

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

**Automatizovaný sklad
elektronických součástek**

Jan Priessnitz

Brno 2016

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor SOČ: 18. Informatika

Automatizovaný sklad elektronických součástek

Autor: **Jan Priessnitz**

Škola: **Gymnázium Brno, tř. Kpt. Jaroše,
příspěvková organizace**

Brno 2016

Prohlášení

Prohlašuji, že jsem svou práci vypracoval samostatně, použil jsem pouze podklady (literaturu, SW atd.) citované v práci a uvedené v přiloženém seznamu.

Dále prohlašuji, že tištěná i elektronická verze práce SOČ jsou shodné a nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, oprávech souvisejících s právem autorským a změně některých zákonů (autorský zákon) v platném znění.

V Brně dne: 14. dubna 2016

podpis:

Poděkování

Děkuji Jaroslavu Páralovi, mému školiteli, za obrovskou pomoc a mnoho užitečných rad a připomínek při práci na SOČ.

Dále děkuji Domu dětí a mládeže Helceletova za poskytnutí zázemí, především Jakubovi Streitovi za užitečné rady a připomínky.

Tato práce byla vypracována za finanční podpory JMK a JCMM.



Anotace

Cílem práce je vyvinout řídicí webovou aplikaci pro automatizovaný sklad elektronických součástek, který usnadňuje skladování a manipulaci s těmito součástkami. Sklad je určen na součástky v pouzdře pro povrchovou montáž (SMD), kde je manuální skladování z důvodu malých rozměrů součástek náročné a neefektivní. Výhody skladu spočívají v jednoduché údržbě, rychlém výdeji součástek a bezproblémovém využití skladu více uživateli. Webová aplikace slouží jako primární rozhraní pro ovládání skladu. Dalšími funkcemi jsou správa uživatelů, databáze a evidence součástek a zpracování dat do statistik.

Klíčová slova:

automatizace, skladování, SMD součástky, webová aplikace, evidence

Abstract

The aim of this work is to develop a web application for automated storage machine for electronic parts, which simplifies manipulation and storage of these parts. Storage machine is designed to store solid mounted devices (SMDs). These parts are very small and manual storage is difficult and inefficient. The advantages of the storage machine are easy maintenance, fast part retrieval and non-problematic usage of the storage machine by multiple users.

Keywords:

automation, storage, solid mounted devices, web application, records

Obsah

| | |
|--|-----------|
| 1 Motivace | 3 |
| 1.1 SMD součástky | 3 |
| 1.2 Současná situace | 3 |
| 1.3 Řešení | 4 |
| 2 Sklad elektronických součástek | 5 |
| 2.1 Můj podíl | 6 |
| 3 Návrh aplikace | 6 |
| 3.1 Požadavky | 6 |
| 3.2 Možné přístupy | 7 |
| 3.3 Použitý hardware a software | 7 |
| 3.4 Struktura aplikace | 8 |
| 3.5 Architektura MVC | 8 |
| 4 Daemon | 9 |
| 4.1 Komunikace se skladem | 9 |
| 4.1.1 Paket | 10 |
| 5 Databáze | 11 |
| 6 Webové rozhraní | 11 |
| 6.1 Nette Framework | 11 |
| 6.2 Autentifikace a autorizace | 12 |
| 6.3 Uživatelské rozhraní | 12 |
| 6.3.1 AJAX | 12 |
| 6.3.2 Responzivní design | 13 |
| 6.4 Funkce | 13 |
| 6.4.1 Vybírání součástek | 13 |
| 6.4.2 Doplňování součástek | 14 |
| 6.4.3 Databáze součástek | 14 |
| 6.5 Integrace API obchodů s elektronickými součástkami | 14 |
| 6.6 Statistika | 14 |
| 6.7 Optimalizace výkonu | 14 |

Úvod

Předmětem práce je řídící aplikace pro automatizovaný sklad SMD součástek. Narozdíl od ručního skladování SMD součástek je práce se skladem i jeho údržba jednoduchá pro uživatele a časově nenáročná. Mezi hlavní vlastnosti a funkce skladu s řídící aplikací patří:

- Přístup ke skladu pomocí intuitivního webového rozhraní.
- Jednoduchý a rychlý výběr součástek s více možnostmi navolení těchto součástek (z tabulek podle druhu a vlastnosti nebo z partlist souboru¹)
- Databáze typů součástek, která zjednoduší doplňování a další nakupování součástek.

1 Motivace

Elektronika je dnes všude kolem nás a málokdo by si dnes bez ní dokázal svět představit. Jen v České republice vyvíjí elektroniku tisíce lidí (ať už v práci, nebo ve volném čase). Nedílnou součástí drtivé většiny elektroniky jsou desky plošných spojů. Ty jsou nejčastěji osazovány speciálními součástkami určenými přímo pro ně. Tyto součástky se nazývají SMD součástky (solid mounted devices - součástky v pouzdře pro povrchovou montáž). Ve výrobě je osazování desek a skladování SMD součástek plně automatizováno, ale ve vývoji desek plošných spojů se často desky osazují ručně a vzniká problém skladování SMD součástek.

1.1 SMD součástky

SMD součástky jsou speciálním typem součástek. Jsou určené k použití jen při výrobě desek plošných spojů a proto jsou velmi malé (mají rozměry řádově v milimetrech) a prodávají se ve velkých množstvích zalepené v dlouhých papírových páscích (zhruba 1 cm širokých). Konvenční skladování těchto součástek (např. ručně v šuplících) je proto velmi náročné. Druhý součástek je mnoho a často nejsou nijak označené, takže je složité a pomalé vybírat nebo doplňovat součástky a zároveň udržovat ve skladu pořádek, obzvláště pokud sklad využívá více lidí. Také je těžké udržet přehled nad počty součástek ve skladu a jednoduše se může stát, že nějaký typ součástky neočekávaně dojde a je nutné typ doobjednat. To všechno vývoj zpomaluje a tím pádem i prodražuje. Tento problém jsme řešili i na Robotárně².

1.2 Současná situace

Dnes bohužel žádné řešení na tento problém neexistuje. Velké vývojové firmy mají vlastní skladovny, které ale nejsou plně automatizované. Asi nejpodobnější zařízení skladu SMD součástek je osazovací zařízení, které automaticky osazuje desky plošných spojů SMD součástkami. To má ale často kapacitu jen na pár desítek kusů součástek a jeho cena se pohybuje řádově ve statisících korun.

¹Soubor partlist obsahuje seznam součástek např. pro konkrétní projekt. Generují jej programy pro návrh elektroniky (např. Eagle)

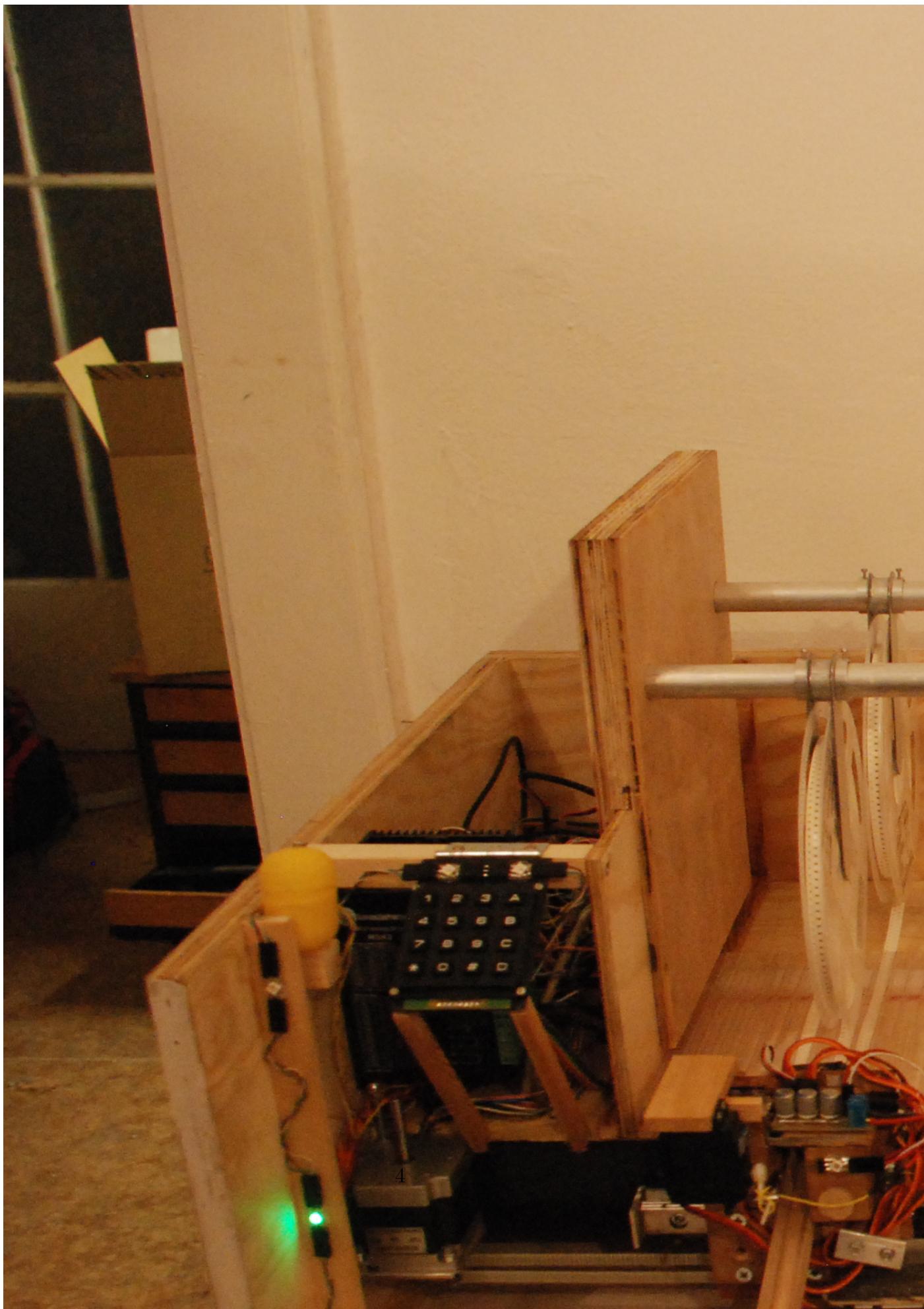
²Dům dětí a mládeže Helceletova

1.3 Řešení

S kolegy jsme se proto rozhodli vyvinout automatizovaný sklad SMD součástek, který má tyto problémy vyřešit. Součástky jsou skladovány na jednom místě na páscích namotaných na kotoučích. Sklad tyto součástky automaticky vydává na povol uživatele přes webové uživatelské rozhraní. Tímto se rapidně sníží potřebný čas k vybrání součástky ve skladu, jelikož člověk již nemusí hledat pásek se součástkou, vysunout jej a ustříhnout příslušný počet součástek. Aplikace bežící ve skladu navíc bude uchovávat informace o počtech kusů součástek a historii výběrů. Správce skladu pak může z aktuálních stavů součástek zjistit, zda je potřeba dokoupit nějaký druh součástky, případně ze statistik o výběrech součástek naplánovat další nákup. Díky hromadnému nákupu součástek tak může výrazně ušetřit. Sklad může být využíván také v obchodech s elektronikou, které zažívají stejné problémy, jako vývojová pracoviště.

Celý projekt jsme rozdělili na 3 části. Mojí prací na tomto projektu je naprogramovat řídicí aplikaci s uživatelským rozhraním. Návrhem elektroniky a mechaniky skladu se zabývají dva mí kolegové.

2 Sklad elektronických součástek



Sklad je zhruba 1,5 m dlouhý a 0,5 m široký. Součástky jsou skladovány v zadní části bud' volně (krátké ústřížky), nebo na kotoučích. V přední části se na kolejnicích pohybuje stříhací hlava, která pásky vytahuje a následně stříhá. V levé části skladu je umístěn elektrický zdroj, servomotory s drivery³, elektroniku a microPC Raspberry Pi, na kterém běží řídící aplikace. Sklad lze rozšiřovat o další patra, takže kapacita skladu je variabilní. V každém patře je 70 příhrádek, takže není problém vyrobit sklad se stovkami příhrádek.

2.1 Můj podíl

Práce mých kolegů končí u návrhu a naprogramování vnitřní elektroniky skladu. Mým podílem je řídící aplikace, která na jedné straně přijímá příkazy od uživatele a na straně druhé posílá data po sériové lince do vnitřní elektroniky. Veškerá má práce se proto odehrává na Raspberry Pi.

3 Návrh aplikace

3.1 Požadavky

Nejdříve jsem si definoval požadavky na funkce a jiné vlastnosti aplikace, abych později mohl zvolit vyhovující hardware a software:

1. **uživatelské rozhraní** - Sebelepší aplikace se neobejde bez uživatelů. Proto je jednou z hlavních uživatelská přívětivost. Aplikace by měla být přehledná a její používání rychlé a intuitivní.
2. **mnoho funkcí** - Jádro aplikace umožní uživateli vybrat součástku ze skladu. Aplikace by ale mohla uživateli dál usnadnit používání skladu mnoha přidanými funkcemi (např. databáze součástek, integrace API⁴ obchodů s elektronikou).
3. **rozumná rychlosť** - Přestože je rychlosť celého skladu limitována hlavně rychlostí stříhací hlavy, měla by aplikace stíhat zpracovávat přístupy vícero uživatelů v rozumném čase, aby nezpomalovala práci uživatelů.
4. **nízké nároky** - Hardware, na kterém aplikace běží, by měl spotřebovávat co nejméně elektřiny a neměl by zabírat moc místa. Také by měl být levný, aby se jeho cena minimálně promítla do celkové ceny skladu.
5. **dostupnost** - Uživatel by neměl být omezován ve výběru zařízení, se kterým bude sklad ovládat. Aplikace by měla být dostupná na všechny typy zařízení.

³Servomotor je typ nízkootáčkového motoru. Servomotor je nutné řídit speciální elektronikou (drivery).

⁴Application programming interface - rozhraní, pomocí kterého mohou cizí aplikace jednoduše přistupovat k funkcím obchodu

3.2 Možné přístupy

Po definování požadavků na aplikaci úřipadalo v úvahu několik možností, jak by aplikace vypadala:

1. **Server + klientská aplikace** - Na skladu běží jen serverová část aplikace, která nízkoúrovňově komunikuje s klientskou aplikací. Tento přístup nemá velké nároky na sklad a je rychlý. Problém nastává u přenositelnosti aplikace. Pro každé zařízení by bylo potřeba psát novou klientskou aplikaci. Obecně je také složitější navrhnut kvalitní uživatelské prostředí pro nativní aplikace⁵.
2. **Modul pro eshop systém** - Další možností bylo využít jako frontend/footnote Část aplikace, která komunikuje s uživatelem (uživatelské rozhraní). Protikladem je backend, který obvykle provádí vnitřní procesy aplikace. již existující eshopový systém a rozšířit jej o modul zajišťující komunikaci se skladem. Výhodou je zdánlivě velmi podobná architektura, kdy si uživatel vybírá součástky a poté si je koupí (vybere ze skladu). Při bližším pohledu jsou bohužel vidět rozdíly: eshopové systémy kladou velký důraz na způsob dopravy, platbu, ceny, aj.
3. **Webová aplikace** - Nakonec jsem se rozhodl jít cestou webové aplikace. Hlavní výhoda spočívá v dostupnosti ze všech zařízení, ale i v přívětivosti uživatelského prostředí. Přestože náročnost je o trochu větší, než u prvního přístupu, stále se jedná o rychlé a relativně nenáročné řešení.

3.3 Použitý hardware a software

Jako zařízení, na kterém aplikace poběží, jsem zvolil microPC Raspberry Pi. Výkon Raspberry Pi je pro webovou aplikaci dostatečný a spotřeba nepřekračuje 5 W, takže je energeticky nenáročné. Výhodou je také cena, která se pohybuje okolo 1000 Kč. Pro Raspberry Pi existuje linuxová distribuce⁶ Raspbian, která poskytuje široký výběr softwaru.

Místo abych některé součásti aplikace psal sám a znova vynalézal kolo, snažil jsem se využít již dostupný software. Některé volby konkrétního softwaru mají svá opodstatnění. Jindy jsem si vybíral jednoduše podle toho, jestli již mám s nějakým softwarem zkušenosti:

1. **Apache web server** - standardní linuxový web server
2. **jazyk PHP** - asi nejpoužívanější jazyk pro psaní webových aplikací. Pro jazyk PHP existuje mnoho knihoven a frameworků⁷.
3. **Nette Framework** - jednoduchý, ale kvalitní framework pro jazyk PHP
4. **Bootstrap** - knihovna CSS s tlpkou
5. **Doctrine 2** - PHP knihovna pro definování databázové struktury

⁵Aplikace běžící přímo na zařízení. Protikladem mohou být například webové aplikace

⁶linuxová distribuce - operační systém postavený na jádře Linux

⁷framework - nástavba na určitý jazyk, která usnadňuje vývoj aplikací

6. **MySQL** - databázový server
7. **jazyk Python** - skriptovací jazyk. Užitečný při tvorbě výkonově nenáročných aplikací

3.4 Struktura aplikace

tady bude obrázek

Aplikaci jsem rozdělil na několik částí, které budou vykonávat různé funkce a budou vzájemně komunikovat:

1. **daemon⁸** - čte příkazy z databáze a posílá je po sériové lince do skladu. Zpracovává chyby, které mu sklad nahlásí a provádí automatické funkce aplikace (např. posílání upozornění emailem).
2. **databáze** - zprostředkovává komunikaci mezi daemonem a webovým rozhraním. Také je to jediné místo, kam se ukládají data aplikace.
3. **webové rozhraní** - poskytuje uživatelské rozhraní a komunikuje s uživatelem. Uživatelské příkazy zapisuje do databáze.

3.5 Architektura MVC

Architektura MVC (Model-View-Controller) je návrh aplikace, který se ji snaží rozdělit na více částí, které spolu co nejméně souvisí. V ideálním případě to znamená, že budu moci jednu z částí nahradit, aniž bych musel upravovat ostatní části. MVC konkrétně rozděluje aplikaci na 3 části:

1. **Model** - část aplikace, která se stará o ukládání dat. Ostatní části aplikace přes ni přistupují k informacím.
2. **View** - stará se o prezentování dat získaných od modelu uživateli a zprostředkování komunikace mezi uživatelem a Controllerem.
3. **Controller** - vykonává všechny logické operace a příkazy aplikace, které jsou vyvolány přes view a upravují data přes model.

Výhodou webové aplikace je to, že architekturu MVC respektuje a jednotlivé komponenty jsou striktně oddělené. Kdyby uživatel z nějakého důvodu potřeboval, aby části aplikace běžely na různých strojích, není to problém. Jediná část, která musí na skladu běžet, je daemon. Ostatní části aplikace pak mohou běžet vzdáleně a komunikace může probíhat po síti.

⁸Daemon je označení pro aplikaci, která běží na pozadí a neustále

4 Daemon

První komponentou celé aplikace je daemon. Je to program, který se automaticky spustí po zapnutí skladu⁹ a běží nepřetržitě na pozadí. Výhodou oddělení daemonu od webové aplikace je zejména modularita. Pokud se změní elektronika nebo fyzická stavba skladu, stačí upravit daemon a webovou aplikaci není třeba měnit. Daemon je napsán v Pythonu, což je skriptovací jazyk, který umožňuje jednoduchý přístup jak k databázi, tak k sériové lince. Funkce daemonu jsou:

1. **předávání příkazů z databáze do skladu** - Hlavní funkcí daemonu je čtení příkazů z databáze a následné poslání příkazu skladu. Sklad se po celou dobu dívá do databáze, zda není potřeba něco vykonat, a pokud ano, zpracuje příkaz a po sériové lince pošle skladu sekvenci paketů.
2. **hlášení chyb ze skladu do databáze** - V případě, že při vydávání součástky nastane problém. Sklad pošle po sériové lince zpět do daemonu sérii paketů o chybě a daemon chybu zapíše do databáze. Webová aplikace poté chybu zobrazí uživateli.
3. **automatické oznámení emailem** - Nastanou-li události vyžadující pozornost správců skladu, pošle jim daemon upozorňující email. Tyto události zahrnují nedostatek součástek nebo chyba při vydávání.
4. **ovládání LED indikátorů** - Ve skladu jsou zabudované LED diody signalizující stav skladu (napájení, síť, chyby). Tyto LED diody jsou napojené na Raspberry Pi a jsou ovládané daemonem.

Daemon jsem psal jako objektově orientovanou aplikaci. To znamená, že je rozdělen do několika logických celků (tříd), které obstarávají jednotlivé funkce a v případě potřeby nových funkcí není potřeba přepisovat celou aplikaci, ale jen třídy, kterých se změna týká. Daemon obsahuje následující třídy:

- **Daemon** - hlavní třída komponenty. Třída Daemon sdružuje všechny ostatní třídy a definuje funkci loop(), která se neustále znovu spouští, a ve které probíhají všechny funkce daemonu.
- **Database** - třída pro přístup k databáze. Ostatní třídy daemonu pomocí ní získávají nové příkazy a další informace.
- **Device** - třída pro přístup ke skladu. Jako jediná komunikuje přímo se skladem (má přístup k sériové lince) a je v ní definován komunikační protokol. Pomocí funkcí run(), move(), pull() a cut() je možné sklad ovládat.
- **PowerIndication** - třída pro ovládání LED diod.

4.1 Komunikace se skladem

Daemon komunikuje se skladem přes sériovou linku. Ta neposkytuje žádný abstraktní způsob komunikace, ale jen proud bytů. Daemon čte ze vstupního streamu¹⁰ data ze skladu a zapisuje do výstupního streamu příkazy.

⁹Daemon se spouští přes soubor rc.local

¹⁰stream - speciální typ souboru, který imituje datový proud

4.1.1 Paket

Vzhedem k povaze komunikace (povely) není možné, aby se data posílala jen jako proud bytů bez další abstrakce. Proto jsme s kolegou, který se zabývá elektronikou, museli navrhnout komunikační protokol, který bude tvořen pakety.

| | |
|--------------|-------------------------------|
| 1. byte | start byte |
| 2. byte | číslo paketu |
| 3. byte | číslo příkazu |
| 4. byte | délka argumentů / 1. argument |
| 5. - ?. byte | další argumenty |

Tabulka 1: Popis paketu

Každý paket je tvořen byty. Maximální délka každého paketu je 64 bytů. Přestože komunikace probíhá po sériové lince, která je krátká a neměly by nastávat žádné chyby, jsou v paketu zahrnuty bezpečnostní prvky, které by případné chyby měly eliminovat. Paket vždy začíná bytem 0x80 (start byte), který by měl zajišťovat to, že v případě neočekávaného vložení nebo odstranění bytu z datového proudu žádná strana nebude číst stream 'posunutě' (tím by se zásadně změnil význam paketu). Hodnota 0x80 je zvolena kvůli svému zápisu v binární soustavě (01000000), který je speciální a je málo pravděpodobné, že by taková posloupnost bitů vznikla chybou (např. šumem). 2. bytem je číslo paketu, které je zde také kvůli bezpečnosti. Pokud totiž jedna strana pošle paket, na který má druhá strana odpovědět dalším paketem, je potřeba, aby první strana věděla, jestli, a na jaký paket jí přichází odpověď. Jelikož se do 1 bytu vejde jen hodnota do 255, bude toto číslo přetékat. To by ale neměl být problém (pravděpodobnost, že by jeden paket 'předběhl' 255 dalších je zanedbatelná). 3. bytem je číslo příkazu (viz tabulku číslo 2). Dalšími byty jsou argumenty příkazu. Pokud je z čísla příkazu možné jednoznačně určit počet argumentů, hned 4. bytem začínají argumenty. Pokud ne, ve 4. bytu je délka argumentů, které začínají až 5. bytem. Dalším bezpečnostním prvkem jsou také příkazy ACK a NACK, které jsou poslány jako odpověď na každý příkaz.

| Číslo příkazu | Název | Popis |
|---------------|-------|--|
| 0x00 | NACK | not acknowledged - odpověď na příkaz, který nebyl úspěšně proveden; následuje kód chyby |
| 0x01 | ACK | acknowledged - odpověď na úspěšně vykonaný příkaz, podle toho, na jaký příkaz odpovídá, může mít další argumenty |
| 0x02 | move | posun na dané místo, následují 2 byty s pozicí (Uint16) |
| 0x03 | pull | vytáhnutí kusu pásku z příhrádky, následují 2 byty s počtem dírek, které se mají vytáhnout (Uint16) |
| 0x04 | cut | ustříhnutí vytáhnutého kusu pásky, nemá argumenty |
| 0x05 | init | inicializaci - zařízení po startu dojede s hlavou na oba konce, aby zkalirovalo motory a zjistilo přesnou polohu, nemá argumenty |
| 0x06 | power | napájení krokových motorů - příkaz je potřeba provést ještě před inicializací, nemá argumenty |

Tabulka 2: Přehled příkazů

5 Databáze

Další komponentou aplikace je databáze. Ta zajišťuje ukládání veškerých dat a tím pádem i komunikaci mezi webovou aplikací a daemonem. Databáze běží na databázovém serveru MySQL a ostatní komponenty k databázi přistupují pomocí unix socketu¹¹, ale je zde i možnost vzdáleného připojení po síti, což umožňuje spustit jednotlivé komponenty na více strojích (viz kap. 3.5).

Data se v databázi ukládají formou objektů, které jsou mezi sebou databázovými relacemi provázané. V databázi budou uloženy následující objekty:

- **User** - objekt uživatele. Obsahuje informace o uživateli, jeho přístupové údaje a uživatelská práva
- **Part** - objekt součástky. Obsahuje základní informace o jednom typu součástky.
- **Attribute** - atribut součástky. Definuje nějakou fyzikální vlastnost (např. odpor), kterou může součástka mít.
- **PartAttribute** - konkrétní hodnota vlastnosti nějaké součástky
- **Type** - Definuje druh součástky (např. rezistor, dioda).
- **Socket** - objekt přihrádky. Obsahuje informace o konkrétní přihrádce ve skladu. Tzn. její polohu, odkaz na součástku v ní a množství součástek.
- **QueueEntry** - položka ve frontě. Popisuje jeden příkaz na výdej součástky. Do tabulky těchto objektů se dívá i daemon.
- **HistoryEntry** - záznam o výběru. Obsahuje informace o již proběhlém příkazu. Po úspěšném zpracování položky ve frontě se vytvoří záznam o výběru a uloží se do databáze.

6 Webové rozhraní

Třetí a největší komponenta celé aplikace je webové rozhraní. Jak již jsem dříve zmínil, je webové rohnaní napsáno v PHP s využitím Nette Frameworku a běží na Apache Web Server. Jelikož má celá aplikace za účel hlavně usnadnit uživateli používání skladu, zaměřil jsem se na tuto komponentu ze všech nejvíce, jelikož je zároveň uživatelským rohnaním celé aplikace.

6.1 Nette Framework

Nette je český framework pro psaní webových aplikací v PHP. Jeho hlavní prioritou je jednoduchost, což narozdíl od konkurenčních frameworků, které se zaměřují spíše na co nejvíce funkcí, splňuje. Zároveň však nepostrádá žádnou funkci, kterou bych při vývoji webového rozhraní potřeboval. Prvním krokem k vytvoření funkční webové aplikace je založení projektu pomocí Composeru, který zároveň vyřeší závislosti na dalším softwaru.

¹¹způsob komunikace mezi aplikacemi na jednom zařízení

Vytvoří se adresářová struktura a poté již stačí psát aplikaci pomocí nově definovaných tříd. Každá aplikace v Nette by měla dodržovat rozdelení komponent na Model-View-Controller/footnoteMVC - architektura při vytváření aplikací s uživatelským rozhraním. Viz kap. 3.5. Aplikace se primárně dělí na presentery, které načítají data z databáze pomocí modelu a připraví je tak, aby je na ně napojené šablony mohly jednoduše zobrazit. Uživatel tedy vidí šablonu, která je naplněna daty z presenteru.

6.2 Autentifikace a autorizace

Kvůli bezpečnosti jsem přístup do aplikace omezil uživatelskými účty. Člověk bez uživatelského účtu se do aplikace vůbec nedostane. Dále jsou definovány 3 úrovně uživatelských práv, aby se rozlišilo mezi lidmi, kteří sklad užívají a těmi, kteří se o něj starají a potřebují mít zpřístupněno více funkcí. Kromě lepší bezpečnosti (obyčejný uživatel bude moci napáchat mnohem méně škod) to také zpřehlední uživatelské rozhraní, protože uživatel nebude zbytečně zatěžován mnoha tlačítka, která nevyužije.

Webová aplikace tedy bude ozlišovat mezi těmito uživatelskými rolemi:

1. **uživatel** - základní uživatelský účet. Má možnost vybírat součástky a prohlížet statistiku a databázi součástek.
2. **správce** - správcovský účet. Má možnost doplňovat součástky, manipulovat s rozložením součástek v příhrádek a měnit databázi součástek. Na email mu budou chodit vybraná upozornění.
3. **administrátor** - nejmocnější účet. Může dělat všechno, co správce i uživatel, ale navíc bude moci přidávat, upravovat i mazat uživatelské účty.

6.3 Uživatelské rozhraní

Důležitou součástí vývoje aplikace byl návrh uživatelského rozhraní. Snažil jsem se uživatelské rozhraní udělat co nejjednodušší a nejpřehlednější (někdy i na úkor estetiky). Také jsem se snažil propojit části webové aplikace tak, aby uživatel nemusel zbytečně moc klikat. Při návrhu jsem použil CSS knihovnu Bootstrap, se kterou se dá relativně jednoduše vytvořit čistý a přehledný design webu. Hlavním navigačním prvkem webu je horní lišta, která obsahuje odkazy na všechny části webu. Zbytek prostoru je vyplněn vždy danou částí webu. K vytváření seznamů jsem použil rozšíření pro Nette Framework Mesour Datagrid, které umožňuje vytvářet chytré seznamy s tříděním a filtrováním. K reprezentování dat různými grafy jsem použil knihovnu D3.js.

6.3.1 AJAX

AJAX¹² je koncept, který říká, že po provedení nějaké uživatelské akce je někdy lepší nenačítat celou stránku znova, ale jen ji aktualizovat. V praxi to znamená, že pokud uživatel například změní rozložení součástek ve skladu, není potřeba celou stránku načítat znova, ale stačí jen vykreslit upravenou část znova. Technicky je to řešeno tak, že na straně uživatele (v prohlížeči) běží Javascript, který posílá HTTP požadavky na server.

¹²Asynchronous Javascript and XML - název neodpovídá konceptu

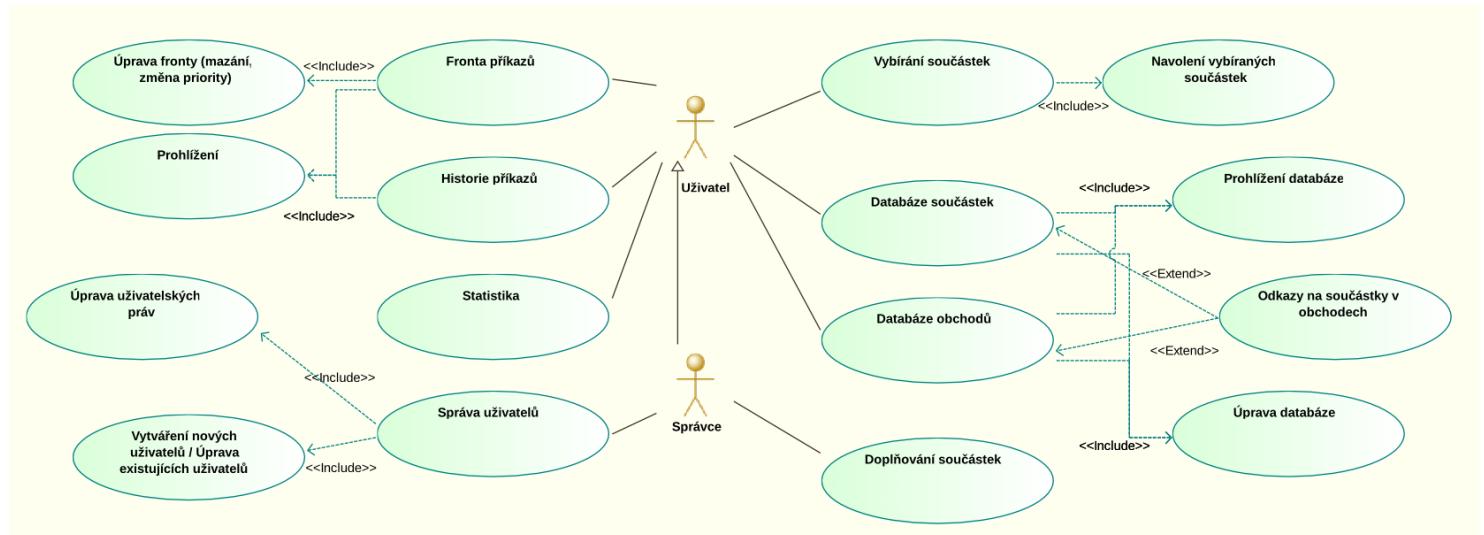
Server podle požadavku může například něco změnit v databázi a pošle zpět HTML kód aktualizované části stránky. Na straně uživatele pak Javascript nový HTML kód vykreslí.

Snažil jsem se ve webové aplikaci AJAX používat co nejvíce, protože si myslím, že neustále znovunačítání stránek je pomalé a hlavně nepříjemné.

6.3.2 Responzivní design

Největší výhoda knihovny Bootstrap je že za vývojáře velmi jednoduše vyřeší responzivní design. Není tedy potřeba psát aplikaci zvlášť pro zařízení s nízkým rozlišením (mobilní telefony), ale vzhled se automaticky přizpůsobí. Proto je celá webová aplikace přizpůsobena jak pro počítače a tablety, tak pro mobilní telefony a uživatelé se ke skladu dostanou opravdu odkudkoli.

6.4 Funkce



Obr. 4: Use case diagram aplikace

6.4.1 Vybírání součástek

Vybírání součástek je hlavní funkce celé aplikace. Proto jsem se snažil, aby měl uživatel několik možností, jak si součástky navolit:

- podle typu** - Z jednoduché tabulky všech součástek si uživatel vybere, co potřebuje. Bude mít možnost tabulku filtrovat podle typu součástky (rezistor, kondenzátor, aj.).
- podle vlastnosti** - Uživatel si nejdříve vybere, který druh součástky potřebuje. Podle vybraného druhu se mu zobrazí složitá tabulka se součástkami daného druhu a jejich vlastnostmi. Tuto možnost využijí hlavně lidé, kteří potřebují součástku o daných parametrech, ale nevědí, zda ve skladu nějaká vyhovující je
- ze souboru** - Uživatel jednoduše nahraje partlist soubor (viz úvod) projektu, na kterém pracuje. Vůbec tedy není potřeba vybírat součástky z tabulek.

6.4.2 Doplňování součástek

V případě, že nějaká součástka dochází, stačí, aby správce skladu vyměnil kotouč a na webu zadal nové množství součástek v přihrádce. Není třeba jakkoli manipulovat s rozložením součástek ve skladu.

6.4.3 Databáze součástek

Aby mohli uživatelé vyhledávat mezi součástkami na skladu, je potřeba, aby byly o každé součástce na skladu informace v databázi součástek. Přidávání součástky do skladu tedy může být velmi pracné, ale opak je pravdou. Díky integraci API obchodů s elektronikou stačí vyfotit čárový kód na kotouči se součástkami a web se o zbytek postará. V případě, že již součástka v databázi existuje, přiřadí ji do přihrádky. V případě, že součástka ještě v databázi není, automaticky ji i se všemi vlastnostmi přidá. Díky standardu HTML 5 mohou nově webové aplikace používat fotoaparát v mobilu. Správce skladu tedy jen vyfotí čárový kód, zadá číslo přihrádky, do které chce součástku přidat, a je hotovo.

6.5 Integrace API obchodů s elektronickými součástkami

Kromě obrovského zjednodušení přidávání součástek má integrace API obchodů s elektronickými součástkami také další výhodu: zjednodušení nákupu součástek. V případě, že chce správce dokoupit docházející součástky, nemusí ručně procházet všechny obchody a hledat nejlepší cenu. Web správce rovnou přesměruje do obchodu s nejlepší cenou. V extrémním případě by správci došel email s objednávkou docházejících součástek v nejlepším obchodě.

6.6 Statistika

Web umí vykreslovat mnoho různých grafů a generovat výtahy z dat o výběrech součástek. Správce skladu může tyto statistiky využít například při plánování nákupu součástek a předpovědět, kdy bude potřeba součástky dokoupit. Pokud bude například kupovat více součástek najedou v jedné objednávce, může výrazně ušetřit za dopravu.

6.7 Optimalizace výkonu

Při testování aplikace na Raspberry Pi se bohužel objevily problémy s rychlostí aplikace. Načítání stránky trvalo i 5 sekund a celkově byla aplikace pomalá. Zjistil jsem, že problém není v rychlosti procesoru nebo nedostatku operační paměti, ale v rychlosti SD karty. Nette Framework využívá pro vylepšení výkonu cachování¹³, takže si ukládá velké množství souborů krátkodobě na SD kartu. Na běžném serveru se standardním hard diskem by to nebyl problém, ale na Raspberry Pi s pomalou SD kartou to výrazně zpomaluje celou aplikaci. Proto jsem na Raspberry Pi nainstaloval PHP OPCache, která způsobuje, že se ke cachování místo SD karty využívá operační paměť, která je mnohonásobně rychlejší. Problém se tímto vyřešil a aplikace ted' běží dostatečně rychle.

¹³cachování - ukládání výpočtů na disk a pozdější opětovné využití za účelem snížení nároků na výpočetní výkon

Závěr

Podle požadovaných funkcí a parametrů jsem navrhl strukturu aplikace pro automatizovaný sklad součástek, kterou jsem následně vyvinul a nasadil na reálné zařízení. Aplikace zjednodušuje přístup k zařízení a jeho obsluhu a rozšiřuje funkce skladu o evidenci součástek ve skladu, nástroje pro správu skladu a nákup součástek a upozorňování uživatelů na dané události.

Do budoucna s kolegy počítáme s rozšířením skladu na více pater a zavedení přihrádek o různých šírkách. Aplikace je na toto připravená, ale bude potřeba ji o tyto možnosti rozšířit. Kromě toho bych chtěl aplikaci rozšířit o další funkce, které nesouvisí s mechanikou skladu. Především bych chtěl přidat funkci třídění součástek na webu podle jejich vlastností, vybírání součástek stylem nakupování v eshopech, vybrání všech součástek na daný projekt podle souboru se seznamem součástek, ale také integraci API velkých obchodů s elektronikou do aplikace. Také chceme přizpůsobit sklad pro využití v obchodech s elektronikou. Při testování aplikace jsem také narazil na některé nevýhody stávající architektury. Proto chci znova navrhnut architekturu některých částí aplikace a tyto části podle nové architektury znova implementovat. Také bych rád zpřehlednil kód aplikace (přidal anotace funkcí a proměnných, sjednotil styl kódování).

Seznam použité literatury a softwaru

- Oficiální webové stránky projektu Raspberry Pi - <http://www.raspberrypi.org/>
- Oficiální webové stránky jazyka PHP - <http://php.net/>
- Oficiální webové stránky jazyka Python - <https://www.python.org/>
- Oficiální webové stránky frameworku Nette - <http://nette.org/cs/>
- Oficiální webové stránky databázového serveru MySQL - <http://www.mysql.com/>
- Oficiální webové stránky knihovny Bootstrap - <http://getbootstrap.com/>
- Oficiální webové stránky modelovacího programu Modelio - <https://www.modelio.org/>