

**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

<b>1 Motivace</b>	<b>2</b>
1.1 SMD součástky . . . . .	2
1.2 Současná situace . . . . .	2
1.3 Řešení . . . . .	3
<b>2 Sklad elektronických součástek</b>	<b>4</b>
2.1 Můj podíl . . . . .	5
<b>3 Návrh aplikace</b>	<b>5</b>
3.1 Požadavky . . . . .	5
3.2 Možné přístupy . . . . .	6
3.3 Použitý hardware a software . . . . .	6
3.4 Struktura aplikace . . . . .	7
3.5 Architektura MVC . . . . .	7
<b>4 Daemon</b>	<b>8</b>
4.1 Komunikace se skladem . . . . .	8
4.1.1 Paket . . . . .	9
<b>5 Databáze</b>	<b>9</b>
<b>6 Webové rozhraní</b>	<b>10</b>
6.1 Funkce . . . . .	10
6.1.1 Změna stavů součástek v přihrádkách (Doplňování součástek) . . . . .	10
6.1.2 Správa přihrádek . . . . .	11
6.1.3 Databáze součástek . . . . .	11
6.1.4 Databáze obchodů . . . . .	11
6.1.5 Správa uživatelů . . . . .	11
6.1.6 Fronta příkazů . . . . .	11
6.1.7 Historie příkazů . . . . .	11
6.2 Nette framework . . . . .	11
6.3 Autentifikace a autorizace . . . . .	12
6.4 Uživatelské prostředí . . . . .	13
6.4.1 Hlavní komponenty prostředí . . . . .	13
6.4.2 Funkce prostředí . . . . .	14
<b>7 Popis daemonu</b>	<b>17</b>
7.1 Autostart . . . . .	17
7.2 Struktura daemonu . . . . .	18
7.3 Posílání emailových upozornění . . . . .	18
7.4 Popis komunikačního protokolu . . . . .	18
7.4.1 Popis paketu . . . . .	18

# 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 souboru<sup>1</sup>)
- 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 Robotárně<sup>2</sup>.

### 1.2 Současná situace

Dnes bohužel žádné řešení na tento problém neexistuje. Velké vývojové firmy mají vlastní sklady, 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.

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

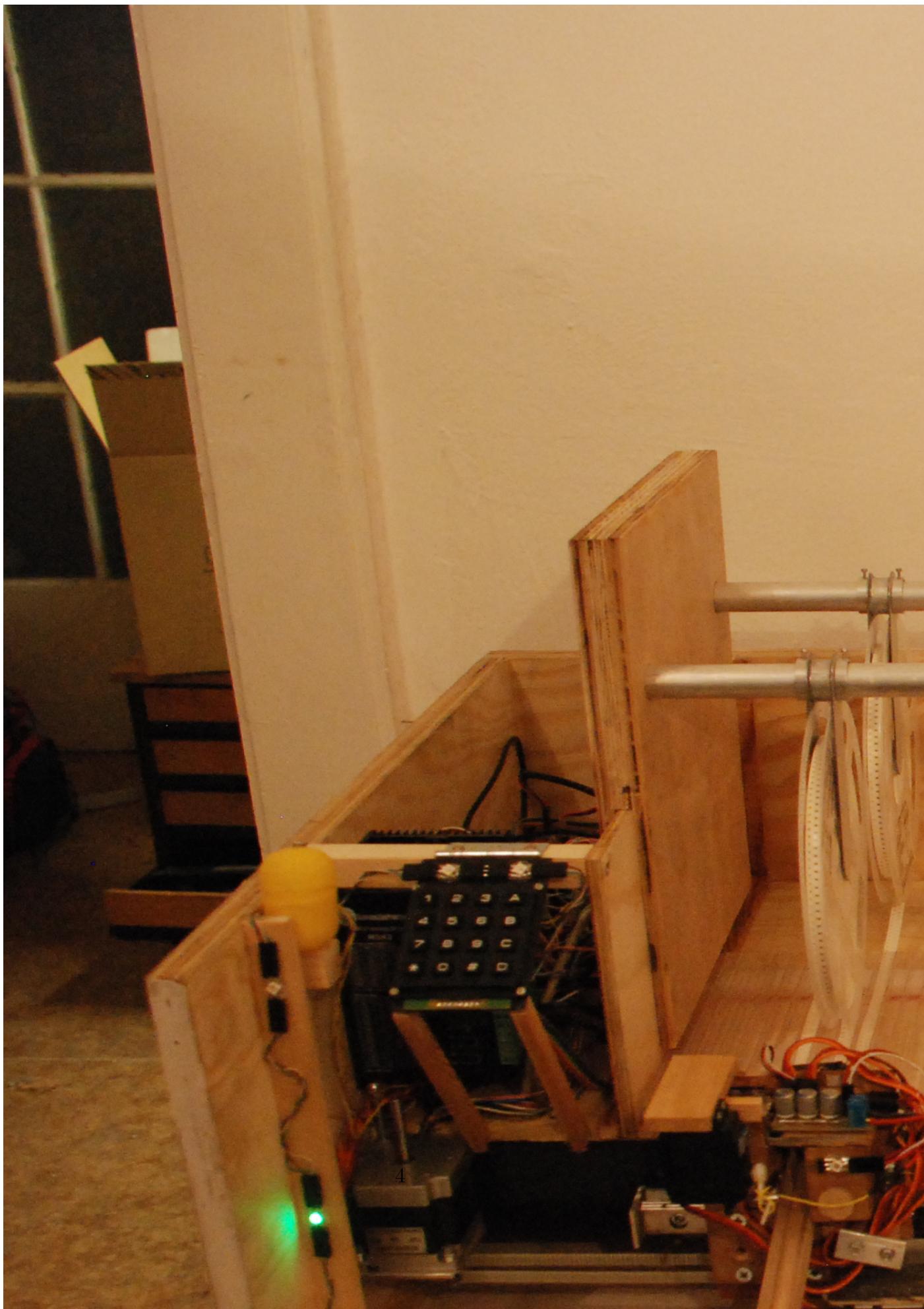
<sup>2</sup>Dům dětí a mládeže Helceletova

### **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 drivery<sup>3</sup>, 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 API<sup>4</sup> 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.

---

<sup>3</sup>Servomotor je typ nízkootáčkového motoru. Servomotor je nutné řídit speciální elektronikou (drivery).

<sup>4</sup>Application 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 aplikace<sup>5</sup>.
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 distribuce<sup>6</sup> 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 frameworků<sup>7</sup>.
3. **Nette Framework** - jednoduchý, ale kvalitní framework pro jazyk PHP
4. **Bootstrap** - knihovna CSS s tl. pl.
5. **Doctrine 2** - PHP knihovna pro definování databázové struktury

---

<sup>5</sup> Aplikace běžící přímo na zařízení. Protikladem mohou být například webové aplikace

<sup>6</sup> linuxová distribuce - operační systém postavený na jádře Linux

<sup>7</sup> framework - 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. **daemon<sup>8</sup>** - 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.

---

<sup>8</sup>Daemon 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 skladu<sup>9</sup> 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 streamu<sup>10</sup> data ze skladu a zapisuje do výstupního streamu příkazy.

<sup>9</sup>Daemon se spouští přes soubor rc.local

<sup>10</sup>stream - 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 'posunuté' (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.

## 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 socketu<sup>11</sup>, 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:

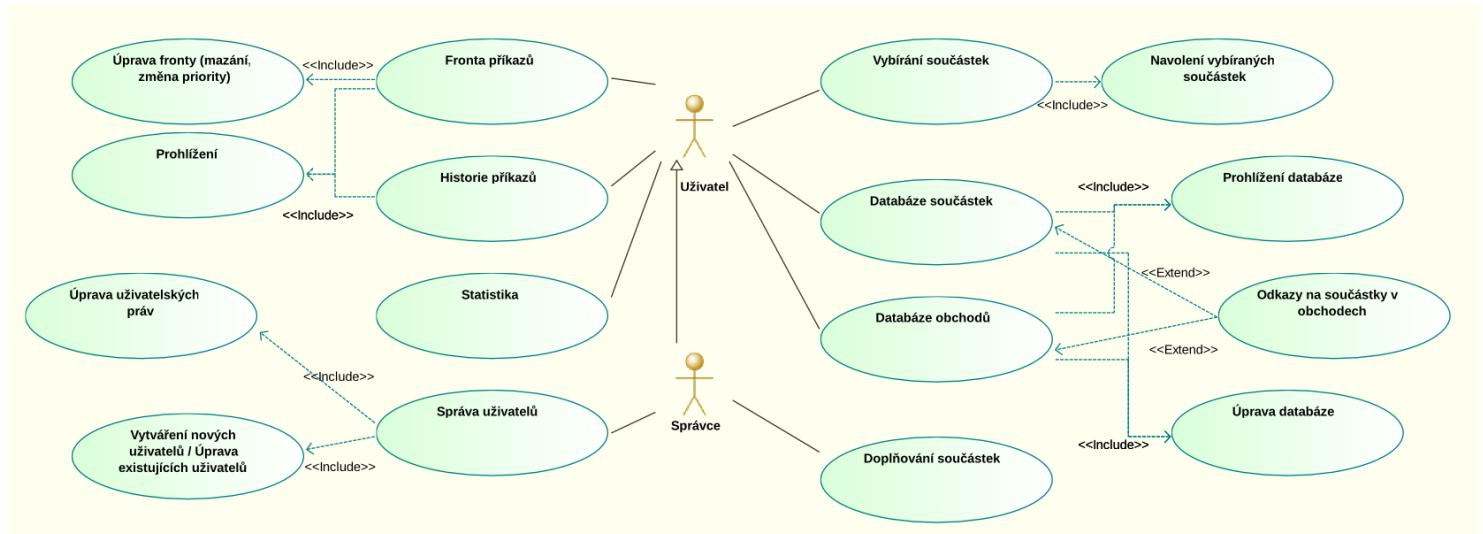
- **User** - objekt uživatele. Obsahuje informace o uživateli, jeho přístupové údaje a uživatelská práva
- **Part** - objekt součástky. Obsahuje základní informace o jednom typu součástky.

<sup>11</sup>způsob komunikace mezi aplikacemi na jednom zařízení

- **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

### 6.1 Funkce



Obr. 4: Use case diagram aplikace

Přední funkcí webu bude navolení a součástek a zařazení jich do fronty na vybrání. Web bude mít ale kromě vydávání součástek ještě několik dalších praktických funkcí, které bude moci vykonávat nezávisle na zařízení.

#### 6.1.1 Změna stavů součástek v přihrádkách (Doplňování součástek)

Web bude umět zobrazit množství součástek v jednotlivých přihrádkách v přehledné tabulce. Tuto tabulku bude moci uživatel měnit (a s ní i hodnoty v databázi), když například doplňuje součástky.

### **6.1.2 Správa příhrádek**

Uživatel bude moci jednoduše změnit typy součástek v jednotlivých příhrádkách a měnit jejich polohu.

### **6.1.3 Databáze součástek**

Jakmile uživatel přidá nějakou součástku, uloží se do databáze její jméno, obrázek a popis. Kromě toho se uloží také šířka a délka, což jsou hodnoty potřebné pro zařízení. Výhledově budou v databázi uloženy i vlastnosti součástky a uživatel si bude moci vybrat součástku jednoduše podle vlastností.

### **6.1.4 Databáze obchodů**

Pro všechny součástky v databázi bude možnost uložit i odkazy na webové stránky součástek v obchodech s elektronikou, takže uživatel si uživatel bude moci rychle vybrat, kde součástku nakoupí, bez zbytečného hledání. Do budoucna počítám i s implementací API některých obchodů do aplikace. Aplikace tak bude moci např. automaticky upozornit uživatele, že nějaká součástka dochází a odkáže ho na obchod s nejnižší cenou.

### **6.1.5 Správa uživatelů**

Uživatelé budou mít různá přístupová práva a ti s vyššími přístupovými právy budou moci vytvořit nový uživatelský účet, nebo změnit nebo smazat již existující s nižšími právy.

### **6.1.6 Fronta příkazů**

Web bude umět zobrazit frontu všech příkazů v tabulce. Uživatel bude poté moci příkaz smazat, nebo jej posunout ve frontě. Pokud nastane chyba, uživatel se z tabulky dozví, jaký příkaz chybu způsobil a bude moci zkuskit příkaz provést znova, nebo jej přeskočit.

### **6.1.7 Historie příkazů**

Po provedení příkazu se řádek v databázi v tabulce fronty příkazů přesune do tabulky historie. Uživatel si na webu bude moci prohlédnout historii příkazů ve zvoleném časovém úseku.

## **6.2 Nette framework**

Nette je PHP framework pro tvorbu dynamických webových aplikací. Poskytuje mnoho předprogramovaných funkcí např. pro přístup k databázi nebo k autentifikaci a autorizaci. Nette framework jsem použil především kvůli šablonovacímu systému a jednoduchému vytváření bezpečných formulářů. Vytváření webových aplikací v Nette je rozděleno podle architektury MVC. Nejprve je potřeba napsat presenter. To je třída, která zpracovává všechna data (např. z databáze) a připravuje je do šablony. Kromě toho také připravuje všechny formuláře a jiné webové komponenty. Následně se zavolá šablona dané stránky,

do které se vloží všechny proměnné a komponenty připravené v presenteru. Odděleně se programují logické části aplikace (např. vlastní autentifikace), které s šablonou nesouvisí.

```
<?php
namespace App\Presenters;
use Nette,
    App\Model;
|
/**
 * PartitionList presenter.
 */
class PartitionListPresenter extends BasePresenter
{
    private $database;
    public function __construct(Nette\Database\Context $database)
    {
        $this->database = $database;
    }
    public function renderDefault()
    {
        if (!$this->getUser()->isInRole('user'))
        {
            $this->flashMessage("Nejste přihlášeni.");
            $this->redirect("Login:");
        }
        $this->template->partitions = $this->database->table('partitions')->select('*');
    }
}
```

Obr. 5: Ukázka kódu presenteru

```
<h1>Seznam přihrádek</h1>


|                   |                                                            |                           |                          |                                                          |                                                            |                                                           |
|-------------------|------------------------------------------------------------|---------------------------|--------------------------|----------------------------------------------------------|------------------------------------------------------------|-----------------------------------------------------------|
| ID                | Součástka (ID)                                             | Počet                     | Šířka                    |                                                          |                                                            |                                                           |
| {\$partition->id} | {\${\$partition->ref('part')}->name} ({\$partition->part}) | {\${\$partition->amount}} | {\${\$partition->width}} | <a href="Partition:edit {\$partition-&gt;id}">Změnit</a> | <a href="Partition:delete {\$partition-&gt;id}">Smazat</a> | <a href="Partition:free {\$partition-&gt;id}">Uvolnit</a> |

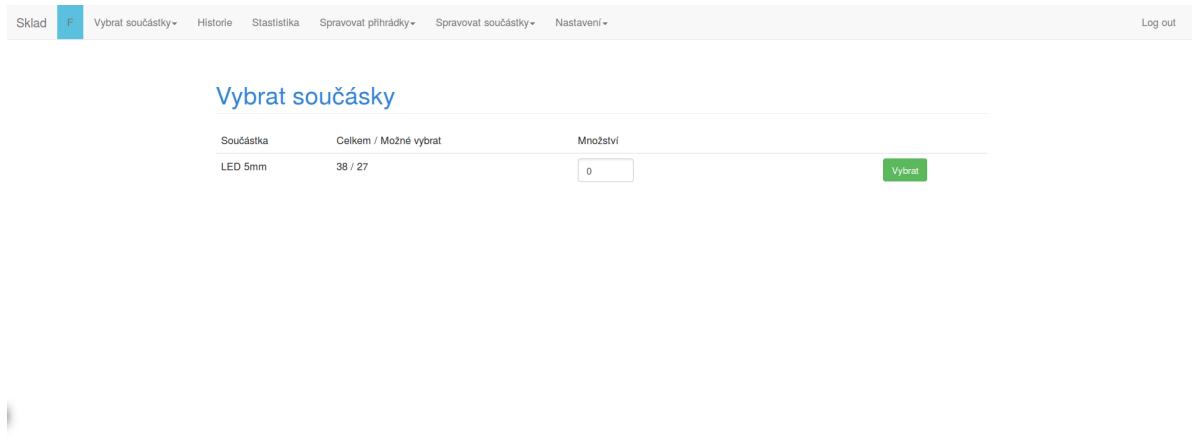

```

Obr. 6: Ukázka kódu šablony

### 6.3 Autentifikace a autorizace

Kvůli bezpečnosti jsem do aplikace implementoval autentifikaci (přihlašování) a autorizaci (přístup k jednotlivým funkcím webu na základě uživatelských práv). Autentifikaci budu řešit jednoduše pomocí jména a hesla. Autorizaci budu řešit pomocí rolí. Každý uživatel bude mít určité role, které mu dovolí přistupovat k určitým funkcím. Role jsem zatím rozdělil jednoduše. Každý uživatelský účet bude mít roli user, která ho bude opravňovat k základním funkcím (vybírání, doplňování součástek). Dále role manager (správce), který bude mít přístup skoro je všem funkcím, kromě správy uživatelských účtů, kde bude moci manipulovat pouze se základními účty s rolí user. Role administrator bude umět manipulovat i s účty s rolemi manager a administrator.

## 6.4 Uživatelské prostředí

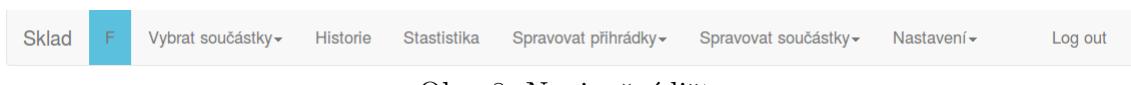


Obr. 7: Screenshot webového rozhraní

Aplikaci budou využívat různí uživatelé, takže je potřeba věnovat velkou pozornost uživatelskému prostředí. Mělo by být především jednoduché a intuitivní, ale také by se mělo dát rychle používat a mělo by umět přehledně prezentovat data. Snažil jsem se také o to, aby nejpoužívanější funkce byly dostupné na co nejméně kliků, zatímco méně používané funkce budou hůře dostupné. Myslím si totiž, že kdybych všechny funkce učinil lehce dostupné, znepřehlednilo by to prostředí.

### 6.4.1 Hlavní komponenty prostředí

#### Navigační lišta



Obr. 8: Navigační lišta

Navigaci po webu na jednotlivé stránky funkcí jsem řešil přes navigační lištu na horním okraji stránky. Zde bude vedle sebe několik odkazů na jednotlivé funkce a několik rozbalovacích seznamů s odkazy na funkce, které jsou podobné (jinak by se vše na liště nevešlo). Například několik stránek pro správu příhrádek je sdruženo do jednoho rozbalovacího seznamu. Na pravém okraji lišty je tlačítko pro odhlášení (odhlašovací tlačítko na pravém okraji navigační lišty považuji za dobrou konvenci, takže jsem ho tam také umístil). Navigační lišta také bude záviset na právech uživatele. Uživateli se budou ukazovat pouze odkazy na funkce, ke kterým má přístupová práva.

## Přihlašování

Nejste přihlášeni.

Username:

Password:

Přihlásit se

Obr. 9: Navigační lišta

Pro přihlašování jsem zvolil řešení prázdné stránky jen s přihlašovacím formulářem. Nebude totiž potřeba prezentovat web jednotlivých skladů nezaregistrovaným uživatelům a ti zaregistrovaní ocení, že nebudou muset přihlašovací formulář hledat.

### 6.4.2 Funkce prostředí

#### Vybírání součástek



#### Vybrat součásny

Součástka	Celkem / Možné vybrat	Množství	
LED 5mm green	16 / 16	0	Vybrat
R0603 1k 1% 1/16W RMC	20 / 20	0	Vybrat
LED 5mm	38 / 27	0	Vybrat
LED 5mm blue	40 / 40	0	Vybrat

Obr. 10: Vybírání součástek

Jelikož je vybírání součástek nejdůležitější a nejpoužívanější funkcí aplikace, na liště na ni odkazuje hned první odkaz. Na stránce bude tabulka všech druhů součástek, které

jsou momentálně ve skladu a bude u nich aktuální počet. Vedle každé součástky bude malé vyplňovací pole, které bude přijímat pouze čísla a tlačítko 'Vybrat'.

## Doplňování součástek

Součástka	Množství	
LED 5mm green	16	<button>Doplnit</button>
R0603 1k 1% 1/16W RMC	20	<button>Doplnit</button>
LED 5mm	38	<button>Doplnit</button>
LED 5mm blue	40	<button>Doplnit</button>

Obr. 11: Doplňování součástek

Doplňování součástek nebude tak moc používaná funkce. Proto jsem se rozhodl začlenit ji do seznamu 'Správa přihrádek' na navigační liště. Uživateli se zobrazí všechny přihrádky se součástkami a bude moci změnit množství součástek v přihrádce (i zmenšit), kterou si vybere.

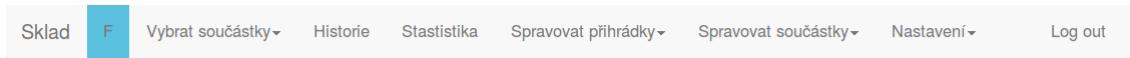
## Správa přihrádek

ID	Součástka (ID)	Počet	Šířka	Změnit	Smazat	Uvolnit
1	LED 5mm green (7)	16	1	<a>Změnit</a>	<a>Smazat</a>	<a>Uvolnit</a>
2	R0603 1k 1% 1/16W RMC (1)	20	1	<a>Změnit</a>	<a>Smazat</a>	<a>Uvolnit</a>
3	LED 5mm (3)	38	1	<a>Změnit</a>	<a>Smazat</a>	<a>Uvolnit</a>
4	LED 5mm blue (5)	40	1	<a>Změnit</a>	<a>Smazat</a>	<a>Uvolnit</a>
5	None (0)	0	1	<a>Změnit</a>	<a>Smazat</a>	<a>Uvolnit</a>

Obr. 12: Seznam přihrádek

Uživatel bude moci jednoduše spravovat všechny přihrádky. Na navigační liště bude odkaz tabulku přihrádek, kde bude moci přihrádku smazat, uvolnit (odstranit součástku z přihrádky) a prohlásit ji za chybovou (systém nedovolí s chybovou přihrádkou pracovat). Dále bude moci přes formulář vytvořit novou, nebo změnit její parametry (pokud se sklad fyzicky mění).

## Správa součástek



### Seznam součástek

ID	Název	Šířka	Délka	Datasheet	Obrázek	Popis			
1	R0603 1k 1% 1/16W RMC	10	20				<a href="#">Smazat</a>	<a href="#">Upravit</a>	<a href="#">Přidat do přihrádky</a>
3	LED 5mm	1	1				<a href="#">Smazat</a>	<a href="#">Upravit</a>	<a href="#">Přidat do přihrádky</a>
4	LED 5mm white	1	1				<a href="#">Smazat</a>	<a href="#">Upravit</a>	<a href="#">Přidat do přihrádky</a>
5	LED 5mm blue	1	1				<a href="#">Smazat</a>	<a href="#">Upravit</a>	<a href="#">Přidat do přihrádky</a>
6	LED 5mm red	1	1				<a href="#">Smazat</a>	<a href="#">Upravit</a>	<a href="#">Přidat do přihrádky</a>
7	LED 5mm green	1	1				<a href="#">Smazat</a>	<a href="#">Upravit</a>	<a href="#">Přidat do přihrádky</a>

Obr. 13: Seznam součástek

Aplikace bude mít svoji databázi součástek, kterou budou uživatelé moci měnit a součástky z ní přiřazovat do přihrádek. Na navigační liště bude odkaz na tabulku součástek, kde uvidí všechny součástky pohromadě a zjistí jejich parametry, případně se může podívat na dokumentaci nebo obrázek. Z této tabulky je možné součástku upravit, smazat, nebo přiřadit do přihrádky. Novou součástku bude moci do databáze zadat pomocí formuláře.

## Správa obchodů

Na navigační liště bude ve skupině 'Spravovat součástky' také odkaz na tabulku obchodů, kde bude moci upravit nebo smazat obchod z databáze. Nový obchod bude moci zadat přes formulář. Součástku může uživatel k obchodu přiřadit přes odkaz v tabulce součástek. Objeví se mu formulář, kam zadá URL stránky součástky v obchodě. Tyto vztahy mezi součástkami a obchody si poté bude moci prohlédnout v detailu součástky.

## Fronta

## Fronta

ID	Stav	Čas	Uživatel	Součástka	Příhrádka	Množství	
9	Čeká	2015-03-04 15:18:15	root	LED 5mm	3	10	<a href="#">Smazat</a>
8	Čeká	2015-03-04 15:18:07	root	R0603 1k 1% 1/16W RMC	2	3	<a href="#">Smazat</a>
7	Čeká	2015-03-04 15:18:04	root	LED 5mm green	1	5	<a href="#">Smazat</a>
6	Čeká	2015-03-04 15:18:00	root	R0603 1k 1% 1/16W RMC	2	3	<a href="#">Smazat</a>
5	Čeká	2015-03-04 15:17:48	root	LED 5mm	3	2	<a href="#">Smazat</a>
4	Čeká	2015-03-04 15:17:45	root	LED 5mm blue	4	10	<a href="#">Smazat</a>
3	Čeká	2015-03-04 15:17:41	root	R0603 1k 1% 1/16W RMC	2	6	<a href="#">Smazat</a>
2	Čeká	2015-03-04 15:17:37	root	LED 5mm green	1	2	<a href="#">Smazat</a>
1	Žádá	2015-03-04 15:17:34	root	LED 5mm	2	1	<a href="#">Smazat</a>

Obrázek 1: Obr. 14: Fronta

Uživatel si bude moci prohlédnout také všechny příkazy ve frontě. Z tabulky může zjistit, jaké součástky jsou ve frontě, kdo příkaz zadal, nebo také jestli nenastala nějaká chyba. Správce bude moci mazat příkazy z fronty.

## 7 Popis daemonu

Jak jsem již zmínil, daemon bude naprogramovaný v jazyku Python. Jeho funkce by měly být:

- Inicializace skladu
- Posílání příkazů skladu
- Zpracování chyb skladu
- Posílání emailových upozornění

### 7.1 Autostart

Daemon bude běžet po celou dobu. To znamená, že se spustí vždy automaticky po startu. Automatické spouštění po startu se v Linuxu dá vyřešit mnoha způsoby. Já jsem využil soubor /etc/rc.local, ve kterém jsou zapsány všechny příkazy, které se po spuštění systému spustí s rootovskými právy. Bez rootovských práv by aplikace nemohla běžet, jelikož využívá přístup k GPIO a sériové lince.

## 7.2 Struktura daemonu

Daemon jsem psal za použití objektově orientovaného programování (třídy), jelikož se daemon dá jednoduše rozdělit na několik částí a také to bude část aplikace, která se bude často upravovat a rozšiřovat (zatímco databáze a webové rozhraní jsou na skladu nezávislé, daemon je naprogramován na míru jednomu zařízení). Použitím objektově orientovaného programování bych chtěl zajistit znovupoužitelnost stávajícího kódu v modifikacích daemonu a přehlednost pro budoucí programátory.

Hlavní třída Daemon bude mít metodu loop(), která bude běžet opakováně a bude zastávat funkci hlavní smyčky programu. Dále třída Database bude poskytovat high-level přístup k databázi (např. třída bude poskytovat přímo jednotlivé příkazy z databáze a samostatně po provedení příkazů upraví data v databázi). Třída Notification bude kontrolovat, zda nemá poslat emailové upozornění, a tato upozornění posílat. Nejdůležitější bude třída StoreInterface, která bude obstarávat komunikaci se zařízením. Bude skladu posílat příkazy a zpracovávat odpovědi a případné chyby.

## 7.3 Posílání emailových upozornění

Pro posílání emailů jsem použil Python SMTP knihovnu smtplib. Upozornění by se měla posílat jednak v reakci na specifická data v databázi (málo součástek na skladu) a také v reakci na chyby skladu, které si žádají pozornost člověka (např. chyba při stříhání pásku).

## 7.4 Popis komunikačního protokolu

Pro komunikaci mezi daemonem a skladem bylo potřeba navrhnout komunikační protokol. Obě strany budou komunikovat přes sériovou linku (UART) pomocí povelů. Předmětem komunikace pomocí UART je sice proud bytů, přesto jsem jako základní komunikační jednotku zvolil paket, který se bude skládat z několika bytů. Pro paket jsem rozhodl kvůli povaze komunikace (posílání povelů). Povely se totiž dají jednoduše posílat jako pakety.

### 7.4.1 Popis paketu

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 zahrnutý 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 přihrádky, 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 zkalibrovalo 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

## 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/>