobem spouštět, aby mohl být volán přes http protokol od vzdálených klientů. Pro spouštění existuje několik metod.

Přímo Django obsahuje jednoduchý webový server, určený pro vývoj a testování. Tento server zjednodušuje vývoj aplikací, ale není vhodný k ostrému nasazení. Důvodem je především neschopnost zpracovávat více požadavků současně. Server běží pouze v jednom vlákně, takže všechny požadavky zpracovává postupně. V reálném nasazení by i při malém využití aplikace často docházelo k dlouhému čekání klientů na odpovědi serveru. Zvláště v situaci, kdy některé požadavky budou vyřizovány delší dobu.

Velmi známým způsobem je spouštění přes WSGI [15]. Tento způsob podporuje hned několik webových serverů. Pravděpodobně nejoblíbenější je spouštění právě přes WSGI v kombinaci s webovým serverem nginx [10].

Dalším hodně používaným způsobem je FastCGI [7]. Jde o jeden z možných způsobů komunikace mezi webovým serverem a aplikací. Aplikace stále běží, a webový server jí předává všechny požadavky. FastCGI podporuje většina běžně používaných webových serverů.

Modul mod_python pro webový server Apache [4]. Tento způsob je méně častý než předchozí zmíněné, spousta společností jej však používá právě z důvodu dobré spolupráce s tímto velmi oblíbeným webovým serverem. Jedná se především o ty, kteří používají web server Apache a z nějakého důvodu nechtějí nebo nemohou měnit. Například používají nástroje na tomto serveru závislé, a chtějí používat pouze jeden webový server.

Výhodou Django je, že výsledná aplikace nijak nezávisí na použitém způsobu spuštění ani webovém serveru. Jednu aplikaci lze naprosto beze změn spouštět libovolným způsobem. Je pouze na správci, který způsob zvolí, programátor se výběrem nemusí zabývat.

5.3 Bootstrap

Obrázek 5.3: Logo rámce Bootstrap [http://blog.getbootstrap.com/]

Jedná se o velmi moderní nástroj pro vytváření webu a webových aplikací. Vytvořili jej vývojáři Mark Otto a Jacob Thornton z Twitteru. Původně se jednalo o interní nástroj který jim měl usnadnit práci. V roce 2011 jej vydali veřejně pod Open Source licencí MIT. Jeho cílem je umožnit pohodlně vytvářet webové aplikace, které budou navíc vypadat dobře, a to jak na širokých monitorech počítačů, tak na tabletech nebo i malých obrazovkách mobilních telefonů. Díky této již zabudované schopnosti lze bez práce navíc vyhovět

požadavku na vytvoření mobilní verze aplikace. Výrazně usnadní práci při tvorbě náročných moderních webových aplikací, a to bez zhoršení kvality výsledné aplikace. Dochází k zjednodušení vývoje oproti používání různých nezávislých nástrojů, které nejsou nijak sjednocené a způsobují nejednotnost zdrojových textů webu. Může se jednat o jakéhokoliv uživatelské rozhraní ve webové aplikaci. Není podstatné jestli to je uživatelské rozhraní v administraci nebo běžná uživatelská aplikace.

Nabízí podporu nejrůznějších webových technologií a mnoha prvků, které je možné velmi snadno vložit do vytvářené stránky. Pro používání rámce Bootstrap jsou nutné základní znalosti tvorby webových stránek. Požadované interaktivní nástroje, například tlačítka, boxy, menu a další prvky je možné vložit pouze za použití jednoduchého HTML a CSS.

Kromě základních možností předpřipraveného dobře vypadajícího vzhledu přizpůsobenému právě používanému zařízení umožňuje použití dalších klientských webových nástrojů. Například nástroje na tvorbu tabulek které budou mít stránkování, možnosti seřazení podle různých kritérií, nebo filtraci položek. Nebo nástroje na tvorbu grafů, kdy ze serveru stačí dodat data která si přejeme zobrazit a nástroj z nich vytvoří pěkně vypadající graf.

Apache licence kterou je licencován Bootstrap umožňuje bezproblémové použití nástroje Bootstrap i v komerčních aplikacích (což je zde důležité). Mnoho Open Source licencí použití programů v komerční sféře komplikuje nebo dokonce znemožňuje.

5.3.1 Bootstrap šablona SB Admin

Administrační šablona [13] pro Bootstrap. Samotný Bootstrap je kolekce nástrojů. V případě jasně definovaných požadavků na aplikaci lze použít některou z existujících šablon. V našem případě aplikace musí a bude vypadat naprosto zřejmě jako administrační rozhraní. Proto lze využít už poskládanou kolekci nástrojů určených právě pro administrační rozhraní.

Licence šablony je Open Source a umožňuje použití pro libovolné účely zcela zdarma.

Šablona používá mimo samotného Bootstrapu z doplňkových nástrojů Tablesorter na vytváření tabulek, které lze řadit podle různých hodnot. Pro vytváření grafů pak dává na výběr nástroje morris.js a Flot. Samozřejmě nic nebrání, v případě že by tato potřeba nastala, použití dalších nástrojů.

5.4 MariaDB

Obrázek 5.4: Logo databáze MariaDB [https://mariadb.org/]

Jedná se o relační databázi vzniklou jako větev MySQL [9]. MySQL byla odkoupena firmou Oracle, a ne všechny nové vlastnosti byly uvolňovány jako Open Source. Z tohoto důvodu z iniciativy původnich vývojářů MySQL vznikla tato nová Open Source odnož. Původně byla zcela totožná s MySQL, novější verze pak přidávají nové vlastnosti, které nejsou zcela shodné s vlastnostmi MySQL. Avšak rozhraní pro přístup k databázi zůstává zcela kompatibilní.

MariaDB protokoly pro připojování a používání databáze jsou shodné s protokoly používanými v MySQL. Proto není problém MariaDB a MySQL dle potřeby nahrazovat. Taktéž základní vlastnosti a chování databází je téměř stejné.

Databázi MariaDB jsem vybral z důvodu omezení rizika špatného výběru technologie pro implementaci. Nemusí být za všech okolností nejvhodnější, ale znám ji a mám zkušenosti s tím jak ji vhodně používat. Navíc MySQL patří mezi několik málo databází oficiálně podporovaných webovým rámcem Django, A, jak již bylo řečeno, MariaDB je s MySQL kompatibilní.

Kapitola 6

Možné problémy a rizika

V každém softwarovém projektu mohou nastat různě závažné problémy, které mohou projekt ohrozit. V nejhorším případě by mohlo dojít až k nedokončení projektu, ale i menší problémy mohou způsobit nemalé komplikace. Může se jednat o zdržení projektu, omezení funkcí, nebo třeba špatnou použitelnost pro uživatele.

6.1 Špatný odhad potřebnosti projektu

Projekt může být zcela bezchybný podle zadání, přesto se může stát že nakonec nebude využit v praxi, protože nebude mít praktické využití. U spousty projektů se jedná o opodstatněný problém, například pokud se jedná o zcela novou, inovativní službu, která zatím nemá žádnou ani podobnou konkurenci. V našem případě by k takovému problému dojít nemělo, protože potřeby jsou přesně dané schopností prodejních zařízení. Software bude využívat toho co prodejní zařízení podporují. Navíc už existuje předchozí verze tohoto softwaru, takže zadání nebylo pouze odhadované, ale vychází z reálných už ověřených potřeb.

6.2 Nedostatečná analýza

V analýze požadavků by mohlo dojít k opomenutí podstatného požadavku. Tento problém nelze nikdy zcela vyloučit. Naštěstí, pokud by opomenutý požadavek nebyl příliš zásadní, lze se k tomuto kroku vrátit, doplnit jej, upravit návrh tak aby byl požadavek splněný ale opravy byly co nejmenší, a změny naimplementovat. Postup však není úplně snadný, proto bylo potřeba analýze věnovat dostatek pozornosti, a všechny možné požadavky zanalyzovat, a ty požadované pak zahrnout do řešení. Pokud by došlo k přehlédnutí podstatné potřeby která by měla velký vliv na výslednou aplikaci, hrozí, v případě objevení až po implementaci, nutnost celou implementaci vytvořit znovu.

6.3 Nevhodný návrh

Nevhodně navržená aplikace by se velmi špatně implementovala. Zbytečně by docházelo ke zvyšování složitosti. Což je v rozporu s požadavkem na bezchybnou funkčnost. Proto je potřeba vytvořit kvalitní, co nejjednodužší návrh, který bude snadno implementovatelný. Velkou výhodou u tohoto projektu byla relativní zřejmost při tvorbě návrhu. Už přímo z požadavků na funkčnost vyplývá základní rozdělení aplikace na části, kdy každá část má už předem specifikované určení. V případě že by bylo nutné vytvořit nějaké změny

v návrhu v průběhu imlementace, nebudou tyto změny nijak zásadní, a půjdou udělat poměrně snadno. Výrazné změny v návrhu hrozí v případě opomenutí důležité funkce už při analýze.

6.4 Nekvalitní implementace a nebo nedostatečné testování

Kvalitně navržený produkt, který nebude kvalitně fungovat je k ničemu. Proto je nutné provést kvalitní implementaci, která bude průběžně testována. Nekvalitně vytvořenou aplikaci lze opravit, bez zanechání větších následků, pouze dokud nebude nasazena na mnoha místech. Nasazením nekvalitní aplikace získáme od uživatelů špatnou reputaci, a té se půjde jenom obtížně zbavit. Proto je nezbytné neopomenout co nejdelší a nejdůkladnější testování.

6.5 Špatný nebo nevhodný výběr technologie

Tento problém už dodatečně nelze vyřešit bez kompletní nové implementace. Proto jsem všechny použité technologie vybíral z těch, se kterými už mám alespoň nějaké praktické zkušenosti. Nemohu vyloučit že pro některou část by technologii nešlo vybrat lépe. Ale na druhou stranu u všech částí vím že daná technologie či daný programovací jazyk bude vyhovovat pro daný účel, a nebude tvořit překážku pro úspěch projektu. Navíc jsem při výběru výrazným způsobem zohledňoval snadnost použití zvolené technologie, čímž si usnadním dodržení bezchybné funkčnosti.

Kapitola 7

Analýza

V této kapitole se zaměříme na podrobnější požadavky, které je potřeba při implementaci splnit. Zanalyzujeme požadavky vyplývající ze zadání a upřesníme tím priority při tvorbě softwaru.

7.1 Dostatečný výkon

Jedním ze základních požadavků je zajištění dostatečné rychlosti aplikace. Částečně lze rychlost aplikace řešit zakoupením vykonnějšího počítače, ale tento přístup lze aplikovat jenom v omezené míře a navíc přináší další zvyšování náladů na provoz.

7.1.1 Část pro komunikaci se zařízeními

V této části není prioritní co největší rychlost odpovídání na požadavky, odpovědět stačí do předepsaného několikavteřinového limitu. Pokud zohledníme možné zpoždění na síti, bylo by dobré na všechny požadavky odpovědět nejpozději přibližně do vteřiny. Podstatné je však zajistit dostatečný výkon. Prodejních zařízení připojených k jednomu ovládacímu serveru může být až tisíce. Každé toto zařízení může poměrně často komunikovat, někdy i s rozestupem jednotlivých zpráv v intervalu několika málo vteřin až desítek vteřin. Z tohoto vyplývá, že jeden server musí být schopen zpracovávát až stovky požadavků za vteřinu. Na jednotlivé požadavky sice není nutné odpovídat okamžitě, ale je potřeba zajistit aby se požadavky neztrácely a všechny byly v limitu odpovězeny. V případě že nebudou zodpovězeny zavčas, zařízení bude předpokládat že se požadavek ztratil při přenosu přes síť, a pošle jej znovu. Tím hrozí nekontrolované zahlcení centrálního serveru a v důsledku toho pak naprostá nefunkčnost ovládacího software.

7.1.2 Uživatelské rozhraní

U uživatelského rozhraní je nutné zajistit co nejrychlejší odezvu uživateli. Dostatečně rychlá odezva aplikace patří mezi základní činitele podílející se na celkové uživatelské přívětivosti aplikace. V této části není potřeba řešit celkový výkon, uživatelů připojených k aplikaci budou maximálně jednotky současně. Je však nutné těmto uživatelům zajistit uživatelský komfort při používání aplikace. Pro tento komfort je nutné mimo jiné zajistit rychlou odezvu aplikace. Rychlost odezvy je jedním z velmi důležitých parametrů pro zajištění přívětivosti aplikace pro uživatele.

7.1.3 Mezičást pro zpracování databáze

Slouží právě k zajištění dostatečně rychlé odezvy uživatelského rozhraní. Předpočítává různé souhrnné přehledy, stavy zařízení na základě seznamu jednotlivých zpráv a podobně. Na tuto část nejsou kladeny zvláštní výkonnostní požadavky, ale bylo by dobré zajistit pravidelné přepočítávání, aby data v uživatelském rozhraní neměla příliš velké zpoždění oproti skutečnosti. Pokud by zde docházelo ke zpoždění, důsledkem by byla neaktuální data zobrazovaná v uživatelském rozhraní. Ne však pomalost při používání tohoto rozhraní. Neaktuální data v rozhraní by pak znamenala neaktuální informace například o chybách zařízení a v důsledku pak zpožděné řešení těchto chyb. Přílišná neefektivita této části aplikace může znamenat omezení pro ostatní části. Především v případě přílišného zatěžování databáze může dojít k omezení rychlosti ostatních částí při databázových operacích. Proto je třeba dbát především na zátěž na databázi.

7.2 Moderní uživatelské rozhraní

Pod moderním uživatelským rozhraním si lze představit mnoho různých věcí. Někdo si představí spoustu barev, moderní grafiku, někdo například napojení na sociální sítě. Spousta lidí považuje za důležité aby se s aplikací dobře pracovalo přes mobilní telefony nebo tablety. Pro účely administračního softwaru je potřeba definovat konkrétní požadavky, jejichž kvalitní zvládnutí bude znamenat moderní uživatelské rozhraní, které si co nejvíce uživatelů oblíbí.

Velmi důležitou součástí projektu je kladné přijetí od uživatelů, v našem případě lidí kteří budou prodejní zařízení spravovat. A uživatelé kromě bezchybné funkčnosti požadují taktéž aby se jim se softwarem dobře pracovalo. A chtějí aby uživatelské prostředí vypadalo co nejlépe. Rozhraní musí být především intuitivně použitelné, bez studia složitých návodů k použití. Dalším požadavkem je použitelnost z mobilních zařízení. Přehlednost výstupů a dobrá orientace v poskytovaných informacích musí být samozřejmostí.

Pro maximální zjednodušení splnění tohoto požadavku na moderní uživatelské rozhraní bude vhodné zvolit klientský webový rámec Bootstrap. Tento rámec má předpřipravené pěkně vypadající elementy stránky jako jsou například menu, tlačítka nebo formuláře. Další součástí jsou klientské nástroje pro různé funkční prvky. Například různé druhy grafů, seřazování tabulek, nebo přepínání obsahu na stránce. Pro pohodlné zobrazování aktualních informací uživateli je potřeba zajistit automatické načítání nových informací v už otevřených stránkách.

7.3 Bezchybná funkčnost

Bezchybná funkčnost aplikace zní jako samozřejmost, ale praxe ukazuje že se o samozřejmost nejedná. K bezchybné funkčnosti nové aplikace je potřeba splnit několik předpokladů. Nejprve musí být aplikace vhodně navržena, nesmí obsahovat zbytečné komplikace. Čím jednodušeji bude aplikace navržena, tím snazší bude její realizace. A snazší realizace znamená menší počet příležitostí k chybě. Navíc příliš komplikovanou funkčnost bude pro používání obtížné vysvětlit uživateli.

I po dodržení předpokladů v předchozím odstavci je naprogramování zcela bezvadné aplikace ihned na první pokus velmi obtížné, ne-li nemožné. Proto jako další důležitý předpoklad je kvalitní testování. Jednak je potřeba testovat už v průběhu implementace jednotlivé části aplikace. A nakonec před vydáním aplikace uživateli je potřeba všechno znovu

důkladně otestovat. Kvalitní testování dokáže odhalit prakticky všechny chyby, které se mohou reálně projevit.

7.4 Možné vlastnosti aplikace

Zařízení, které bude software obsluhovat, mají definovanou množinu schopností a funkčnosti. S této množiny, a zároveň s požadavků v zadání, je potřeba vycházet při analýze možností co by ovládací software mohl a měl umět.

7.4.1 Statistické funkce

Velmi užitečnou, prakticky základní, schopností ovládací aplikace jsou statistické funkce. Každý automat hlásí každou transakci ovládací aplikaci. Proto lze vytvářet statistiky z různých pohledů. Souhrnně pro všechna zařízení, jednotlivě pro každé zařízení zvlášť, pro různá časová období, porovnání jednotlivých zařízení a podobně. Počet plateb mincemi a platebními kartami, případně i jinými způsoby.

Všechny statistiky lze zobrazovat v tabulkách. Tyto tabulky by mělo jít seřazovat podle různých kritérií. Nejen podle datumu, ale i například podle velikosti vybrané částky. Dalším užitečným pohledem jsou grafy. Vykreslení grafů pro různé statistiky zajistí výrazné zpřehlednění pro uživatele. Tito se budou moci ve statistikách lépe orientovat, a snadno uvidí různé trendy.

7.4.2 Hlášení chyb a varování

Z pohledu servisního technika naprosto nejdůležitější funkce, která umožní chod zařízení s co nejmenším počtem co nejkratších výpadků. Pro zajištění co nejbezproblémovějšího chodu zařízení je nutné mít velmi dobrý přehled o chybách a varováních na jednotlivých zařízeních. Zařízení posílají různé zprávy které identifikují stav zařízení. Stav může být žádná chyba, varování a chyba. Ideální situace je když budou všechna zařízení v bezchybném stavu. Pokud je zařízení ve stavu varování, znamená to že sice stále funguje, ale brzy z nějakého důvodu fungovat přestane. Typickým příkladem varování je docházenící nějaká náplň. Stav chyba znamená nefunkční zařízení, například proto že náplň už zcela chybí.

7.4.3 Nastavení

Sem lze zahrnout veškerá nastavení která lze s ovládacím softwarem provádět. Jednak sem patří nastavení která se týkají samotné aplikace, pak nastavení k zobrazování jednotlivých zařízení. A nakonec nastavení která se posílají přímo prodejním zařízením.

Nastavení aplikace

Jedná se o nastavení které se žádným způsobem neposílá do prodejních zařízení. Přesto je toto nastavení velmi důležité. Jednak musí v aplikaci fungovat nastavení umístění centrálního serveru. Tedy především portu na kterém aplikace naslouchá jednotlivým zařízením. Dále pak může být nastavitelné kde se bude spouštět webové uživatelské rozhraní. Případně může jít nastavit chování a vzhled rozhraní aplikace, například barvy, umístění menu a další možnosti podle požadavků vznesených uživateli. Přestože taková nastavení nejsou nezbytná, uživatelům zpříjemní práci.

Nastavení k jednotlivým zařízením

Týkají se jednotlivých zařízení, ale informace jsou pouze pro potřeby zobrazování v uživatelském rozhraní. Například by mělo jít nastavovat jména zařízení, jejich fyzické umístění případně poznámky k zařízení a podobně. Tato nastavení slouží k následné lepší použitelnosti rozhraní. Opět se nejedná o nezbytné nastavení, ale slouží k zjednodušení práce uživatelů.

Nastavení odesílaná do zařízení

Nejdůležitější část nastavení. Zařízení podporují provádění vzdálené správy, a toto je pak potřeba implementovat i do ovládacího softwaru. Vzdáleně lze nastavovat jednotlivé ceny a prodávané varianty. Ale lze i provádět vzdálenou aktualizaci vnitřního ovládacího softwaru v zařízení. Možnosti nastavení jsou dány schopnostmi zařízení, zde proto nelze vymyslet cokoliv by se uživateli líbilo, je nutné vycházet z existujících schopností.

Kapitola 8

Návrh

Jak už bylo řečeno výše, aplikace bude rozdělena na dvě hlavní a jednu pomocnou část. Pojďme si podrobně rozebrat a navrhnout tyto tři jednotlivé části.

8.1 Část pro komunikaci se zařízeními

Část musí umět komunikovat jako server, a to jak na protokolu TCP tak UDP. Server musí naslouchat požadavkům na těchto protokolech, přijímat zprávy od jednotlivých zařízení, tyto zprávy vyhodnocovat, ukládat a odpovídat na ně.

Tuto část si rozdělíme na dvě. První bude mít na starosti samotnou komunikaci, a bude ve dvou verzích, pro TCP a pro UDP. Druhá pak bude zprávy ukládat do databáze. Části budou muset velmi těsně spolupracovat, proto budou součástí jednoho procesu. A pouze z důvodu rychlosti budou běžet ve dvou různých vláknech.

8.1.1 Principy firemního komunikačního protokolu

Komunikační protokol funguje na principu zasílání zpráv po síťových protokolech TCP nebo UDP. Klient odešle zprávu na ovládací server, ten odešle klientovi odpověď. V případě bezestavového protokolu UDP se jedná o dva samostatné datagramy, jeden s požadavkem, druhý s odpovědí. V případě stavového protokolu TCP se jedná o jedno spojení, které navazuje klient. Klient naváže spojení, odešle zprávu. Server zkontroluje správnost zprávy a odešle klientovi odpověď. Ihned na to uzavírá spojení. V případě, že chce klient poslat další zprávu, musí si otevřít nové spojení.

8.1.2 Část první – varianta TCP

TCP je stavový protokol, který navazuje spojení. Navázání spojení, příjem zprávy od klienta a odpověď zpět trvá nějaký čas, řádově v desetinách vteřiny, ale někdy může trvat i více než vteřinu. Takovýto postup zpracování v jednom vlákně zcela neodpovídá požadavku na zpracování stovek zpráv za vteřinu. Problém lze řešit dvěma různými způsoby.

Použití více vláken

Velmi obvyklý způsob s poměrně jednoduchým principem. Server naslouchá na pokusy o navázání spojení od klientů. Jakmile dojde k požadavku na spojení, server vytvoří nové vlákno a tomuto předá obsluhu nově vznikajícího spojení. Tento princip má však zásadní

problém pro praktické nasazení u aplikací vyžadujících velké množství spojení. Jednak vytvoření vlákna není výkonnostně vůbec levná záležitost. Pro vytváření stovky nových vláken každou vteřinu je potřeba poměrně výkonný hardware. A i při přistoupení na požadavek nasazení na výkonném hardware zůstává jiný problém. Nelze mít otevřeno současně příliš velké množství vláken. Vzhledem k očekávané zátěži ve stovkách spojení každou vteřinou a možnou celkovou dobou trvání jednoho vlákna až vteřinu, znamená tento přístup mít vytvořeno stovky vláken. A to je prakticky téměř nereálné. Naštěstí existuje ještě jiný méně obvyklý způsob zpracování takového množství spojení.

Asynchroní zpracování

Zpracování navázání spojení, jednoho požadavku, odpovědi a ukončení spojení trvá velmi dlouhou dobu. Ale jenom naprosto minimální čas procesor opravdu něco dělá, drtivou většinu doby se čeká na odpovědi ze sítě. Po tento čas je standardně vlákno blokováno, nic aktivního nedělá a pouze zabírá systémové prostředky.

Asynchroní zpracování v principu využívá neblokujícího čtení ze socketů. To znamená, že nedochází k čekání na odpovědi ze sítě. Z toho vyplývá, že dané vlákno nadále pracuje, zpracovává existující požadavky, naslouchá novým požadavkům. Z toho pak vyplývá, že zpracování všech požadavků může zajistit jedno jediné vlákno, a to dostatečně rychle, protože zvládá současně zpracovávat stovky aktivních spojení.

Jako příklad běžně používané aplikace, která využívá asynchronní zpracování, lze uvést webový server Nginx. Naproti tomu webserver Apache, náchylný na útoky typu Slowloris [14], používá vlákna.

8.1.3 Část první – varianta UDP

Použití UDP je v principu znatelně jednodušší. Tento protokol nenavazuje spojení, nedochází tudíž nikde k blokování požadavků a lze jednoduše použít zpracování v rámci jednoho vlákna. Nevýhodou protokolu UDP je nespolehlivost – nikde nezaručuje doručení odeslaných zpráv. Tento problém ale snadno řeší firemní protokol a chování platebních zařízení. Pokud zařízení nedostane odpověď na odeslanou zprávu v zadaném časovém limitu, odesílá zprávu znovu. Zprávy jsou číslované, takže pokud server dostane jednu zprávu podruhé, předpokládá, že se někde ztratila odpověď. Takže pošle odpověď znovu. Pouze pro případnou diagnostiku problémů na síti tuto skutečnost zaznamená, ale uživateli ji nebude ukazovat jako chybu.

8.1.4 Část druhá – ukládání do databáze

První část zkontroluje kontrolní součty a korektnost zprávy. Klientovi odešle odpověď a původní zprávu uloží do fronty pro ukládání do databáze. V této druhé části už není důležité okamžité zpracování, ale příliš dlouhé zpracování znamená přílišné zpožďování údajů zobrazovaných uživateli. Pokud doba ukládání zpráv bude v jednotkách, maximálně malých desítkách vteřin, rychlost je zcela dostačující.

Dále už tato část pracuje poměrně jednoduše. Vezme zprávu z fronty a uloží ji do databáze. Pokud se vhodně nastaví klíče v tabulkách, nastaví se unikátnost na položku číslo zprávy, samotné ukládání zajistí detekci duplicitních zpráv. Po detekci duplicitní zprávy není třeba nijak zvlášť reagovat, například odesláním jiné odpovědi klientovi, pouze se tato skutečnost zaznamená. A duplicitní zpráva se samozřejmě neuloží, což je správné a požadované chování.

8.2 Mezičást pro zpracování databáze

Přímo navazuje na ukládání dat do databáze. V kroku přijetí zprávy a následnému uložení do databáze se zprávy ukládají tak, jak jsou, bez nějakého dalšího zpracování. Pouze se jednotlivé položky rozdělí do jednotlivých sloupců v tabulce. Ale práce s takto uloženými daty v klientské aplikaci by byla velmi pomalá. Jenom například zjištění aktuálního stavu automatu by znamenalo projítí všech zpráv o změnách stavu a postupná modifikace stavu od počátku až po aktuální okamžik. Nebo třeba vytváření souhrnných statistik by znamenalo procházení všech zpráv o platbách a sčítání položek. A teď si vezměme požadavek na poměrně jednoduchý údaj, kolik všechna zařízení vydělala od začátku provozu. Nutnost sečíst obrovské množství údajů z databáze navíc současně z několika tabulek. Proto je nezbytné průběžně počítat některé údaje a ukládat je. Takto spočítané údaje lze vždy spočítat znovu na základě tabulek s přijatými zprávami, ale jejich neustálé kompletní přepočítávání by bylo velmi náročné na výkon. Proto je nutné umět podle nově přijatých zpráv průběžně dopočítávat aktualizace uložených údajů.

8.3 Uživatelské rozhraní

Jedná se o standardní webovou aplikaci. Data jsou v tabulkách v běžné SQL databázi, v našem případě MariaDB, což je databáze kompatibilní s MySQL. Klienti přistupují přes webový prohlížeč.

Na straně serveru bude použit programovací jazyk Python a webový rámec Django. Pro stranu klienta musí být generováno a posíláno HTML, případně data do JavaScriptu, ale pro usnadnění dalších funkcí bude použito klientského webového rámce Bootstrap. Použití tohoto rámce výrazně zjednoduší především tvorbu základního vzhledu aplikace, ale také usnadní vytvoření uživatelských prvků typu tabulka s možností seřazení, vykreslené grafy a podobně. Další výhodou použití tohoto rámce je automatické vytvoření přizpůsobivého vzhledu, Bootstrap stránka se automaticky přizpůsobí šířce okna webového prohlížeče. Podle šířky okna prohlížeče se přeskládají prvky na stránce, a to buďto vedle sebe, nebo v užších oknech pod sebe. Díky tomu je možné jednu verzi aplikace pohodlně používat kromě počítačů i na tabletech a mobilních telefonech.

8.3.1 Funkce uživatelského rozhraní

Zde lze zařadit všechno co aplikace bude umět z pohledu uživatele. Jedná se o podporu všech vlastností aplikace popsaných v kapitole analýza. Následuje podrobnější popis jednotlivých funkcí rozdělených na stránky v aplikaci.

Úvodní stránka – Dashboard

Úvodní stránka je první co uživatel-správce zařízení uvidí po otevření webové aplikace pro správu prodejních zařízení. Proto je nezbytné všechny podstatné a souhrnné informace zobrazovat přímo zde. Jedná se o hlavní rozcestník a nejdůležitější stránku aplikace.

Naprosto nezbytnou funkcí je upozornění na nefunkční zařízení, případně na zařízení ve stavu varování. Proto jako první a nejdůležitější informací, která bude na úvodní stránce, jsou počty zařízení podle jejich stavu. S tlačítek zobrazujících tyto počty zařízení pak musí být umožněn proklik na seznam zařízení v daném stavu. Uživatel pak rychle uvidí seznam

všech například nefunkčních zařízení včetně příčin nefunkčnosti. Pro lepší uživatelskou orientaci v případě většího počtu zařízení je možné spočítat a zobrazit procenta funkčních a nefunkčních zařízení.

Dále by bylo užitečné zde zobrazit základní souhrnné přehledy. Graf o celkových příjmech všech zařízení a počtu plateb po jednotlivých dnech, celkové příjmy všech zařízení a počty plateb. Rozložení plateb podle platební metody – platební karty nebo mince. Pokud na úvodní stránce ještě zbyde místo, bylo by možné zobrazit seznam několika posledních přijatých stavových zpráv s odkazem na výpis všech zpráv.

Vzhledem k důležitosti stránky by bylo vhodné zajistit automatickou aktualizaci informací na stránce bez znovunačítání. Případnou poruchu některého ze zařízení tak uživatel uvidí okamžitě a nebude se muset zdržovat ručním znovunačítáním stránky.

Seznam všech zařízení s možností zobrazení detailu

Zásadní funkčnost ovládacího softwaru. Seznam zařízení by měl jít seřazovat podle různých kritérií, případně filtrovat podle stavu zařízení. Například schopnost zobrazit všechna nefunkční zařízení může být velmi užitečná. Přímo v seznamu zařízení lze zobrazovat i některé podrobnější informace, například celkové příjmy zařízení, příjmy zařízení za poslední den a podobně. Podle těchto údajů může taktéž fungovat řazení. Kliknutím na konkrétní zařízení v seznamu zařízení se zobrazí jeho detail s podrobnými informacemi.

Detail musí obsahovat informace o nejen souhrnné funkčnosti zařízení, ale i informace o funkčnosti a nefunkčnosti jednotlivých částí, resp. informaci o tom z jakého důvodu je zařízení nefunkční. Jako doplňkovou informaci by měl tento detail zobrazovat další podrobnosti o zařízení jako je verze vnitřního software. Dále musí součástí detailu o zařízení být podrobnosti o uskutečněných platbách, množství a částka plateb podle jednotlivých typů plateb. Pro zjištění případných nejasností v rozhraní musí být umožněno zobrazit seznam všech typů zpráv které dané zařízení posílá – informace o jednotlivých platbách i jednotlivé stavové zprávy tohoto zařízení.

Součástí detailu o zařízení by měla být podpora nastavení tohoto zařízení. Uživatel si bude moci zařízení pojmenovat, případně si napsat další poznámky, umístění zařízení a podobně. V případě že budeme podporovat i změny nastavení v zařízení, jako je nastavení cen, aktualizace vnitřního software a podobně, zde je vhodné místo kam v ovládacím software tuto funkčnost vložit.

Výpis příchozích plateb s možností filtrace podle zařízení

Výpis příchozch plateb je velmi důležitá funkčnost, umožňuje zjistit přesné podrobnosti o platbách včetně přesného času platby, výše platby, lze z plateb vyčíst nejmenší rozestup mezi jednotlivými platbami, časy kdy k platbám dochází a libovolné další informace bez toho aby tato schopnost byla speciálně implementovaná v uživatelském rozhraní. Do výpisu filtrovaného podle konkrétního zařízení se bude možné prokliknout z detailů jednotlivých zařízení. Tento výpis musí jít řadit podle jednotlivých položek, ať už se jedná o číslo zařízení, datum a čas platby, nebo výši platby. Případně filtrovat podle jednotlivých zařízení, nebo i typů platby.

Výpis stavových zpráv s filtrací podle zařízení

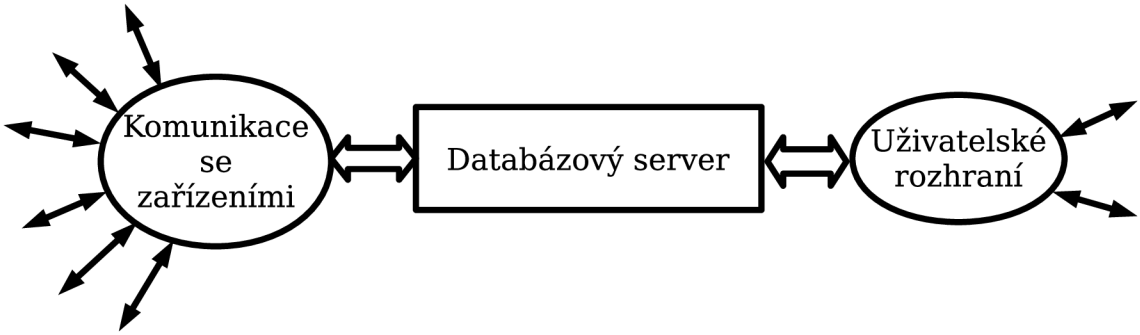
Podobný princip jako v předchozím bodu, pouze se stavovými zprávami. Taktéž je možné se do filtrované verze seznamu prokliknout z detailu zařízení. V seznamu zpráv budou

tentokrát informace o významu zprávy, jestli se jedná o nahlášení chyby nebo varování, případně zpráva informující o vyřešení problému. Nebo informační zpráva která neznamená změnu stavu automatu, ale pouze informuje o nějaké skutečnosti, která nemění použitelnost zařízení. Z výpisu stavových zpráv lze zjistit veškerou historii, mimo plateb, libovolného zařízení, co se týče poruch, oprav a další historie.

Globální nastavení

Minimální nutná funkčnost je umožnit uživateli nastavit port na kterém bude komunikační část naslouchat na zprávy od jednotlivých zařízení. Pak je zde vhodné místo pro umístění dalších možných globálních nastavení, například možnost nastavit heslo pro přístup k rozhraní, různé chování rozhraní a případné další vlastnosti. Je zde velký prostor pro případné rozšíření možností cokoliv nastavit.

8.4 Dodatečné změny návrhu



Obrázek 8.1: Výsledná architektura ovládacího software [vlastní]

V průběhu realizace projektu došlo ke změně priorit pro funkčnost výsledného programu. Program bude určen pro demonstraci možností ovládacího softwaru. V první fázi pro představení investorovi, poté pak pro předvádění zákazníkům, a to i na případných veletrzích kde bude součástí prezentace celého systému automatů.

Nebude proto v současné verzi kladen důraz na schopnost obsluhovat velké množství zařízení. Zato je však nutné vytvořit použitelné uživatelské rozhraní, které bude celý systém prezentovat. Rozhraní musí být přehledné, rychlé a reprezentativní tak, aby mohlo nalákat případné zájemce o koupi tohoto systému. Prezentace bude probíhat s jednotkami automatů, a tyto musí program umět bezchybně a přehledně ovládat. Způsob ovládání by měl být natolik intuitivní, aby jej potenciální zákazníci rychle pochopili.

Kapitola 9

Implementace

Implementace probíhala z naprosté většiny podle návrhu. Prakticky došlo pouze k jedné menší změně v požadavcích. Vytvořená verze programu bude prozatím sloužit pouze k demonstračním účelům. Z tohoto důvodu ustoupil požadavek na schopnost obsluhovat velké množství zařízení do pozadí. S tímto požadavkem je potřeba počítat do budoucna, aby nemuselo dojít k úplnému přepisu programu, ale v demonstrační verzi postačuje zvládnout obsluhu několika jednotek zařízení. Naopak je třeba se výrazně zaměřit na uživatelské rozhraní. Právě podle uživatelského rozhraní bude aplikace hodnocena potenciálními zákazníky.

9.1 Komunikační část

Diky ustoupení od požadavku na velký výkon aplikace mohlo dojít k zjednodušení této části. Proto jsem se rozhodl sloučit původně rozdělené části které pracují s databází. Jedná se o část spadající pod komunikaci se zařízeními – ukládání zpráv do databáze. A pak o mezičást pro zpracování databáze. Každá přijatá zpráva je stejně jako v návrhu uložena do databáze. Ale okamžitě poté tato stejná část programu zaktualizuje souhrnné informace v databázi. Tímto spojením se ušetřilo řešení předávání zpráv do další části a testování optimální frekvence přepočítávání statistik s ohledem na uživatelskou přívětivost a dostatečnou rychlost. Současně však nedošlo k znemožnění rozdělení a tím zrychlení aplikace v budoucnu. Změnu bude možné provést bez většího zásahu do dalších částí aplikace včetně struktury databáze. Navíc zcela bez zásahu do uživatelského rozhraní.

9.1.1 Výsledná architektura

Serverovou část aplikace tvoří jediný program v současné době skládající se pouze z jednoho souboru. Tento program je rozdělen na tři vlákna. První dvě jsou komunikační, jedno komunikuje se zařízeními přes protokol TCP, druhé přes UDP. Třetí vlákno má na starosti ukládání dat do databáze a jejich předzpracování do vhodné podoby pro použití v uživatelském rozhraní. Vlákna používají sdílenou frontu pro předávání zpráv. První a druhé vlákno do fronty vkládají zprávy, třetí zprávy čte a ukládá do databáze.

Výsledný program se skládá z jednoho skriptu. Na začátku skriptu je jednoduchá třída řešící připojení k databázi. Zajišťuje automatické znovypřipojení k databázi v případě výpadku spojení. Následuje konfigurace připojení do databáze. Poté pomocné funkce, následuje třída která slouží k zpracování zpráv a uložení do databáze. Zbytek kódu pak řeší

síťovou komunikaci a vlákna programu. V případě dalšího rozšiřování programu by bylo vhodné zvážit rozdělení do více souborů – modulů.

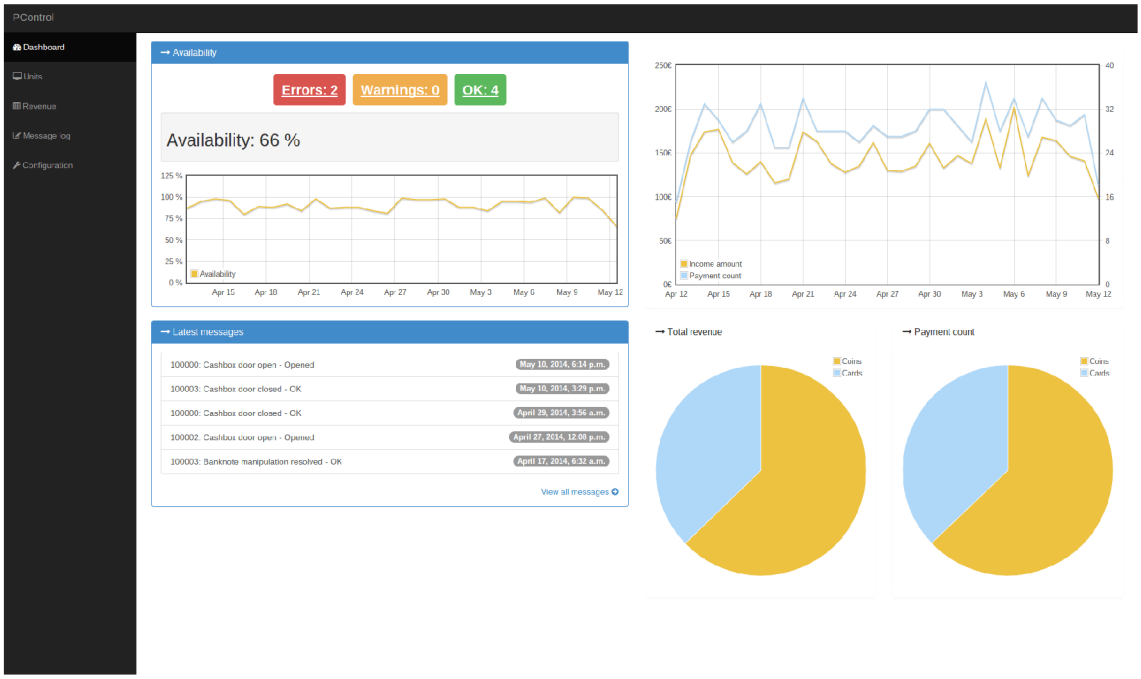
Simulátor zařízení

Simulátor zařízení není program nezbytný pro použitelnost softwaru. Ale byl velmi důležitý pro správný postup implementace kvůli možnosti testování ihned bez přítomnosti skutečných zařízení. Zmiňuji jej však ještě z jednoho důvodu. Jak již bylo řečeno, program slouží v současné fázi k demonstračním účelům. A k demonstračním účelům je důležitý i simulátor, a to kvůli schopnosti vygenerovat data která se tváří jakoby pocházela ze skutečných zařízení, a to navíc i zpětně. Simulátor v kombinaci s například jedním skutečným zařízením, může poskytnout velmi dobrou prezentaci systému s menší prací než zapojování mnoha zařízení kvůli každému předvádění systému.

9.2 Uživatelské rozhraní

Uživatelské rozhraní zůstalo beze změn oproti návrhu. Serverová část je implementována ve webovém rámci Django, klientská pak využívá Bootstrap.

9.2.1 Úvodní stránka



Obrázek 9.1: Dashboard - úvodní stránka aplikace

Na úvodní stránce se nachází přehled nejdůležitějších informací, které aplikace poskytuje, s možností prokliknutí k podrobnostem o zobrazované informaci.

V sekci dostupnost (availability) jsou zobrazené aktuální počty zařízení nacházející se v různém stavu. Každý stav lze prokliknout na seznam všech zařízení, které se v daném stavu nacházejí. Dále je tam graf celkové dostupnosti za posledních 30 dní.

Vedle sekce dostupnosti se nachází graf historie plateb za posledních 30 dní. Obsahuje počet plateb a celkovou částku plateb v jednotlivých dnech.

V sekci posledních zpráv (latest messages) je zobrazeno posledních pět stavových zpráv. Při kliknutí na libovolnou ze zpráv se zobrazí stránka zobrazující všechny zprávy, které odeslalo zařízení od něhož pochází prokliknutá zpráva. Pod seznamem je pak přímý odkaz na stránku zobrazující všechny stavové zprávy.

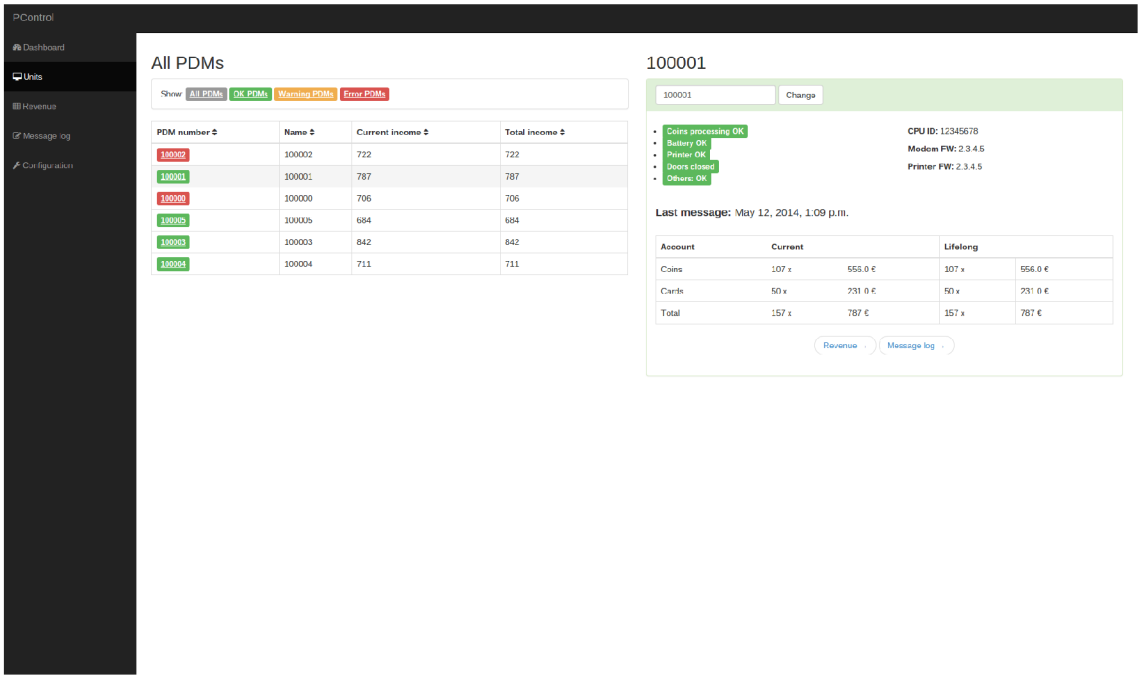
Vedle sekce posledních zpráv jsou koláčové grafy zobrazující celkové výdělky a celkový počet plateb. Grafy zobrazují poměr jednotlivých druhů plateb, na obrázku výše pouze platby mincemi a platebními kartami.

Informace zobrazované na úvodní stránce se automaticky aktualizují bez znovunačtení stránky.

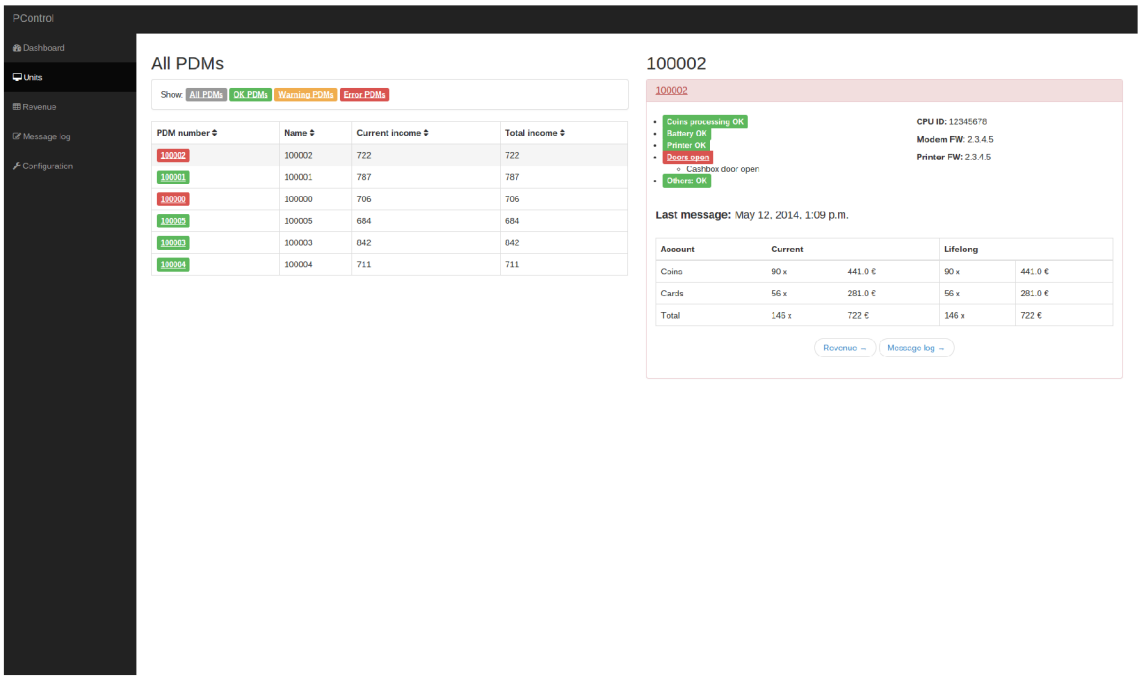
Ajax (Asynchronous JavaScript and XML)

Pro automatickou aktualizaci informací na stránce bez znovunačtení stránky byla použita technologie Ajax. Webová stránka se vždy periodicky dotazuje na aktualizace od webového serveru, a pokud aktualizace dostane, přepíše aktualizovanou část svého obsahu. V současné době existuje více způsobů jak zajistit aktualizace dat na stránce. Například WebSocket [7]. Tyto moderní metody sice přinášejí některé zajímavé novinky, ale jejich podpora zatím nemusí být v prohlížečích ideální. Jedná se o novou techniku, která zatím není příliš rozšířená, byť se na rozšiřování pracuje a v budoucnosti půjde velmi pravděpodobně o hodně zajímavou technologii. Ajax už je naprosto běžně podporován poměrně dlouhou dobu, proto jeho výběrem lze předpokládat omezení možných nečekaných problémů na minimum.

9.2.2 Seznam a detail zařízení



Obrázek 9.2: Detail zařízení se zobrazeným formulářem pro změnu pojmenování



Obrázek 9.3: Detail zařízení s chybovým stavem

V levé části je seznam jednotlivých zařízení. Seznam lze filtrovat podle stavu zařízení, a seřazovat podle jednotlivých sloupců tabulky.

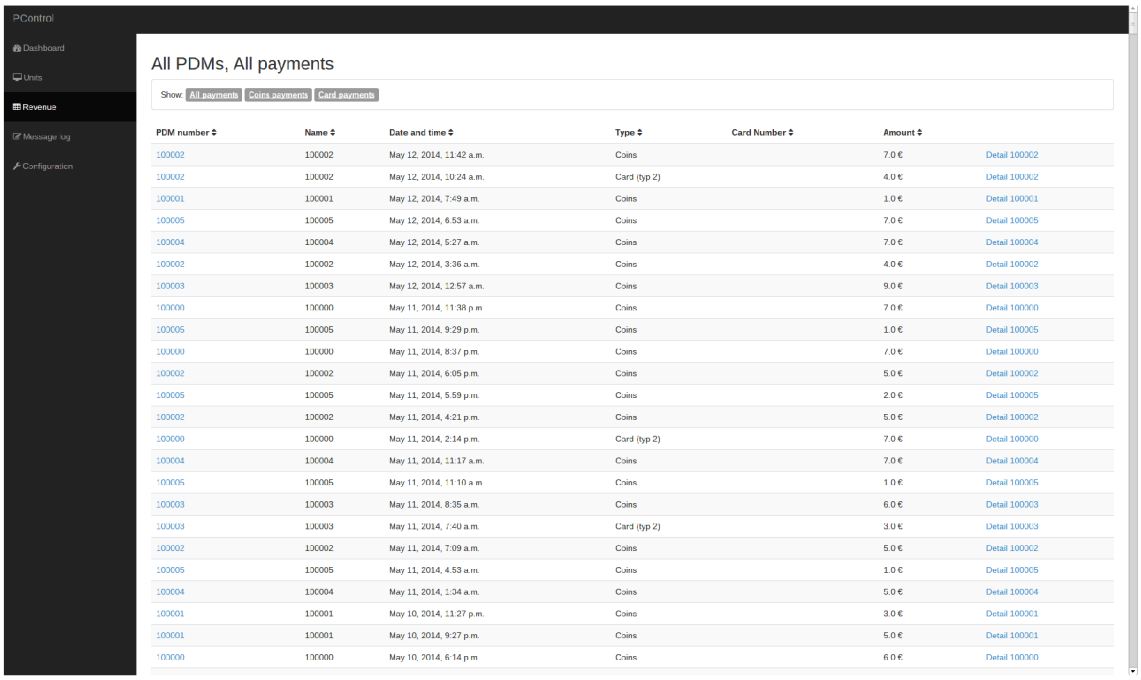
V pravé části je zobrazen detail zařízení. Jednotlivá zařízení se přepínají kliknutím na zařízení v seznamu v levé části. Kliknutím na název zařízení v barevném (podle stavu zařízení) titulku rámečku se zobrazí formulář kterým lze zařízení přejmenovat. Přejmenování má význam pouze v uživatelském rozhraní a usnadňuje uživateli orientaci v zařízeních. Název může obsahovat informace usnadňující uživateli identifikovat o které zařízení se jedná, snáze než pouze podle čísla. V horní části uvnitř rámečku jsou stavové informace o zařízení a verze vybraných součástí zařízení. Nad tabulkou je zobrazen datum a čas poslední libovolné zprávy od zobrazovaného zařízení. Tabulka obsahuje celkové příjmy zařízení. Pod tabulkou jsou odkazy na seznam plateb a stavových zpráv od daného zařízení.

9.2.3 Seznam příchozích plateb

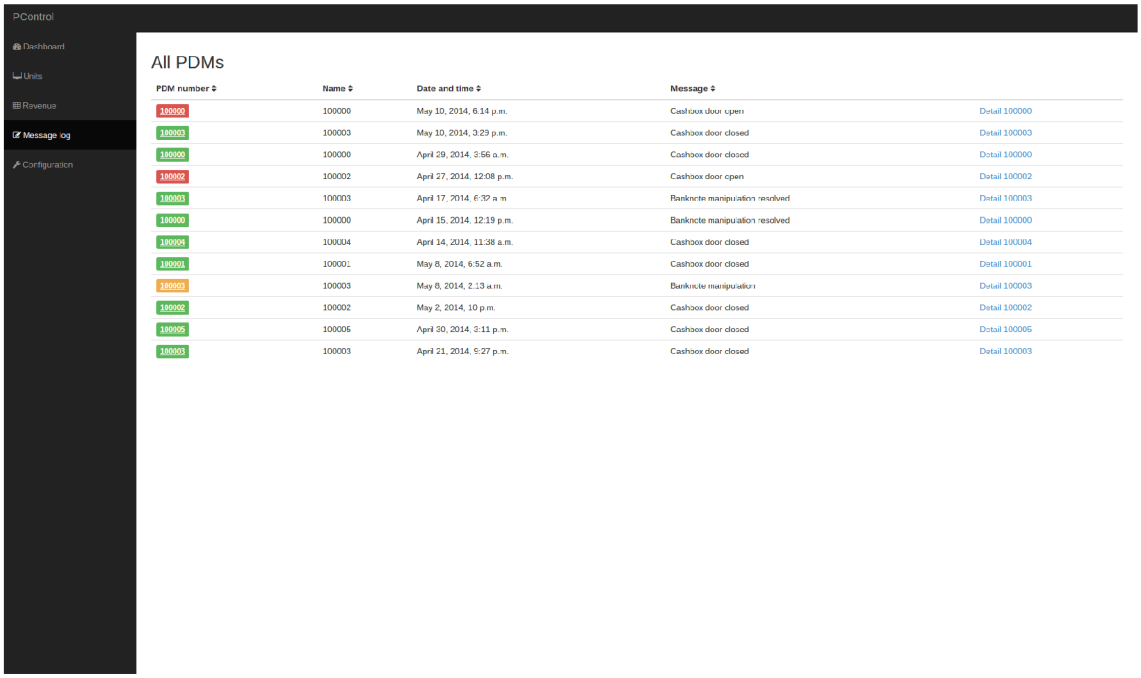
Seznam příchozich plateb lze seřazovat podle hodnot libovolného zobrazeného sloupce. Kliknutím na číslo zařízení se zobrazí pouze platby v daném zařízení. Nad seznamem jsou odkazy na zobrazení pouze plateb podle vybraného typu platby. Filtry podle zařízení a typu platby lze libovolně kombinovat.

9.2.4 Výpis stavových zpráv

Výpis stavových zpráv lze seřazovat podle hodnot ve sloupcích tabulky. Kliknutím na číslo zařízení se zobrazí pouze zprávy od daného zařízení. V posledním sloupci je odkaz na detail zařízení.



Obrázek 9.4: Seznam příchozich plateb, všechna zařízení, seřazeno podle času



Obrázek 9.5: Výpis stavových zpráv, všechna zařízení, seřazeno podle čísla zařízení

9.2.5 Nastavení

Nastavit lze port na kterém naslouchá komunikační část aplikace na zprávy od zařízení.

Kapitola 10

Testování

Nezbytnou součástí každého softwarového projektu musí být testování. Zvlašte v situaci, kdy bezchybná funkčnost je základním požadavkem, je důkladné testování velmi důležité. Testování lze rozčlenit do několika fází, podle aktuálního stavu vývoje projektu. Dále je nutné rozdělení na testování funkčnosti a testování přívětivosti uživatelského rozhraní.

10.1 Testování komunikační části v průběhu implementace

K testování komunikační části je nutné mít zařízení, které bude posílat data pro tuto část, jinak není možné testování provádět. Avšak testování se skutečnými zařízeními je poměrně zdlouhavé a nepohodlné. Proto bylo nutné, alespoň v počátku, vymyslet jiný způsob testování, který by byl rychlejší, pohodlňejší a nemusel jsem k němu mít neustálý přístup ke skutečným zařízením. Proto, už současně s programováním komunikační části, jsem programoval i simulátor zařízení. Simulátor generoval zprávy jakoby pocházely od několika zařízení a posílal je komunikační části aplikace. Ta je zpracovávala a ukládala, jako by se jednalo o skutečná data odesílaná skutečnými zařízeními. Tato fáze testu umožnila vytvořit funkční celek. Problém je ve skutečnosti, že jsem obě části programoval já. Tudíž tato fáze testu nemohla odhalit špatně pochopené části protokolu, ani neočekávané situace. Což neznamená, že toto testování nebylo důležité. Bez simulátoru by nebylo prakticky možné ověřit ani základní funkčnost programu a fáze testování se skutečnými zařízeními by se neúměrně protáhla.

10.2 Testování komunikační části se skutečnými zařízeními

Z již zmíněných důvodů nemohlo testování skončit u simulátoru. Proto musela nastat další fáze testu, kdy už byla sice většina chyb a problémů odstraněna, ale muselo dojít ještě k odstranění chyb, které nebyly simulátorem odhaleny. Proto v této fázi došlo k odstranění simulátoru a použití skutečných zařízení. Při testu bylo odhaleno špatně pochopené formátování některých položek, především položky obsahující datum a čas. Právě pro zopakování této chyby jak v simulátoru tak v aplikaci tyto chyby nemohly být odhaleny dříve. Po delším testování pak došlo k odhalení problému s vypršením připojení k databázi, ke kterému docházelo při malém zatížení, kdy jednotlivé zprávy měly mezi sebou příliš velký časový rozestup. Právě velký časový rozestup způsobil nepoužívání připojení k databázi po příliš dlouhou dobu a vypršení spojení. Protože testy se simulátorem neběžely v kuse po dostatečnou dobu, nemohl být problém odhalen dříve.

Testování se skutečnými zařízeními probíhalo poměrně dlouhou dobu v řádech jednotek měsíců. I když pouze s velmi malým počtem málo používaných testovacích zařízení používaných v kancelářích firmy právě za účelem testování různých vyvíjených součástí.

10.3 Testování uživatelského rozhraní

Úkolem projektu bylo zajistit nejen bezchybnou funkčnost, ale také uživatelskou přívětivost aplikace. Uživatelská přívětivost je velmi subjektivní záležitost a špatně se měří. Proto k jejímu zajištění nelze definovat a vytvořit nějaké metriky, které ji nějakým způsobem zjistí a změří, ale je nutné provést testování na lidech.

První fáze testování uživatelského rozhraní probíhalo s vývojáři ve firmě. Tito lidé průběžně přinášeli poznatky a návrhy k úpravám rozhraní a s jejich pomocí na základě jejich připomínek jsem vytvořil konečnou podobu rozhraní.

Další fáze testu uživatelského rozhraní proběhla s investorem. Ovládací aplikace byla představena společně s dalšími součastmi projektu platebních terminálů. Předvádění prováděli vybraní zaměstnanci a dle jejich slov bylo předvádění úspěšné. Nový ovládací software má být v budoucnu používán místo stávajícího řešení.

Kapitola 11

Zprovoznění

11.1 Uživatelské rozhraní

Nejprve je třeba rozběhnout uživatelské rozhraní. Postup je poměrně jednoduchý a lze jej rozdělit do následujících kroků. Všechny kroky vycházejí z použití na operačním systému Linux, v jiných systémech bude postup obdobný.

11.1.1 Instalace potřebných balíčků

Před samotným zprovozněním aplikace je nutné nainstalovat potřebné nástroje. Především je nutný Python, a to ve verzi 2. Vývoj probíhal konkrétně na řadě 2.7. Dále je nutné doinstalovat Django. Pro instalaci nejnovější verze je nejjednodužší postup použít nástroj pip (`pip2 install django`), který je součástí instalace Pythonu. Poté je nutné ověřit přítomnost a případně doinstalovat balíček pro podporu MySQL `MySQL-python`. Při instalaci je třeba ověřit, aby probíhala instalace pro python z řady 2. V případě, že bude program po spuštění hlásit nějaké chybějící moduly, je potřeba je doinstalovat. V různých distribucích mohou být zahrnuty v základní instalaci jiné moduly a doinstalovávané balíčky se v této situaci budou lišit.

11.1.2 Konfigurace připojení k databázi

Konfigurace databáze probíhá v souboru `settings.py` v podadresáři `interface`. Je nutné změnit obsah proměnné `DATABASES` které určuje způsob připojení k databázi. V tomtéž souboru je dále možné nastavovat časové zóny, případně lokalizaci. Je důležité neměnit typ databáze, protože komunikační část počítá s použitím MySQL a jinak by nefungovala.

11.1.3 Vytvoření databázové struktury

Součástí projektu je soubor `manage.py`, který je možné používat pro správu Django aplikace. Spustíme příkaz `manage.py syncdb`, čímž vytvoříme základní strukturu databáze. Skript se bude dotazovat jestli má vytvořit uživatele s právy administrátora. Na tuto otázku je možné odpovědět negativně, autentizace uživatelů není v této verzi řešena.

Po vytvoření struktury databáze je nutné nahrát do databáze konfigurační data. Soubor `mnemonics.csv` naimportujeme do tabulky `uiapp_mnr`. Bez tohoto kroku aplikace správně nerozezná význam stavových zpráv a tím pádem nebude korektně zobrazovat stav jednotlivých zařízení. Chybný import se pozná podle toho, že všechny zprávy budou v uživatelském rozhraní označeny jako neznámé.

11.1.4 Spuštění webového serveru

V tomto kroku je možných několik zcela různých postupů. Výběr záleží na požadavcích na náročnost zprovoznění, celkový výkon aplikace, případně na už používaných postupech pro provoz webových aplikací. Následuje popis několika možných způsobů zprovoznění. Není to kompletní výčet možných způsobů, ale pouze příklad základních možností. Pro otestování aplikace doporučuji použít první navrhovaný způsob.

Zabudovaný testovací web server

Nejjednodužší způsob zprovoznění který je pro testování nejvhodnější. Skript `manage.py` již byl použit pro vytváření struktury databáze. Použijeme tentýž skript, pouze s jiným parametrem – `manage.py runserver`. Tím spustíme jednoduchý web server na lokálním rozhraní na portu 8000. Pokud chceme zajistit přístupnost serveru i přes síť, použijeme další parametr – `manage.py runserver 0.0.0.0:8000`. Tečkami oddělené nuly značí naslouchání na všech adresách a síťových rozhraních bez omezení. Za dvojtečkou je číslo portu, které lze libovolňe zvolit. Pouze pozor, pro přístup na porty do 1024 jsou potřeba administrátorská oprávnění, server by v takovém případě musel běžet pod uživatelem root. Odkládání práv po připojení k privilegovanému portu, známé z komplexních serverových aplikací, zde není podporováno.

Nginx a WSGI

Složitější postup, který je však vhodný k produkčnímu nasazení. Následující kroky jsou přibližné, mohou se lišit podle konkrétního systému. Pro přesný návod by bylo nutné vybrat konkrétní systém včetně přesné verze.

V prvním kroku nainstalujeme web server Nginx a aplikační server uWSGI včetně rozšíření pro Python. Následně je třeba začít s konfigurací.

Následuje důležitá součást konfiguračního skriptu pro Nginx. Soubor bude nejpravděpodobněji umístěn na `/etc/nginx/nginx.conf`.

```
server {
 listen *:80;
 listen [::]:80;
 server_name localhost;
 root /var/www;
 error_log /var/log/nginx/error.log;
 access_log /var/log/nginx/access.log combined;
 location = /favicon.ico {
 log_not_found off;
 access_log off;
 }
```

```
location = /robots.txt {
allow all;
log_not_found off;
access_log off;
}
location / {
include uwsgi_params;
uwsgi_pass 127.0.0.1:9000;
}
location /static/ {
access_log off;
alias /var/www/uiapp/static/;
}
}
```

Webový server nginx máme nakonfigurovaný, nyní musíme zprovoznit aplikační server uWSGI, abychom zajistili zdroj dat pro nginx. Konfigurační soubory `uwsgi.xml` a `django.wsgi` jsou součástí zdrojových textů projektu, pouze v nich v případě potřeby upravíme cesty. Spuštění pak proběhne jedním příkazem `uwsgi_python27 -x uwsgi.xml:pcontrol`. Pokud nebude spuštění fungovat, je nutné přečíst poskytnuté chybové hlášení a poupravovat cesty podle potřeby. V každé linuxové distribuci se mohou umístění a případně i názvy souborů lišit. Možná bude nutné i doinstalovat chybějící součásti.

11.1.5 Konfigurace portu pro komunikační část

Ještě před spuštěním komunikační časti je nutné ve webovém uživatelském rozhraní v nastavení nastavit port, na kterém má komunikační část naslouchat na zprávy od zařízení.

11.2 Komunikační část

Pro komunikační část je potřeba Python řady 3, otestována je verze 3.3 a měla by fungovat ve všech novějších. Přes pip doinstalujeme podporu pro databázový server MySQL, `pip3 install PyMySQL`.

Zprovoznění komunikační části je znatelně jednodužší, než u uživatelské části. Soubor `server.py` je jediný program, který musí běžet, a to neustále pokud má komunikace fungovat. Téměř na začátku je konfigurace přístupu do databáze. Proměnná `db` obsahuje napojení na databázi, která by se měla lišit od databáze nakonfigurované v uživatelském rozhraní. Což však nevylučuje, že se může jednat o stejný databázový server. Proměnná `db2` musí obsahovat napojení na stejnou databázi jako uživatelské rozhraní. Není nutné žádné vytváření

databázové struktury mimo ty již vytvořené pro uživatelské rozhraní. Další potřebné tabulky si skript vytvoří automaticky.

Pro vygenerování testovacích dat slouží skript `client.py`. Tento skript náhodně vygeneruje data jakoby pocházela od několika různých zařízení za posledních 30 dní. Skript slouží jako simulátor zařízení, neukládá data do databáze, ale standardně, stejně jako skutečná zařízení, komunikuje přes síťový protokol s komunikační částí. Skript sloužil v první fázi pro testování. Nyní slouží jako generátor vzorku dat na kterém lze aplikaci odzkoušet, bez nutnosti používat skutečná zařízení. Nebo pro zvětšení vzorku dat při použití malého počtu zařízení.

Kapitola 12

Zhodnocení projektu

Objektivně výsledek jako autor hodnotit nemohu, proto se budu snažit držet faktů ke kterým došlo v průběhu implementace a popsat zpětnou vazbu získanou od druhých lidí.

Výsledkem projektu je program, určený k demonstraci možných vlastností. Vytvoření programu se povedlo, a to podle analýzy požadavků a v souladu s návrhem. Během projektu došlo pouze k jedné změně, a to odstranění požadavku na schopnost obsluhovat velké množství zařízení. Důvodem této změny byla změna priorit. Hlavní prioritou se stala možnost projekt prezentovat zákazníkům s tím, že předvádění bude probíhat na menším počtu zařízení. Bylo možné omezit množinu funkcionality, a to tak, aby omezení byla při předvádění co nejméně patrná, a zajistit že tato funkcionalita bude zcela bezchybně a intuitivně fungovat. S podporou pouze základní funkcionality se však počítalo už od začátku.

Program byl úspěšně předveden investorovi do projektu automatů. Bylo schváleno nahrazení původního ovládacího software tímto novým projektem. Jako součást systému automatů byl vystavován na mezinárodní výstavě v Bruselu, a to, i přes zatím omezenou funkcionalitu, už namísto původního ovládacího softwaru.

Kapitola 13

Pokračování projektu

Zájmem firmy je výsledky projektu využít, což znamená pokračování ve vývoji. Současný stav je prototyp určený pro demonstraci možností aplikace uživateli. Umí pouze základní funkcionalitu a není optimalizovaný pro velkou zátěž.

13.1 Rozšíření funkcionality

Současný stav aplikace je zaměřen na zobrazování přehledů a statistik. Jedná se o základ, který však není k plnému nasazení dostačující. Jako další krok je potřeba implementovat možnost do zařízení posílat nastavení a aktualizace ovládacího softwaru. Pak je třeba počítat s přibývajícími možnostmi zařízení. Protože i funkcionalita zařízení se s postupným vývojem rozšiřuje, je třeba každé nové rozšíření, které lze ovládat nebo má nějaký výstup, zařadit i do ovládacího software.

Jako další užitečné rozšíření může být řízení uživatelských práv. V současné verzi nejsou práva nijak řešena, každý uživatel může dělat všechno co software umí. Ale s přibývající funkcionalitou bude přibývat smysluplnost řešení práv uživatelů. Někdo, například účetní, může mít právo jenom číst finanční statistiky. Naopak technik například nemusí mít k finančním výsledkům vůbec přístup a pouze se dozví pokud je se zařízením nějaký problém. A nadřízený bude mít i oprávnění měnit ceny nebo jiné nastavení v automatech.

13.2 Optimalizace

I přes původní záměr zajistit velký výkon aplikace, došlo ve fázi prototypu od ustoupení od tohoto požadavku. Ale před nasazením k reálnému používání je dostatečný výkon aplikace nezbytný. Je potřeba optimalizovat především části kde komunikační část aplikace ukládá data do databáze. V současné době dochází k ukládání surových dat i jejich předzpracování pro uživatelské rozhraní okamžitě z komunikační části. Pro zrychlení je potřeba předzpracování dat oddělit a zefektivnit. Poté je potřeba výkonost otestovat, odhalit další slabá místa a tato pak optimalizovat.

Kapitola 14

Závěr

Účelem této práce bylo zanalyzovat možnosti, navrhnout a implementovat software pro správu prodejních zařízení. V práci bylo popsáno složení tohoto ovládacího softwaru, navržena architektura. Probrány byly možné problémy, které se mohly při realizaci projektu vyskytnout. Byly vybrány technologie, které budou pro tvorbu jednotlivých částí použity, a to včetně krátkého zdůvodnění výběru a popisu vlastností těchto technologií. Po výběru technologií byla popsána implementace včetně vlastností, které software má a funkčnosti, kterou software má. Dále bylo vysvětleno testování, rozčleněné na jednotlivé fáze. Téměř na konci bylo popsáno jak software zprovoznit. Poté následovalo zhodnocení projektu a návrhy na další rozšíření.

Literatura

[1] B., G. J. R.: *Foundations of Python Network Programming: The comprehensive guide to building network applications with Python.* Apress, 2010, ISBN 1430230037.

[2] Siemens: Firemní dokumentace.

[3] Stevens, R., W.: *Unix Network Programming, Volume 1: The Sockets Networking API.* Addison-Wesley Professional, 2003, ISBN 0131411551.

[4] WWW stránky: Apache. http://www.apache.org/, [cit. 2014-05-27].

[5] WWW stránky: Bootstrap. http://getbootstrap.com, [cit. 2014-05-27].

[6] WWW stránky: Django The Web framework for perfectionists with deadlines. http://www.djangoproject.com, [cit. 2014-05-27].

[7] WWW stránky: FastCGI. http://www.fastcgi.com/, [cit. 2014-05-27].

[8] WWW stránky: MariaDB. http://mariadb.org, [cit. 2014-05-27].

[9] WWW stránky: MySQL. http://www.mysql.com/, [cit. 2014-05-27].

[10] WWW stránky: nginx. http://nginx.org/, [cit. 2014-05-27].

[11] WWW stránky: The Open Source Initiative. http://opensource.org/, [cit. 2014-05-27].

[12] WWW stránky: Python. https://www.python.org/, [cit. 2014-05-27].

[13] WWW stránky: SB Admin. http://startbootstrap.com/sb-admin, [cit. 2014-05-27].

[14] WWW stránky: Slowloris HTTP DoS. http://ha.ckers.org/slowloris/, [cit. 2014-05-27].

[15] WWW stránky: Web Server Gateway Interface. http://wsgi.org/, [cit. 2014-05-27].

Příloha A

Obsah CD

`Data na CD se zdrojovými texty musí zůstat utajena!`

Na CD jsou zdrojové texty vytvářeného programu. V adresáři interface jsou soubory související s uživatelským rozhraním. V adresáři tcp jsou soubory používané v komunikační části.