



**FAKULTA
INFORMAČNÍCH
TECHNologiÍ
ČVUT V PRAZE**

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název: Frontend skladového systému
Student: Bc. Oldřich Malec
Vedoucí: Ing. Jiří Hunka
Studijní program: Informatika
Studijní obor: Webové a softwarové inženýrství
Katedra: Katedra softwarového inženýrství
Platnost zadání: Do konce letního semestru 2019/20

Pokyny pro vypracování

Cílem práce je kompletní tvorba frontendu skladového systému. Návrh nového frontendu bude zohledňovat (a rozšiřovat) funkcionalitu starého skladového systému Sysel od společnosti Jagu s.r.o. Při návrhu spolupracujte s Bc. Pavlem Kovářem, který bude připravovat serverovou část systému. Zpracujte kompletně minimálně roli skladníka a manažera skladu.

Postupujte dle následujících kroků:

1. Analyzujte všechny případy užití současného řešení a zjistěte, jaké nové funkce by měl systém také podporovat. Při řešení neopomeňte zhodnotit konkurenční řešení.
2. Analyzujte možnosti evidence logistiky - správy zboží připraveného k vyskladnění.
3. Na základě analýzy proveďte vhodný návrh, zaměřte se na efektivitu práce jednotlivých uživatelských rolí.
4. Návrh zrealizujte v podobě funkčního prototypu.
5. Prototyp podrobte vhodnými Vámi navrženými testy.
6. Na základě testování prototyp upravte.
7. Společně s Bc. Pavlem Kovářem zajistěte vydání alfa verze.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 13. února 2019

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

KATEDRA SOFTWAREVÉHO INŽENÝRSTVÍ



Diplomová práce

Frontend skladového systému

Bc. Oldřich Malec

Vedoucí práce: Ing. Jiří Hunka

25. listopadu 2019

Poděkování

TODO Tým z NURů

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. Dále prohlašuji, že jsem s Českým vysokým učení technickým v Praze uzavřel licenční smlouvu o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ust. § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 25. listopadu 2019

.....

České vysoké učení technické v Praze

Fakulta informačních technologií

© 2019 Oldřich Malec. Všechna práva vyhrazena.

Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci

MALEC, Oldřich. *Frontend skladového systému*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019. Dostupný také z WWW: <https://gitlab.fit.cvut.cz/malecold/master-thesis>.

Abstrakt

TODO

Klíčová slova TODO

Abstract

TODO

Keywords TODO

Obsah

Úvod	1
1 Analýza	3
1.1 Analýza požadavků dle současného systému	3
1.2 Analýza nových požadavků	5
1.3 Analýza konkurence	6
2 Návrh, realizace a testování Lo-Fi a Hi-Fi prototypů	11
2.1 Návrh uživatelského rozhraní - Lo-Fi prototyp	11
2.2 Návrh uživatelského rozhraní - Hi-Fi prototyp	12
2.3 Testování prototypu	17
3 Návrh funkcí a architektury reálné aplikace	19
3.1 Návrh pokročilých funkcí aplikace	19
3.2 Návrh procesů v novém skladovém systému	23
3.3 Volba technologie	24
4 Implementace	35
4.1 Nízkoúrovňové vývojové nástroje	35
4.2 Router	37
4.3 Plnohodnotná webová aplikace	42
4.4 Překlady	43
4.5 State management pattern	43
4.6 Hlídání konektivity	45

4.7	Renderování formulářů	46
4.8	Realizace podpory pro undo	50
4.9	Generování kódu ze specifikace OpenDoc	51
4.10	Barevná paleta aplikace	54
4.11	Záznam stráveného času v úkolech	58
4.12	Perličky z vývoje	60
5	Testování	63
5.1	Testování aplikace	63
	Závěr	65
	Zdroje	67
A	Seznam použitých zkratk	73
B	Slovník pojmů	75
C	Diagramy průchodu systému skladníkem	79
D	Návrh procesů ve skladovém systému	83
E	Zápis z uživatelského testování aplikace během vývoje	95
F	Obsah přiloženého média	99

Seznam ukázek kódu

2.1	Základní soubor index.html pro Hi-Fi prototyp	14
2.2	Funkce pro načtení obsahu komponent Hi-Fi prototypu	15
4.1	Nastavování titulků stránek pomocí Vue routeru - úprava definic . .	39
4.2	Nastavování titulků stránek pomocí Vue routeru - úprava instance routeru	40
4.3	Automatické generování drobečkové navigace	41
4.4	Manifest pro webové aplikace	42
4.5	Definice překladů pro i18n	44
4.6	Vuex pro snackbar-message: definice	45
4.7	Vuex pro snackbar-message: Snackbar komponenta	45
4.8	Vuex pro snackbar-message: použití z jiné komponenty	46
4.9	Použití mixinu pro zjednodušení zasílání zpráv do Snackbaru	46
4.10	Příklad definice formuláře: jednoduché skladové umístění	48
4.11	Parsování názvů argumentů funkce	52
4.12	Diff nastavení fontu ikon ve Vuetify	61

Seznam tabulek

3.1	Volba frameworku: Datum vydání	25
3.2	Volba frameworku: Zázemí	25
3.3	Volba frameworku: Licence	26
3.4	Volba frameworku: Obtížnost	27
3.5	Volba frameworku: Dokumentace	28
3.6	Volba frameworku: Testování	29
3.7	Volba frameworku: Devtools	30
3.8	Volba frameworku: Počet hvězdiček na GitHubu	30
3.9	Volba frameworku: Počet npm balíků	31
3.10	Volba frameworku: Otázky na Stack Overflow	31
3.11	Volba frameworku: Shoda s firemním stackem	32
3.12	Volba frameworku: Počet vývojářů na LinkedIn	32
3.13	Volba frameworku: Integrace se Sentry	33
3.14	Volba frameworku: Výsledky	33

Seznam obrázků

1.1	Ukázka špatné práce s horizontálním místem ve starém skladovém systému	6
2.1	Rozhraní skladníka pro naskladňování položek: starý skladový systém	12
2.2	Rozhraní skladníka pro naskladňování položek: Lo-Fi prototyp . . .	13
2.3	Rozhraní skladníka pro naskladňování položek: Hi-Fi prototyp . . .	16
4.1	Analýza velikosti výsledné aplikace: překlady moment.js	38
4.2	Navigace na standardní webové stránce	39
4.3	Navigace v aplikaci pomocí Vue Router	40
4.4	Formulář vykreslený pomocí jednotné formulářové komponenty . .	50
4.5	Undo: Možnost vrácení provedených změn	50
4.6	Undo: Potvrzení vrácení provedených změn	51
4.7	Barevná paleta aplikace: Trojúhelníkové schema	55
4.8	Barevná paleta aplikace: Rozděleně-doplňkové schema	56
4.9	Barevná paleta aplikace: Analogové schema	56
4.10	Barevná paleta aplikace: Zvolené schema pro mou aplikaci	57
4.11	Barvy znázorňující idempotenci akcí: Dialog běžícího času	58
4.12	Barvy znázorňující idempotenci akcí: Tlačítko tvorby úkolu	58
C.1	Diagram průchodu systému skladníkem: první část	79
C.2	Diagram průchodu systému skladníkem: druhá část	80
C.3	Diagram průchodu systému skladníkem: třetí část	81

Úvod

TODO

Analýza

Cílem nového skladového systému je nahradit a rozšířit funkce stávajícího skladového systému Sysel - proto jsme při analýze požadavků vycházeli¹ jednak z tohoto řešení a dále z požadavků primárního potenciálního zákazníka - jednoho ze současných uživatelů tohoto systému.

Současná verze Sysla je již oproti svému původnímu návrhu velmi upravena, a to se podepisuje na snížené možnosti dalších úprav a celkové komplexnosti systému.

Pro nový systém je na jedné straně důležité, aby bylo stále možné provádět ty procesy, které jsou ideální již za současného stavu. Na straně druhé je pak důležité zohlednit nové požadavky a pokusit se umožnit i snadné reakce na budoucí požadavky.

1.1 Analýza požadavků dle současného systému

Jádrem současného skladového systému je rozdělení na dvě role: skladník a vedoucí. Aplikace podporuje práci ve středně velkém skladu, který používá čárové kódy na zboží a umístěních, ale nemá další pokročilé automatizace jako například pohyb objednávkových bedýnek po páse, roboty, kteří autonomně vyzvedávají zboží atp. Veškeré pohyby jsou realizovány lidskými zdroji a tomu odpovídá

¹Já a kolega Pavel Kovář, který se zabývá backendem nového systému

i realizace podpůrného systému. Výhodou tohoto řešení je možnost použití systému i v menších skladech a to případně i o velikostí pouze jednoho správce.

1.1.1 Funkční požadavky

Role skladníka:

- přijímat nové dodávky zboží,
- naskladňovat zboží,
- přesouvat zboží v rámci skladu,
- přesouvat celá umístění,
- vyskladňovat zboží,
- provést inventuru,
- zobrazovat úlohy.

Role vedoucího:

- spravovat sklady,
- spravovat umístění ve skladech,
- spravovat výrobce,
- spravovat zboží,
- zadávat úlohy skladníkům,
- sledovat stav probíhajících a uzavřených úloh.

Tyto požadavky vychází ze stávající aplikace Sysel a jejich procesy měly zůstat zachovány, proto jsme konkrétní use-case a detailní procesy navrhli na základě průchodů současného systému. Diagramy, které jsou výstupem této části analýzy jsou součástí přílohy C.

1.2 Analýza nových požadavků

1.2.1 Logistika

Hlavním novým požadavkem, který současný Sysel nepodporuje, je správa *Logistiky* - tedy místa ve skladu, kde dochází ke kompletaci odchozího zboží, jeho balení atp.

Toto místo se od běžného skladového umístění odlišuje v tom, co se zde navíc eviduje:

- *Použitý spotřební materiál* Zaznamenávají se spotřebované balíky či fólie, ale například i palety.
- *Doba uložení* Ukládá se, jak dlouho zde dané zboží „zabíralo“ místo.
- *Činnosti skladníka* Zde se eviduje, kolik a jaké úkony musel skladník se zbožím provést - jako například *balení do folie, přeskládání* atp.

1.2.2 Evidence stráveného času

Podobně jako se u *Logistiky* má ukládat čas, po který zboží leželo v Logistice, mělo by být možné v aplikaci celkově evidovat, kolik času strávil skladník jakým úkolem. Při přijetí úkolu se automaticky spustí stopky měřící strávený čas daným úkolem, také je ale nutné mít možnost stopky pozastavit - například během pauzy na oběd apod.

Časové přehledy skladníků by měly být použitelné jednak pro kontrolu výkonu skladníků, ale také pro potenciální fakturaci nákladů třetím stranám - pro případ, že by sklad provozoval jiný subjekt, než je reálný majitel zboží, který platí za uskladnění.

1.2.3 Rozhraní pro nemobilní zařízení

Současný systém je navržen mobile-first, což by mělo být zachováno - skladníci používají mobilní zařízení se zabudovanou čtečkou. Toto rozhraní se ale prakticky vůbec nezmění i při použití větší obrazovky - na tabletech je rozhraní ještě poměrně použitelné, ale na notebooku či stolním počítači je zobrazení přinejmenším *neoptimalizované* - neexistuje zde žádná práce s horizontálním místem, vše je řazeno vertikálně. Jako ukázkou přikládám screenshot úryvku

1. ANALÝZA



Ukázky čekající na schválení (0)			
Čekající tržby a vyskládání (7)			
#51711Vyskládání	Eshop s.r.o. (ESHOP) - Objednávky	(volná)	21.6.2019 20:10
#51712Vyskládání	Eshop s.r.o. (ESHOP) - Objednávky	(volná)	21.6.2019 20:10
#51721Vyskládání	Eshop s.r.o. (ESHOP) - Objednávky	(volná)	22.6.2019 18:10
#51753Vyskládání	Eshop s.r.o. (ESHOP) - Objednávky	(volná)	23.6.2019 15:00
#51755Vyskládání	Eshop s.r.o. (ESHOP) - Objednávky	(volná)	23.6.2019 15:00
#51770Vyskládání	Eshop s.r.o. (ESHOP) - Objednávky	(volná)	23.6.2019 15:00
#51784Vyskládání	Eshop s.r.o. (ESHOP) - Objednávky	(volná)	24.6.2019 07:40
Čekající dodávky (0)			
Vytvořit maskované bez dodávky			
Přehledy a srovnání (12)			
SA1	2 (1) polož.	2 (1) k.	
SA1	1 (1) polož.	1 (1) k.	
Export papírově (XLSX)			

Obrázek 1.1: Ukázka špatné práce s horizontálním místem ve starém skladovém systému

domovské obrazovky vedoucího skladu (obrázek 1.1), obsahující náhled na reálný stav používaného skladu, s anonymizovanými daty.

Požadavkem tedy je, aby rozhraní bylo plně responzivní - skladník sice typicky na počítači opravdu používán nebude, ale vedoucí skladu by měl mít možnost volně přecházet mezi různými zařízeními.

1.3 Analýza konkurence

Před započítím tvorby samotných návrhů je potřeba analyzovat konkurenční řešení.

V dnešní době existuje na trhu nespočet skladových systémů, jednak veřejně dostupných, které nabízejí například osekanou verzi zdarma, jiné placené, ale s popisem jejich vlastností, a pak také spousta uzavřených systémů, které si nechá někdo vytvořit na míru přesně pro své potřeby, a pro veřejnost není daný systém vůbec dostupný.

Z tohoto důvodu jsem se rozhodl zaměřit především na funkcionality veřejně dostupných skladových systémů, které jsou primárně určené na mobilní zařízení.

1.3.1 Analýza mobilních aplikací pro evidenci skladu

1.3.1.1 Storage Manager: Stock Tracker

Tuto aplikaci lze nalézt v Google Play Store a je k dispozici její free varianta, kterou jsem také vyzkoušel.

Stejně jako Sysel umožňuje skenovat čárové kódy nejen na zboží, ale i na umístění. Má podobné možnosti manipulace se zbožím: naskladnění, vyskládání, přesun, inventura. Navíc umožňuje pracovat i s objednávkami.

Synchronizace probíhá přes úložiště třetí strany (Dropbox...) a to vždy pouze při startu / ukončení a nebo na vyžádání. Jedná se tedy primárně o single-user systém. Zdá se, že nejsou podporovány ani různé role.

Aplikace vůbec nepočítá se systémem "úkolů". Skladník zde musí sám vědět, co má dělat - nebo informace zjišťovat z jiného systému - nebo pracovat s objednávkami, které systém narozdíl od SYSLA podporuje.

Naskladnění, vyskladnění a i přesun zboží funguje vždy pouze s jedním typem zboží - nelze hromadně přesunout např. celou paletu, na které je různé zboží. U každé položky se znovu vyplňuje celý formulář.

Zajímavé funkcionality

- Sken čárových kódů klasickým fotoaparátem zařazení.
- Možnost konfigurace, které prvky se na homepage zobrazují.
- Možnost konfigurace, které atributy skladových položek, objednávek atp. se zobrazují.

Analýza UI Jedná se o nativní android aplikaci, v designu Androidu Jelly Bean.

Klady UI

- U pole množství jsou vždy zobrazena + a - pro usnadnění rychlých změn.
- Focus na prvcích logicky přeskakuje na další, na nových obrazovkách je většinou jako výchozí zvolen EAN.
- Pole výběru data nabízí nativní Androidí kalendář.
- Přehled provedených transakcí.
- V jakýchkoliv seznamech lze vždy hledat, řadit i filtrovat.
- V detailu produktu je dole vždy informační proužek zobrazující, kolik kusů zboží je na skladě.

Zápory UI

- Důležité prvky (uložení nového zboží) jsou vždy umístěny v horní liště, která někdy může být špatně dostupná.
- Autofocus na některých prvcích neotevívá automaticky klávesnici, je tedy stejně nutné znovu do pole tapnout.
- Aplikace nemá menu. Tudíž když se zanořím někam hluboko, musím hodněkrát použít „back“, abych se dostal na domovskou obrazovku.
- Při hromadném vyskladňování nelze pracovat „z jednoho místa“ - u každého produktu se vždy volí znovu umístění - nezůstává ani nepředvyplněné.
- Celkově se jedná spíše o jednoduchý seznam potřebných informací, UI není nějak extra nápomocné.

1.3.1.2 Simple Stock Manager

Tato aplikace je také dostupná na Google Play zdarma, avšak narozdíl od předchozí testované, tato nemá plnou verzi za peníze, ale obsahuje reklamy. Nabízí pouze základní funkcionality naskladnění a vyskladnění (neumí tedy ani řešit umístění, zdaleka neumí role, synchronizaci, úkoly, dodací listy, faktury...) Provedené transakce lze zpětně upravovat - pro použití této aplikace je to vhodné (je určena pro single-user použití), pro Sysla je to funkce nevhodná z důvodu kontroly nad stock flow.

Zajímavé funkcionality

- Kalkulačka u polí množství (lze tak efektivně zadat například „5*50 + 7“.
- Možnost zobrazení přehledu produktů, kterých je na skladě málo.

Analýza UI Jedná se o jednoduchou aplikaci, která má ale některé zajímavé funkcionality.

Nabízí například zajímavé grafy pohybu konkrétního zboží.

Klady UI

- Při zadávání množství v naskladnění zobrazuje výsledek, kolik bude na skladě po naskladnění.
- Má menu, které lze vytáhnout z levého okraje. Jednoduše se tak vždy rychle dostanu tak, kam potřebuji.
- Nejvíce potřebné akce jsou na domovské obrazovce dostupné v řádku ve spodní části obrazovky.
- V horní části obrazovky je nástřel drobečkové navigace - bohužel ale aplikace nemá více než 1 zanoření a tudíž je nevyužita.

Zápory UI

- Datum se vybírá z vlastního formuláře, který se používá hůře než vestavěný androidí.
- Dokončení akce zobrazí vždy potvrzovací dialog, což sice na první pohled může vypadat jako dobrý nápad, ale při přehnaném používání těchto dialogů ztrácí uživatel zájem se nad dialogem rozmyšlet, a všechny automaticky potvrzuje, čímž dialog ztrácí smysl [44].
- Chybí auto focus při otevření nových stránek na důležité prvky (vyplnění EAN, hledání...).
- Ovládací prvky UI (potvrzení atp.) jsou často malé a špatně se na ně na dotykové obrazovce strefuje.
- Z přehledu zboží s nízkou skladovostí lze kliknout na „add“ u konkrétního výrobku. Otevře se stránka naskladnění, zvolený výrobek ale není předvyplněn.
- Na domovské obrazovce je plovoucí tlačítko pro přidání nového pohybu zboží. Toto tlačítko ale někdy překrývá jiné ovládací prvky, na které není možné kliknout ani při pokusu odscrollovat níže, protože tam už stránka často končí.
- Když je menu v levé části vytahováno gestem (posun prstu od kraje obrazovky), je nutné táhnout prst asi až do polovny obrazovky, jinak se menu neotevře.

1. ANALÝZA

- V seznamu zboží je pro otevření detailu nutné kliknout na malou akci „Show“, klik na celý řádek produktu nedělá nic.

// TODO další konkurence

Návrh, realizace a testování Lo-Fi a Hi-Fi prototypů

2.1 Návrh uživatelského rozhraní - Lo-Fi prototyp

V poděkování jsem zmiňoval, že při prvotním návrhu UI jsme spolupracovali jako tým v předmětu MI-NUR. V rámci naší semestrální práce jsme navrhovali nový skladový systém vycházející ze Sysla, a zaměřili jsme se pouze na pohled skladníka.

První částí návrhu GUI byl mockup, či chcete-li wireframe. Pro jeho tvorbu jsme využili nástroj Axure RP [49] - jedná se o komplexní nástroj pro tvorbu prototypů, bez nutnosti psaní programového kódu. Lze do něj doinstalovat externí knihovny, které například přidávají hotové designové či funkční prvky a má vlastní řešení kolaborace a verzování ve stylu SVN.

Pro naše potřeby jsme tedy do Axure doinstalovali UX Tool Time [64], což je knihovna pro Axure, která nabízí komponenty v *Material designu*. S její pomocí jsme vytvořili návrhy na základní procesy skladníka.

Zdrojové kódy tohoto návrhy jsou k dispozici buďto na přiloženém médiu, nebo na webové adrese <https://zho703.axshare.com/>.

Příklad jedné obrazovky - konkrétně naskladnění - vkládám i přímo do textu jako obrázek 2.2. Ještě předtím je ale pro porovnání vložena i ukázka stávajícího

Obrázek 2.1: Rozhraní skladníka pro naskladňování položek: starý skladový systém

skladového systému - taktéž rozpracovaného úkolu naskladnění (obrázek 2.1).

2.2 Návrh uživatelského rozhraní - Hi-Fi prototyp

Dalším krokem návrhu uživatelského rozhraní byla tvorba Hi-Fi prototypu, neboli aplikace, která je již vytvořena za použití některých technologií cílové platformy, avšak neobsahuje backend - neukládá žádná data a vždy zobrazuje pouze předpřipravený obsah - vždy však musí správně reagovat na interakci uživatele.



Obrázek 2.2: Rozhraní skladníka pro naskladňování položek: Lo-Fi prototyp

Hi-fi prototyp jsme opět realizovali jako tým v rámci předmětu MI-NUR, avšak přípravu technologie jsem provedl já samostatně a její realizace stojí za zmínku.

2.2.1 Technologie použité v Hi-Fi prototypu

Cílem bylo vytvořit jakýsi mikro-framework, ve kterém bude snadné tvořit Hi-Fi prototyp, a současně bude možné používat standardní prvky Material designu. Kód měl být psán v jazyku cílové platformy, tedy HTML + CSS + JS. Jelikož nikdo z nás neměl zkušenosti s žádným z Javascriptových frameworků typu Angular či React, a chtěl jsem pro naše řešení mít co nejméně závislostí,

rozhodl jsem se vše napsat v čistém Javascriptu (pouze za použití dnes de-facto standardní knihovny jQuery). Základem je jeden výchozí HTML soubor, viz ukázka kódu 2.1.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>SYSEL</title>
5
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <meta charset="UTF-8">
8
9   <!-- Material components all-in-one -->
10  <link rel="stylesheet" type="text/css" href="https://unpkg.com/
11    material-components-web@latest/dist/material-components-web.min.css"/>
12 </head>
13 <body>
14   <div id="main-content" class="homepage-include"></div>
15
16   <!-- Material design -->
17   <script src="https://unpkg.com/material-components-web@latest/dist/
18     material-components-web.min.js"></script>
19   <!-- jQuery -->
20   <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/
21     2.2.4/jquery.min.js"></script>
22   <!-- Custom scripts -->
23   <script src="js/main.js"></script>
24 </body>
25 </html>
```

Ukázka kódu 2.1: Základní soubor index.html pro Hi-Fi prototyp

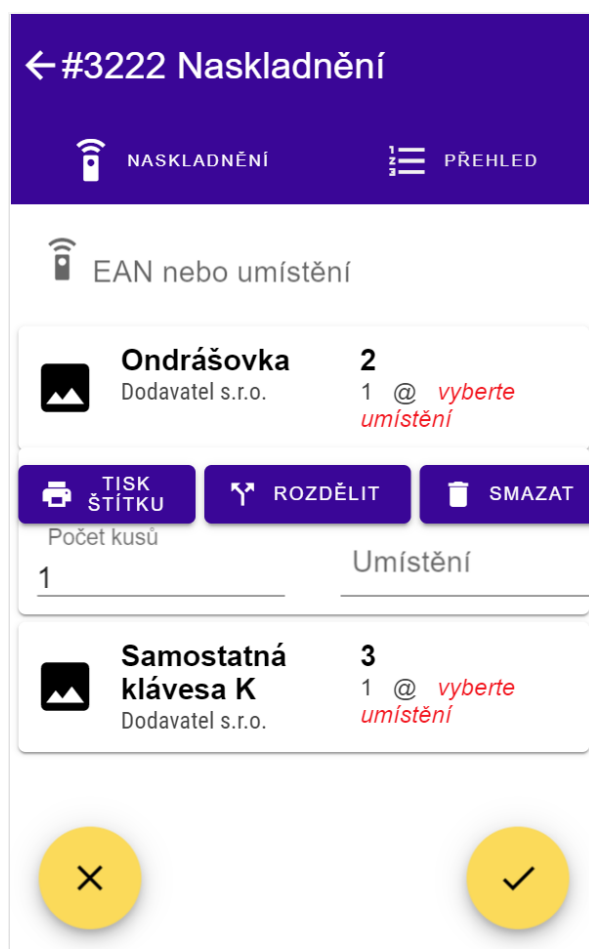
Veškerý ostatní obsah je *komponenta*, což je HTML šablona, která může být libovolně vkládána do jiných šablon, několikrát vedle sebe apod. Postupně jsem umožnil komponenty libovolně parametrizovat a samozřejmě je mezi sebou vyměňovat - například přechod na zcela jinou stránku je realizován přes *výměnu komponenty*, která se zobrazuje v #main-content v ukázce kódu 2.1. Toto vše je možné díky pouhým 67 řádkům JS kódu. Úryvek je vidět v ukázce kódu 2.2.

```
1 function loadDynamicContent(selector = '[class$="-include"]', callback){
2
3   $(selector).each(function (e) {
4     // load all custom components
5     let parent = $(this);
6     $(this).html('');
7     $(this).load("components/" + $(this).attr("class")
8       .replace("-include", "") + ".html", function() {
9
10      // load sub-components
11      $(is).find('[class$="-include"]').each(function() {
12        loadDynamicContent(this, function(){
13          insertParameters(parent)
14        });
15      });
16      if ($(this).find('[class$="-include"]').length === 0) {
17        insertParameters(parent);
18      }
19      callback(this);
20    });
21  });
22 }
23
24 function insertParameters(parent) {
25   let parameters = parent.attr('data-include-content');
26   if (parameters !== undefined) {
27     $.each(JSON.parse(parameters), function(key, value){
28       parent.html(parent.html().replace('{ ' + key + ' ', value));
29     })
30   }
31 }
```

Ukázka kódu 2.2: Funkce pro načtení obsahu komponent Hi-Fi prototypu, včetně podkomponent

Funkce `loadDynamicContent` z ukázky kódu 2.2 funguje tak, že projde existující kód (výchozí je vždy index z ukázky kódu 2.1) a veškeré výskyty tříd ve tvaru `něco-include` nahrazuje obsahem souboru `něco.html` ze složky `components/`. Díky tomu, že Javascriptu může být umožněn přístup na filesystem, lze takto dynamicky načítat obsah souborů z disku a renderovat ho přímo v již načteném obsahu. Tato technika je v některých browserech zablokována kvůli CORS², a tak pro použití například v Google Chrome je nutné zmiňovaný index spouštět

²Cross-origin resource sharing - ochrana před načítáním nezabezpečeného obsahu



Obrázek 2.3: Rozhraní skladníka pro naskladňování položek: Hi-Fi prototyp

přes libovolný webserver, nelze jej přímo otevřít z filesystému. Naopak například v Mozilla Firefoxu je CORS v tomto ohledu volnější, a celý tento mikro-framework tak lze používat bez jakéhokoliv lokálního serveru, stahování závislostí atp. - stačí pouze otevřít index.

Zdrojové kódy Hi-Fi prototypu, který následně v tomto „frameworku“ vznikl, jsou dostupné na příloženém médiu a také v repozitáři na Gitlab FIT [11]. Stejně jako u Lo-Fi prototypu, i zde vkládám obrazovku naskladnění přímo do textu jako obrázek 2.3.

2.3 Testování prototypu

Hi-Fi prototyp popisovaný v předchozí sekci, který je také dostupný na přiloženém médiu, byl testován v rámci předmětu MI-NUR společně mnou, Pavlem Kovářem, Martinem Kubišem a Jakubem Štercem. Abych nekradl cizí práci, uvedu v tomto textu pouze části, které jsou realizoval já osobně, i výstup ostatních kolegů jsem ale zhodnotil v samotné implementaci následné funkční aplikace.

2.3.1 Heuristická analýza

Zvolili jsme několik akcí skladníka, které jsme podrobili heuristické analýze. Já konkrétně jsem řešil obrazovku naskladnění (screenshot přiložen v předchozí kapitole jako obrázek 2.3), která dopadla následovně:

1. *Viditelnost stavu systému:*

Vše v pořádku

2. *Shoda mezi systémem a realitou:*

OK

3. *Minimální zodpovědnost (a stres):*

Při otevření tohoto tasku se mu ten hned přiřadí. Pro navrácení je nutné jej stornovat a vyplnit, že se jednalo o chybu. V systému o tom však bude záznam. Dále není jasné, zda tlačítka ve spodní části úkol již odeslou, nebo se teprve zobrazí nějaký formulář a potvrzení.³

4. *Shoda s použitou platformou a obecnými standardy:*

V pořádku (material design)

5. *Prevence chyb:*

Úkol nelze dokončit, pokud nějaké zboží nemá vyplněno umístění.⁴ Jiné chyby na této stránce (z aplikačního hlediska) prakticky nelze udělat => OK

³V reálné aplikaci je toto vyřešeno: úkol při otevření - pokud ještě není přiřazen - zobrazí pouze informace a tlačítko pro přiřazení úkolu „sám sobě“. Teprve poté je možné v něm provádět práci. Tlačítka ve spodní části byla předělána na běžná tlačítka a byla přidána další obrazovka, kde se úkol dokončuje. Zda tlačítko něco odesílá, řeší rozlišení jejich barev, což jsem popisoval v sekci 4.10.

⁴V reálné aplikaci toto řešeno výrazným označením zboží, které ještě není umístěno.

6. *Kouknu a vidím:*

Při větším množství položek může být seznam dlouhý a nepřehledný

7. *Flexibilita a efektivita:*

OK - zkušený může stále pítat, nový může zkoumat dostupné vstupy.

8. *Minimalita (Klapky na očích):*

OK - obsah karet bude ve finální aplikaci konfigurovatelný⁵.

9. *Smysluplné chybové hlášky:*

Jediná chybová hláška o vyplnění umístění je jasná.

10. *Help a dokumentace:*

Dokumentace není⁶

// TODO more

⁵V době psaní textu se jedná o jeden z návrhů na budoucí vylepšení aplikace

⁶Reálná aplikace základní nápovědu / dokumentaci nabízí

Návrh funkcí a architektury reálné aplikace

3.1 Návrh pokročilých funkčností aplikace

V této sekci budu diskutovat funkce, které byly analyzovány ještě před započetím implementace samotné aplikace a jejichž analýze byl věnován určitý nezanedbatelný čas.

3.1.1 Undo

Undo, neboli možnost vrácení akce, tlačítko *zpět*, *vrátit* či *stornovat*. Tím vším je myšlena možnost odvrátit poslední akci, kterou uživatel v systému provedl. Undo je svým způsobem alternativa potvrzovacím dialogům, u kterých bych se chtěl ještě na chvíli zastavit.

Potvrzovací dialogy Všichni je známe - modální okna, typicky s otázkou a dvěma akcemi: *opravdu smazat/odeslat/...* a *storno*. Jsou používány tam, kde jejich potvrzení je většinou nevratné a dojde k provedení něčeho důležitého. Co když je ale provádění důležitých akcí stěžejní funkčností systému? V příkladu skladového systému jsou veškeré přesuny zboží, naskladnění, vyskladnění atp. důležité akce, u kterých by si měl skladník zkontrolovat, že realita odpovídá stavu v systému. Potvrzovací dialogy na všech těchto akcích ale mohou uživatele i zdržovat - akce, která je v systému vykonávána s vysokou frekvencí,

by neměla stále dokola vyžadovat dvojité potvrzení. Nejen že to bude uživatele zdržovat, ale i samotná četnost zapříčiní, že potvrzovací dialog ztratí svůj účel - uživatel bude zvyklý jej *ihned odkliknout*, aniž by se zamyslel nad tím, zda akci chce opravdu provést [44].

Undo Možnost vrátit akci zpět je svým způsobem volnější než potvrzovací dialog - nezdržuje dvojitým potvrzením, ale v případě nutnosti umožňuje špatné rozhodnutí vrátit. Většinou ale nedává moc času na rozmyšlenou - možnost vrátit akci se typicky zobrazuje pouze několik sekund. Také implementace je často mnohem složitější a vyžaduje řádný návrh a spolupráci prezenční a modelové či datové vrstvy.

Při návrhu *undo tlačítka* pro skladový systém jsme společně s Pavlem Kovářem⁷ zanalyzovali několik možností:

1. **Fronta ve frontendu:** Frontendu by při provedení akce, která by podporovala undo, požadavek reálně neodeslal, ale uložil by jej k pozdějšímu odeslání - například za 5 sekund. Při kliku na undo by se odložené odeslání akce stornovalo, v opačném případě by se požadavek odeslal buďto po uplynutí stanoveného doby, nebo před provedením jiné další akce.

Zhodnocení řešení: Jedná se o nejsnadnější implementaci - nevyžaduje žádné úpravy backendu a vše se řeší pouze na klientských zařízeních. S tím ale přichází i hlavní nedostatky tohoto řešení: Když klient ukončí či ztratí spojení se serverem, požadavek se reálně neodešle, přestože systém hlásí, že je odeslaný. Dále když jiný klient provede stejnou akci, dorazí pak na server obě a druhá nemůže být provedena - je potřeba o tom uživatele informovat a problém řešit. Oba problémy jsou poměrně zásadní, a tak byla tato varianta provedení zavržena.

2. **Fronta jako middleware:** Frontend by odesílal požadavky ihned, avšak mezi ním a backendem by existoval ještě prostředník: fronta ke zpracování. Ta by požadavky zachytávala a čekala s jejich reálným odesláním backendu. Chování by bylo podobné, jako kdyby s odesláním čekal frontend v předchozí variantě.

⁷spolupřítel návrhového řešení - zabývá se backendovou částí aplikace

Zhodnocení řešení: Jedná se o řešení, jak předcházet problémům s potenciální ztrátou připojení. Middleware by sloužil jako fronta pro všechny klienty serveru. Když pak klient ztratí či ukončí připojení, požadavek se provede, neboť middleware jej na backend odešle sám. Při ztrátě připojení by se uživateli, který by chtěl použít undo zobrazil informace o ztrátě připojení a informace, že požadavek již byl proveden a bohužel nemůže být stornován.

I zde ale přetrvává problém se synchronizací mezi více klienty - frontend by musel pro každý požadavek, který šel přes frontu, zpětně kontrolovat, že byl nakonec vykonán bezchybně a případně informovat uživatele. To je nepohodlné, ale na druhou stranu ne zcela nereálné řešení.

3. **Fronta v backendu:** Požadavky by se odesílaly na backend ihned, a ten by si sám držel frontu obdržených požadavků, ke kterým ještě může přijít požadavek na undo. Opět po uplynutí doby, nebo přijetí požadavku na potvrzení by se změny reálně provedly a potvrdily.

Zhodnocení řešení: Třetí řešení je velmi podobné tomu druhému, změnou je, že když si bude frontu udržovat přímo backend, může ten s čekajícími požadavky již počítat při vracení dat jiným klientům (soft lock) - a tím předcházet nekonzistencím v datech: Když uživatel A vyskladní poslední kus výrobku, ale tento požadavek ještě čeká ve frontě, uživateli B se již bude zobrazovat, že je na skladě 0 kusů tohoto výrobků a nemůže ho tak vyskladnit duplicitně.

Problémem tohoto řešení je velmi velká náročnost na zpracování těchto zámků a poměrně vysoká šance na neúmyslné chyby v kódu. Z toho důvodu jsme nakonec tuto variantu také zavrhlí.

4. **Změnové vektory v backendu:** Frontend by vše odesílal hned, backend by vše ihned zpracovával. Kromě zpracování by ale ukládal i změnové vektory, které by bylo možné zpětně odvolat - provést *rollback*. Změnové vektory jsou technikou často používanou například v databázových strojích - veškeré změny jsou zaznamenávány a posléze je možné vše zpětně přehrát a vrátit tak předchozí stav [65].

Zhodnocení řešení: V běžné implementaci tato technika umožňuje navrátit

stav *celé databáze*, nikoliv pouze jednoho uživatele. Pokud by tedy chtěl undo použít pouze jeden klient, vrátil by se stav i ostatním uživatelům, které si undo vůbec nepřeje. Pro potřeby undo jednoho klienta by tak musely být provedeny mnohé úpravy, a ty by byly neúměrně náročné vůči výsledku. Z důvodu robustnosti tedy tuto variantu taktéž zavrhneme.

5. **Přímá podpora undo v API:** Backendové metody by nativně podporovaly možnost zavolat na určité akce undo: u vybraných funkcí by byla příbuzná metoda, která by jako parametr přijímala identifikátor zaslaný jako výsledek předchozí akce. Jednotlivé požadavky by tak nebyly nijak zdržovány a vše by se provádělo ihned, a požadavek na vrácení změn by byl zpracován jako zcela nová akce.

Zhodnocení řešení: Mezi výhody posledního z návrhů patří zejména to, že při něm není potřeba řešit konzistenci dat mezi více klienty, vše je ihned potvrzeno a změny se řeší nezávisle na předchozím požadavku. Nevýhodou je, že to není obecné řešení - je potřeba tyto metody implementovat pro všechny akce, které mají undo podporovat. Opět se ale jedná o reálně použitelné řešení, které bude implementováno pouze tam, kde to dává smysl.

Po zhodnocení všech variant jsem se rozhodl jít prozatím cestou pátého návrhu - přímé podpory pouze u vybraných akcí. Důvodem je i fakt, že ve většině akcí by v systému ani chybu jít udělat *nemělo* - tím jsme se posunuli od potvrzovacího dialogu, přes undo, až po *prevenci chyb*. Například u vyskladnění vidí skladník jasný seznam položek, které má vyskladnit, a pokud nemá vše *napíáno*, systému mu ani úkol dokončit nedovolí. U některých vybraných akcí půjde undo provést, a u zbylých, které vyhodnotím jako důležité a málo časté, kde nepůjde realizovat ani prevence chyb, ani undo, skončím u běžných potvrzovacích dialogů.

Realizace tohoto návrhu je popsána v sekci 4.8.

3.1.2 Zkratky v systému za použití čtečky čárových kódů

Nápad na tuto funkci vychází z faktu, že drtivá většina skladníků, kteří budou se systémem pracovat, budou mít v ruce zařízení podporující skenování položek a okamžité předávání načteného kódu přímo do aplikace.

Návrhem k realizaci tedy je, že by skladník nemusel načítat pouze EANy položek a čárové kódy umístění, ale mohl by přes čtečku iniciovat i určité akce - například v místě, kde přechází k přebírání dorazivších dodávek zboží by mohl být nalepen kód, který v aplikaci spustí úlohu *Příjem dodávky*. V místě, kde se vyzvedávají palety, vozíky či cokoliv jiného, co čeho se kompletuje vyskladnění, by zase mohl být kód pro iniciování úlohy vyskladnění - a pokud by vyskladnění nemohl iniciovat skladník sám, tak by se například mohl otevřít seznam vyskladnění, která čekají na vyřízení.

Konkrétní seznam kódů a akcí, které se mají stát, by ideálně měl být konfigurovatelný přímo ze systému, aby byl skladový systém použitelný pro různé typy skladů.

3.2 Návrh procesů v novém skladovém systému

Přes započítání samotné implementace jsme společně s Pavlem Kovářem vytvořili dokument popisující procesy rolí skladníka a vedoucího skladu.

Dokument je rozdělen na následující části:

1. Slovník pojmů,
2. Entity a jejich atributy z pohledu uživatele,
3. Procesy shodné pro obě role,
4. Procesy role Skladník,
5. Procesy role Vedoucí,
6. Angler (autorizační server)

Z důvodu, že má dokument deset stran a obsahuje formátování napomáhající porozumění, nebudu ho vkládat přímo do textu, ale je možné jej nalézt jako přílohu D nebo v souboru `processes.pdf` na přiloženém médiu.

3.3 Volba technologie

Jelikož bude aplikace rozdělena na backend, kterým se zabývá můj kolega Bc. Pavel Kovář, a frontend, který je předmětem této práce, je žádoucí věnovat jistou část textu volbě vhodné technologie.

Cílová platforma Aplikace je navrhována s ohledem na hardwarové vybavení skladu, ve kterém bude poprvé nasazována: zdejší skladníci jsou vybaveni mobilními telefony *Zebra TC25Bj*, které disponují OS Android 7.1 a vestavěnou čtečkou čárových kódů. Kromě skladníků by měla být aplikace použitelná také z tabletu či stolního počítače pro účely vedoucích pracovníků. Z důvodu jednoduchosti vývoje, testování, možností aktualizací a obecně dobré zkušenosti z jiných projektů bylo hned při úvodním návrhu určeno, že aplikace bude tvořena formou webové služby, která bude na klientských zařízeních zobrazována ve WebView v jednoduchém kontejneru chovajícím se jako nativní aplikace. Řídící pracovníci budou naopak moci využít přístupu odkudkoliv, kde budou připojeni k internetu pouze pomocí běžného webového prohlížeče. Z tohoto důvodu jsou v následující rešerši zhodnocovány frameworky či knihovny, které usnadňují vývoj *webových aplikací*.

Frameworky a knihovny V době psaní této práce patří mezi nejpopulárnější [15] [21] front-endové frameworky či knihovny Angular [3], React [51], Vue.js [72], Ember.js [18] a Backbone.js [8].

Názvosloví Pro účely tohoto textu budu na následujících řádcích používat slovo *framework*, kterým budu označovat jak frameworky, tak knihovny, z důvodu snížení opakování textu.

3.3.1 Datum vydání

Zatímco v současnosti nejčastěji porovnávanými frameworky jsou první dva zmíněné, Vue.js je z této pětice vybraných nejmladší, nabírá ale velké obliby. Ember.js a Backbone.js jsou poté lehce upozaděny z důvodu jejich stáří. Přehled prvního vydání jednotlivých frameworků je v tabulce 3.1

	Angular	React	Vue.js	Ember.js	Backbone.js
Vydání první verze	2010/2016 ^a	2013	2014	2011	2010

Tabulka 3.1: Volba frameworku: Datum vydání

^aV roce 2010 byl vydán AngularJS, který byl v roce 2016 kompletně přepsán do TypeScriptu a vydán jako Angular 2, či jednoduše *Angular*.

Datum vydání ovšem nelze objektivně ohodnotit bodovým ziskem. Na jedné straně stojí fakt, že starší framework může být vyspělejší a tudíž stabilnější atp., na straně druhé nové frameworky se často učí z chyb provedených jejich předchůdci a vyberou z nich pouze to nejlepší. Tato tabulka tedy zůstane čistě přehledová.

3.3.2 Zázemí

Zatímco Angular a React jsou vyvíjeny velkými společnostmi: Googlem, respektive Facebookem, které zná každý, Ember.js je vyvíjen společností Tilde Inc. [62], která také není žádným startupem. Vue.js a Backbone.js by se naopak daly nazvat *komunitními projekty*, neboť jsou vytvořeny převážně jedním autorem (Evan You, respektive Jeremy Ashkenas) a rozvíjeny a udržovány komunitou vývojářů.

Na první pohled by se mohlo zdát, že z tohoto hodnocení budou vycházet lépe ty frameworky, které mají za sebou stabilní firmy, neboť je tím zajištěn jejich kontinuální vývoj. Ve skutečnosti ale velké firmy *zabíjejí* své projekty poměrně často, stačí se podívat například na seznam projektů, které ukončil Google [46]. Oproti tomu komunitní projekty mohou žít dále i v případě, že jejich hlavní autor už na projektu nechce, nebo nemůže pracovat. Z toho důvodu nelze jednoznačně určit, které zázemí je pro budoucnost frameworku výhodnější, a u tabulky 3.2 se tedy opět zdržuji udělování bodů.

	Angular	React	Vue.js	Ember.js	Backbone.js
Zázemí velké společnosti	ano	ano	ne	částečně	ne

Tabulka 3.2: Volba frameworku: Zázemí

3.3.3 Licence

Licence k použití frameworku je důležitá položka při rozhodování. Naštěstí všech 5 porovnávaných frameworků je v době psaní této licencováno pod MIT licenci, která povoluje jakékoliv použití i v komerční sféře, úpravy, distribuce i použití v ne-opensource projektech. Nevýhodou této licence je nulová záruka funkčnosti či zodpovědnost autorů za potenciální spáchané škody tímto softwarem.

Licencování Reactu Facebook původně vydal svůj React pod BSD licenci spolu s dalšími patenty, avšak 24. září 2017 byl React převeden pod MIT licenci [1, 70].

Jelikož jsou všechny frameworky licencovány stejně, neprobíhá v tabulce 3.3 žádné bodování.

	Angular	React	Vue.js	Ember.js	Backbone.js
Licence	MIT	MIT	MIT	MIT	MIT

Tabulka 3.3: Volba frameworku: Licence

3.3.4 Křivka učení

Složitost frameworku je důležitá metrika, neboť má dopady zejména na ekonomickou stránku projektu. Jednoduché prvotní vniknutí do problematiky frameworku ovšem také nemusí být nutně výhodou, pokud v něm je později problémové provést některé pokročilé věci, nebo i v pokročilém stádiu zdržuje svým nízkoúrovňovým přístupem k problémům, které jiné frameworky řeší automaticky.

Angular, React a Vue.js Přehled obtížnosti tří v současnosti nejčastěji skloňovaných frameworků přehledně shrnul Rajdeep Chandra ve své prezentaci *My experience with Angular 2, React and Vue* [23], ze které vychází hodnocení v tabulce 3.4.

Ember.js Tento framework je dle V. Lascika [38] vhodný spíše pro projekty, na kterých pracuje velké množství vývojářů, a to z důvodu své komplexnosti. Proto jej pro použití v mé vznikající aplikaci hodnotím nula body.

Backbone.js U této knihovny je důležité zmínit, že umožňuje vývojáři vytvořit si strukturu aplikace kompletně dle svého uvážení [2]. To s sebou může nést jak výhody pro zkušeného, tak nevýhody pro nezkušeného vývojáře, který v pokročilém stádiu vývoje může zjistit, že některou ze základních struktur navrhl špatně. Samotná obtížnost práce s touto knihovnou je ale poměrně nízká.

	Angular	React	Vue.js	Ember.js	Backbone.js
Obtížnost	vysoká	vyšší	nízká	velmi vysoká ^a	nízká
<i>bodový zisk</i>	1	2	4	1	4

Tabulka 3.4: Volba frameworku: Obtížnost

^aza předpokladu, že na projektu bude pracovat pouze velmi malé množství vývojářů

3.3.5 Oficiální dokumentace

Hlavním zdrojem ke studiu frameworku by měla být jeho oficiální dokumentace, v této sekci tedy budu hodnotit kvalitu a obsáhlost oficiálního manuálu k jednotlivým frameworkům.

Angular Jedná se o velmi obsáhlou a dobře rozdělenou dokumentaci [5], která obsahuje i řadu příkladů a ve srozumitelné stromové struktuře vývojář jednoduše najde, co potřebuje.

React Dokumentace Reactu [20] je o poznání jednodušší než ta Angularu, avšak to je způsobeno tím, že React je pouze knihovna, kdežto Angular je plnohodnotný framework. Dokumentace je rozdělena na jednodušší úvod a pokročilejší techniky, je tedy snadné s ní pracovat.

Vue.js Nejmladší z frameworků má také velmi přátelskou dokumentaci [27], která je podobně jako u Angularu velmi bohatá a stromově strukturovaná.

Ember.js Oficiální manuál Ember.js [19] je taktéž poměrně obsáhlý a strukturovou připomíná dokumentaci Angularu a Vue.js. Obsahuje velké množství ukázek kódu a je logicky strukturován.

Backbone.js Poslední ze zkoumaných frameworků má oficiální dokumentaci [9] na první pohled méně atraktivní a pro nováčka může být matoucí. Oproti ostatním dokumentacím chybí například barevné zvýraznění důležitých bodů a další grafické strukturování textu.

	Angular	React	Vue.js	Ember.js	Backbone.js
Kvalita oficiální dokumentace	velmi vysoká	vysoká	velmi vysoká	velmi vysoká	střední
<i>bodový zisk</i>	3	2	3	3	1

Tabulka 3.5: Volba frameworku: Dokumentace

3.3.6 Testování

Testování je při vývoji softwaru velkým tématem. Mnoho vývojářů nerado testuje, jiní se naopak specializují pouze na testování. V moderních aplikacích je testování z větší části řešeno automatizovanými testy, které se spouští v rámci CI. Přestože se tato práce zabývá pouze frontendem, i ten je možné testovat již od úrovně Unit testů, vyvíjej jej pomocí TDD a nakonec samozřejmě vše zkontrolovat pomocí E2E testů.

Angular Nejkomplexější z frameworků nabízí přehlednou dokumentaci [4], jak by měl být testován. Popisuje jak používat Unit testy i E2E testy a pro druhý zmíněný typ nabízí i samotný testovací framework: Protractor [50]. Jeho testovatelnost tedy hodnotím jako velmi dobrou.

React Tato knihovna ve své dokumentaci na první pohled testování příliš neprosazuje, avšak stránku dedikovanou této kratochvíli [61] lze nakonec také nalézt. Narozdíl od Angularu zde není poskytován dedikovaný framework určený přímo pro testování této knihovny. Stránka popisuje testovací frameworky, které používají různé společnosti: autoři Reactu - Facebook - zmiňují svůj Jest [34], také je ale zmiňován Enzyme [25] od Airbnb. Konkrétní příklady použití bychom tedy dále hledali v dokumentacích těchto frameworků. Celkově je pro

React k dispozici více různých testovacích frameworků, které opět pokrývají jak Unit, tak E2E testování.

Vue.js Ani Vue.js se neztratí co se Unit a E2E testů týká. Přímo ve své dokumentaci [63] popisuje spouštění Unit testů pomocí vestavěných příkazů, které interně používají buďto již zmiňovaný Jest nebo Mocha [40], a dále odkazuje na vlastní kompletní testovací knihovnu [29]. Také je možné použít velké množství různých testových frameworků a aplikace ve Vue.js lze vyvíjet i pomocí TDD [37].

Ember.js Tento framework se ve své dokumentaci věnuje testování poměrně intenzivně a zasvětil mu rovnou několik stránek [26]. Jako výchozí testovací framework představuje vlastní QUnit, avšak informuje, že je možné použít i frameworky třetích stran. Testování přehledně rozděluje na *Unit*, *Container*, *Render*, a *Application* testy a také popisuje jak testovat jednotlivé komponenty. Testování Ember.js tedy také hodnotím jako velmi dobré.

Backbone.js U posledního zkoumaného frameworku na první pohled není testování moc jasné. Oficiální stránka odkazuje na *test suite* [7], což je ovšem stránka, která testy *spouští* a ne popisuje. Zdá se, že na této stránce je možné otestovat svůj prohlížeč, zda podporuje veškeré funkcionality Backbone.js. Co se psaní testů týká, rozumněji již vypadá externí stránka *Backbone.js Testing* [53], která již zmiňuje i zde dříve probírané frameworky jako *Mocha*. Ve výsledku tak bude pravděpodobně testování Backbone vypadat obdobně jako u ostatních frameworků, avšak přístup oficiální dokumentace snižuje jeho pochopení.

	Angular	React	Vue.js	Ember.js	Backbone.js
Možnosti testování a jejich popis v dokumentaci	velmi kvalitní	velmi kvalitní	velmi kvalitní	velmi kvalitní	matoucí
<i>bodový zisk</i>	2	2	2	2	1

Tabulka 3.6: Volba frameworku: Testování

3.3.7 Vývojářské nástroje

Dalším důležitým nástrojem při práci s frameworkem je možnost jeho debuggování. Framework by měl nabízet vlastní řešení, které vývojáři usnadní nalézt chybu, zjistit, jak se jeho kód chová či odladit rychlostní problémy.

Všechny ze zde porovnávaných frameworků nabízejí tyto nástroje formou doplňku do prohlížeče, konkrétně se dále budeme bavit o doplňcích pro Google Chrome.

	Angular	React	Vue.js	Ember.js	Backbone.js
Název	Augury	React Developer Tools	Vue.js devtools	Ember inspector	Backbone Debugger
Počet stažení ^a	230k	1.351k	706k	57k	9,5k
<i>bodový zisk</i>	2	4	3	1	0
Hodnocení (z 5)	3.9	4.2	4.7	4.8	4,5
<i>bodový zisk</i>	1	2	3	4	2

Tabulka 3.7: Volba frameworku: Devtools

^aStav k 24. 12. 2018

3.3.8 Počet hvězdiček na GitHubu

Počet hvězdiček na GitHubu lze velmi volně interpretovat jako oblíbenost frameworku mezi vývojáři. Z tohoto důvodu již v tabulce 3.8 hodnotím frameworky dle počtu získaných hvězdiček.

	Angular	React	Vue.js	Ember.js	Backbone.js
Počet hvězdiček na GitHubu ^a	43,6k	117,7k	122,3k	20,3k	27,3k
<i>bodový zisk^b</i>	2	4	5	0	1

Tabulka 3.8: Volba frameworku: Počet hvězdiček na GitHubu

^aStav k 17. 12. 2018

^bHodnocení přeskakuje bodový zisk 3, aby bylo zhodnoceno i absolutní množství hvězdiček, nejen pořadí.

3.3.9 Počet npm balíčků

Npm [12] je repozitář javascriptových komponent, na kterém jsou sdíleny jednak kompletní řešení (jako například Angular, React, Vue.js a další), ale především různé rozšiřující pluginy do těchto frameworků. Z toho důvodu budu v následující metrice hodnotit, kolik balíčků npm nabízí pro jednotlivé porovnávané frameworky.

Bodové zisky hrubě odpovídají relativnímu počtu nalezených balíčků.

	Angular	React	Vue.js	Ember.js	Backbone.js
Počet npm balíčků	26,6k	73,4k	20,6k	6,4k	1,5k
<i>bodový zisk</i>	<i>2</i>	<i>3</i>	<i>2</i>	<i>1</i>	<i>0</i>

Tabulka 3.9: Volba frameworku: Počet npm balíčků

3.3.10 Otázky na Stack Overflow

Stack Overflow je jedním z portálů sítě Stack Exchange, který zná prakticky každý vývojář. Kdokoliv zde může položit otázku a komunita poté odpovídá, zatímco hlasuje o kvalitě odpovědí, aby byla vybrána ta nejlepší.

Z pohledu volby frameworku může být na jednu stranu vhodné, aby bylo na této stránce hodně otázek týkajících se dané technologie, na druhou stranu to ale může znamenat i nekvalitní dokumentaci. Jelikož ale v předchozí sekci nebyla žádná dokumentace vyhodnocena jako vysloveně špatná, budu dále usuzovat, že větší množství otázek je lepší.

	Angular	React	Vue.js	Ember.js	Backbone.js
Počet otázek na Stack Overflow	146k	118k	28k	23k	21k
Počet <i>zodpovězených</i> otázek	86k	72k	18k	17k	16k
<i>bodový zisk</i>	<i>3</i>	<i>3</i>	<i>1</i>	<i>1</i>	<i>1</i>

Tabulka 3.10: Volba frameworku: Otázky na Stack Overflow

3.3.11 Firemní stack

Další zvolenou metrikou je, jak daná technologie zapadá do firemní stacku firmy Jagu s.r.o., ve které bude tento nový projekt realizován. Firma se

specializuje především na webové aplikace a middlewary na zakázku [30], a mezi nejpoužívanější technologie patří PHP (Nette, Laravel, Symfony), dále provozuje jeden informační systém postavený na Angularu a nově také menší aplikaci ve Vue.js. Tabulka 3.11 shrnuje, jak jsou jednotlivé frameworky blízko k tomuto stacku.

	Angular	React	Vue.js	Ember.js	Backbone.js
Shoda s firemním stackem	ano	ne	ano	ne	ne
<i>bodový zisk</i>	2	0	2	0	0

Tabulka 3.11: Volba frameworku: Shoda s firemním stackem

3.3.12 Dostupnost vývojářů

Metrikou, kterou z hlediska udržitelnosti projektu a jeho ekonomických nákladů nelze opomenout, je dostupnost a cena vývojářů se zájmem o danou technologii.

Tato data se ale obtížněji získávají, většina statistik hovoří o nabídkách práce v dané technologii, nikoliv o počtu lidí, kteří s ní pracují. Z toho důvodu jsem se rozhodl založit tuto metriku na výsledcích vyhledávání osob v profesní síti LinkedIn - tak dokážeme zjistit alespoň hrubý počet lidí, kteří o sobě sami tvrdí, že jsou vývojáři v daném frameworku.

Bodové zisky zde hrubě reflektují relativní počet nalezených profilů.

	Angular	React	Vue.js	Ember.js	Backbone.js
Počet výsledků na dotaz "<název> developer"	344k	333k	78k	21k	76k
<i>bodový zisk</i>	4	4	2	1	2

Tabulka 3.12: Volba frameworku: Počet vývojářů na LinkedIn

3.3.13 Integrace se Sentry

Sentry [56] je nástroj sloužící k automatickému i manuálnímu záznamu chyb v aplikacích. Ve firmě Jagu s.r.o. je využíván v řadě projektů a jeho nasazení bude vhodné i pro aplikaci řešenou v rámci této práce. Z toho důvodu je vhodné

se podívat, jak hlubokou integraci je možné mezi jednotlivými frameworky a Sentry realizovat.

Při pohledu na přehled toho, jaké technologie Sentry podporuje v JavaScriptu [32] rychle zjišťujeme, že všech pět zde zkoumaných frameworků je oficiálně podporováno, včetně rychlého návodu na zprovoznění. Z toho důvodu neprobíhá v tabulce 3.13 žádné bodování.

	Angular	React	Vue.js	Ember.js	Backbone.js
Oficiální integrace se Sentry	ano	ano	ano	ano	ano

Tabulka 3.13: Volba frameworku: Integrace se Sentry

3.3.14 Souhrn průzkumu

V tabulce 3.14 jsou sečteny body z předchozích dílčích hodnocení.

	Angular	React	Vue.js	Ember.js	Backbone.js
bodový zisk celkem	22	26	27	14	12

Tabulka 3.14: Volba frameworku: Výsledky

Výsledky rozdělují frameworky na dvě skupiny. V té vedoucí je trojice Angular, React a Vue.js, v pozadí poté zůstávají Ember.js a Backbone.js. Na tomto místě je také vhodné znovu zmínit, že hodnocení frameworků probíhalo ve vztahu ke konkrétnímu projektu, který je cílem této práce, a také ve vztahu k firmě Jagu s.r.o., která vývoj této aplikace zaštiťuje.

První tři frameworky jsou seřazeny poměrně těsně za sebou, avšak nejlépe vyšel ze srovnání nejmladší Vue.js, který tímto volím jako framework, ve kterém budu na práci dále pracovat.

Implementace

V této kapitole se budu věnovat převážně některým obecným součástem aplikace a jak jsem došel k jejich výsledné podobě. Obsahem *není* popis realizace funkčních požadavků aplikace, přestože jsem jejich implementaci strávil nezanedbatelné množství času. Tento popis je již totiž dostupný ve formě analýzy a návrhu, kterým odpovídá.

4.1 Nízkoúrovňové vývojové nástroje

4.1.1 Webpack

Webpack [68] je software, který zpracovává součásti webových aplikací a tvoří z nich balíčky vhodné pro webové prohlížeče. Primárně je zaměřen na JavaScript, ale dokáže zpracovávat i řadu dalších formátů, přes styly v css, sass či stylus, obrázky v png, jpg, svg, ale také konfigurace v json, yaml a dalších.

Hlavním důvodem, proč používat Webpack, je možnost rozdělování kódu do jednotlivých souborů. Při programování je pohodlné mít oddělené ucelené komponenty, různé konfigurační soubory apod. Výstupem webpacku jsou poté typicky pouze čtyři soubory: dva obsahující logiku aplikace (.js) a dva pro stylování (.css), přičemž logika i styly jsou rozděleny na samotnou aplikaci - kód programátora, a kód třetích stran - tzv. vendor.

Použití loaderů. Další příležitost k použití webpacku přichází ve chvíli, kdy chceme v projektu mít kód, který není v cílovém prohlížeči podporován. Tím může být například SASS, nebo třeba i moderní syntaxe Javascriptu řídicí se standardem ES6 či novějším. S pomocí Webpacku lze nastavit, aby se při zpracování některých assetů použil *loader*, který například převede SASS na CSS, konstrukce ES6 na ES5 apod. [31].

Webpack ve spolupráci s Vue.js. Konfigurace webpacku může být pro neznalého uživatele až příliš obsáhlá a může být jednoduché v ní udělat chybu, kvůli které bude například výsledný build aplikace neoptimalizovaný a tudíž pomalý. Vývojáři Vue tento problém mitigovali tak, že celá optimalizovaná konfigurace webpacku je součástí `vue-cli`, což znamená, že stačí mít v projektu jako závislosti právě tento npm modul, a pokud nejsou vyžadovány žádné pokročilé funkce, funguje vše *automagicky*⁸. Pro účely porovnání jsem se podíval, jak dlouhá je konfigurace webpacku, kterou má Vue automaticky v sobě - jedná se o 1490 řádků kódu.

4.1.2 Vue-UI

V předchozím paragrafu jsem popisoval výhodu použití Vue-CLI, ke kterému je ale vhodné říci i několik dalších slov, jelikož se jedná o velmi praktický nástroj, usnadňující mnoho běžných požadavků na správu projektu. Zaměřit bych se chtěl především na možnost spuštění grafického prostředí, pomocí `vue ui` z terminálu, což si vytvoří vlastní webový server a na adrese `localhost:8000` spustí grafickou správu projektu. Ta obsahuje následující možnosti:

- *přehled projektů*, který nabízí rychlou kontrolu dostupných aktualizací použitých balíčků, kontrolu bezpečnostních mezer nebo RSS čtečku novinek od Vue komunity,
- *instalované pluginy*, což jsou běžné npm balíčky, ke kterým je ale možné přidat další konfiguraci. Jedný se například o pluginy pro samotné `vue-cli`, `eslint` nebo třeba knihovnu `vuetify`, která poskytuje grafické

⁸Nejedná se o překlep, jde o spojení slov automaticky a magicky

komponenty, a pomocí tohoto pluginu vkládá do výsledného buildu pouze ty, které jsou skutečně v aplikaci používány.

- *instalované závislosti*, u kterých je možné jednoduše hromadně aktualizovat na nové verze, zobrazovat, jakou verzi používáme a jaká je nejnovější, nebo přes jednotné hledání instalovat nové závislosti,
- *úlohy spouštěné nad kódem aplikace*, asi nejpraktičtější součást Vue-UI, ze které je možné spouštět lokální dev-server ale také build pro produkci. Největší výhodou spouštění těchto úloh z UI je ta, že po skončení úlohy jsou zde dostupné bohaté informace o velikosti výsledných souborů a přehled, jak velká je která závislost. Jako příklad uvedu dále optimalizaci velikosti překladů.

Optimalizace velikosti aplikace odebráním zbytečných překladů. Jednou ze závislostí, kterou v projektu používám, je knihovna *moment* [41], která se stará o parsování a zobrazování datumů a časů, a jejíž součástí jsou i překlady do 127 jazyků. Jeden soubor s jazykem má sice „pouze“ 172 řádků kódu, ale při vzájemném pronásobení to už je *21844 řádků*. Není tak divu, že při analýze velikosti výsledných souborů pomocí Vue-UI jsem odhalil, že tyto překlady zabírají téměř *15% celé velikosti* souboru s kódem třetích stran (vendor), jak je znázorněno na obrázku 4.1.

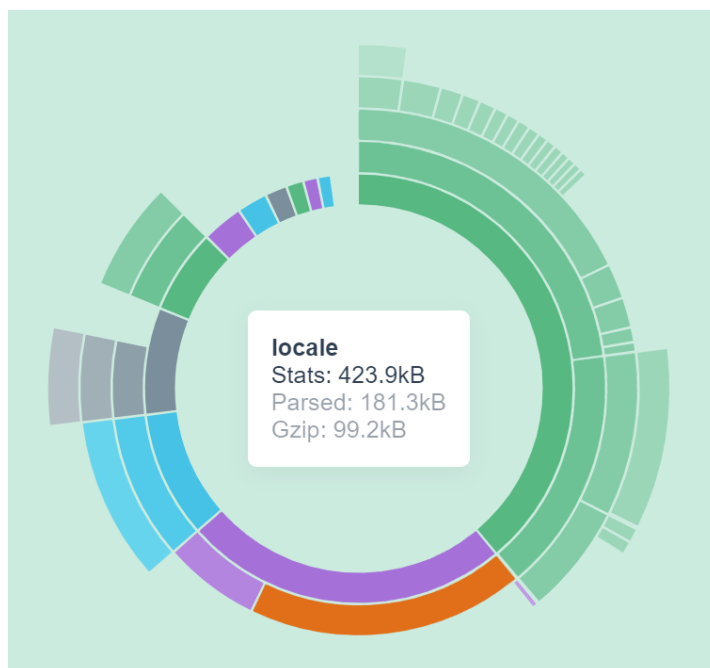
Našel jsem si tedy řešení spočívající v přidání jednoho řádku konfigurace [35], jehož cílem je nezahrnovat do výsledného buildu ty jazyky, které nepoužívám (což jsou všechny, kromě češtiny a angličtiny). Výsledek je takový, že nyní překlady nezabírají *ani jedno celé procento*.

4.1.3 Proměnné prostředí

TODO

4.2 Router

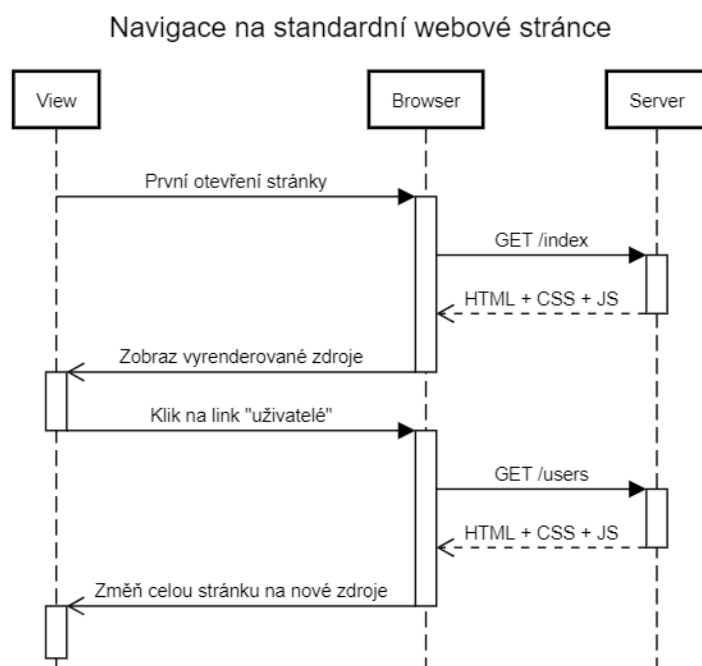
Jednou z prvních komponent, kterou jsem k čistému Vue.js přidal, byl oficiální Vue Router [28]. Koncept routeru je dobře znám z jakékoliv jiné aplikace, která nějak pracuje s prohlížečem uživatele.



Obrázek 4.1: Analýza velikosti výsledné aplikace: překlady moment.js (na obrázku popsáno jako locale a označeno oranžově) zabírají velké procento z celkové velikosti souboru.

Rozdíl Vue.js routeru oproti tomu, který je používán třeba v PHP, spočívá v dynamičnosti celého Javascriptového frameworku: oproti standardním webovým stránkám, kde přesměrování na novou adresu skutečně vyvolá požadavek na server, který odpoví obsahem stránky na dané URL (diagram 4.2), zde se vše děje dynamicky přímo na klientském zařízení, obsah je vyměněn pomocí Javascriptu a hodnota adresního řádku se změní pouze „pro efekt“ - tj. aby uživatel věděl, kde se nachází, a aby mohl adresu zkopírovat a sdílet. Samozřejmě i zde většinou dochází k načítání dat nově otevřené stránky, ale vše je typicky asynchronní - stránka se změní ihned a teprve poté jsou do ní doplněna data - viz diagram 4.3.

Titulky stránek. Lehce matoucí může u tohoto *frontendového routování* být nastavování titulků stránek, tedy `<title>` tagů v HTML. Celá aplikace má totiž stále pouze jeden `<title>` zapsaný v souboru `index.html`, který se sám o sobě při navigaci za pomoci routeru vůbec nemění. K vyřešení tohoto problému sice ve Vue Routeru neexistuje nativní podpora, ale řešení je opravdu snadné - je



Obrázek 4.2: Navigace na standardní webové stránce: Pro každý link existuje cesta na serveru, který odpovídá kompletním obsahem této stránky.

```

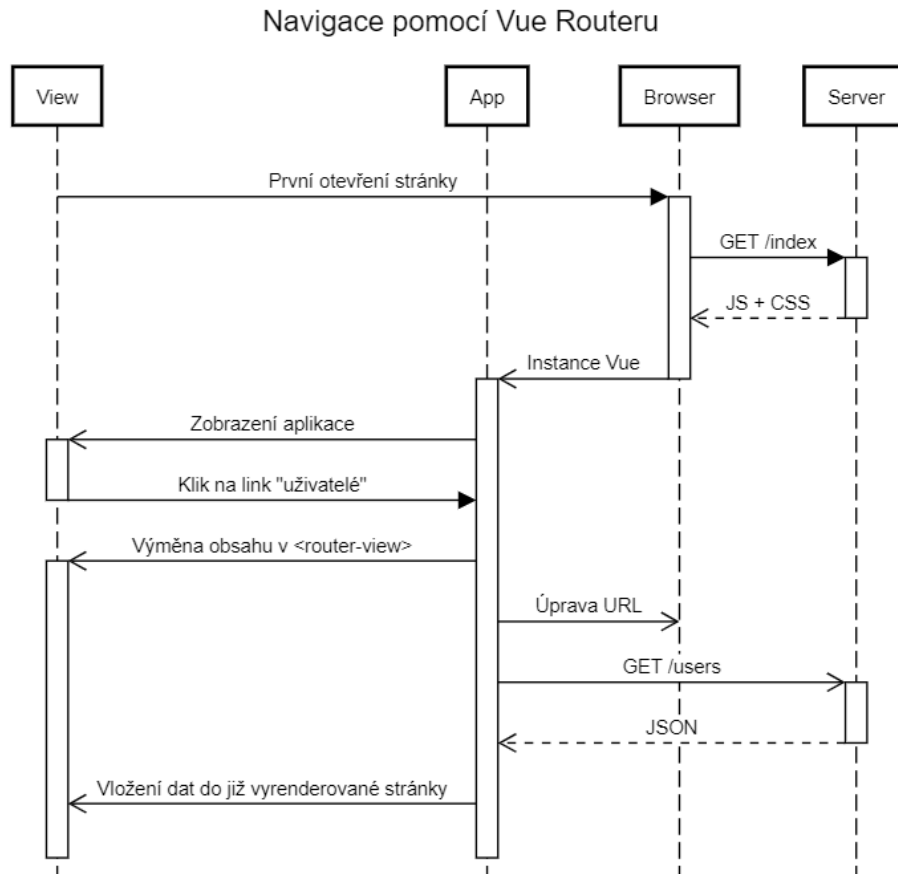
1 {
2   path: '/',
3   component: Homepage,
4   meta: {
5     title: 'Swordfish'
6   }
7 }

```

Ukázka kódu 4.1: Nastavování titulků stránek pomocí Vue routeru - úprava definic

popsáno v jednom z *issues* v repozitáři na GitHubu [71] a spočívá v doplnění titulků do *meta* atributů cest, viz ukázka kódu 4.1, a dále úpravě samotné instance routeru, viz ukázka kódu 4.2.

Toto nastavování titulků stránek jsem nakonec ještě více upravil po svém, jelikož jsem chtěl mít v titulcích i konkrétní hodnoty, které se zobrazují na dané stránce - například při úpravě skladu *Centrální sklad Praha* nechci mít v titulku „Úprava skladu“, ale „Úprava skladu 'Centrální sklad Praha'“. Jelikož jsou ale tyto informace načítány dynamicky až po vstupu na danou stránku



Obrázek 4.3: Navigace v aplikaci pomocí Vue Router: Kompletní obsah aplikace se ze serveru stáhne pouze při prvním přístupu, navigaci v aplikaci řeší Vue Router přímo na klientu a od serveru či jiného API vyžaduje pouze konkrétní data, která v aplikaci zobrazuje.

```
1 router.beforeEach((to, from, next) => {
2   document.title = to.meta.title;
3   next();
4 })
```

Ukázka kódu 4.2: Nastavování titulků stránek pomocí Vue routeru - úprava instance routeru

(URL stránky může být například `/stocks/12/edit` - z té to tedy vyčíst nelze), je nutné i titulek nastavovat dynamicky. Zachoval jsem tedy jako výchozí hodnotu titulku stránky ten, který je nastaven přímo v definici cesty v routeru (ukázka kódu 4.1), avšak zevnitř komponenty je možné pomocí jednotné funkce titulek obohatit dalšími informacemi - jako například názvem upravované položky.

Drobečková navigace. Další úpravou routeru, kterou jsem do jisté míry realizoval po svém, je generování drobečkové navigace. Vue Router ve spojení s Vuetify nemají nativní podporu pro zjišťování rodičovských stránek, a tak jsem si lehkou úpravou a vlastní komponentou pro vykreslování *Breadcrumbs* toto zautomatizoval:

Každá definovaná routa má nastaveného rodiče, ke kterému poskytuje *getter*. Ve vykreslování drobečkové navigace se poté na aktuální routě zjišťují rekurzivně její rodiče, a včetně jejich parametrů se z nich zpětně tvoří celý strom navigace. Co se na první pohled může zdát jednoduché, se zkomplikuje ve chvíli, kdy chceme mít v cestě parametrizovanou stránku.

Například v URL `/stocks/1/locations/12/update` je potřeba v definici routy nahradit všechny identifikátory jejich reálnými hodnotami, které ale naštěstí Vue Router poskytuje i během runtime. Funkce pro vytvoření jednoho odkazu, ze kterých se skládá drobečková navigace, je znázorněna v ukázce kódu 4.3 - jejími argumenty jsou definice cesty v routě a objekt s aktuálními hodnotami parametrů.

```

1 function buildPath(pathWithPlaceholders, parameters) {
2   let routePath = pathWithPlaceholders.replace(/\([^/]+\)/g, '');
3   const matches = routePath.match(/(:[a-zA-Z]+\)|(:[a-zA-Z]+\$)/g);
4   if (matches !== null) {
5     for (const match of matches) {
6       const matchParam = match.replace(/\//, '').replace(/:/, '');
7       routePath = routePath
8         .replace(match, parameters[matchParam] + '/');
9     }
10  }
11  return routePath;
12 }
```

Ukázka kódu 4.3: Automatické generování drobečkové navigace z definic Vue Routeru, včetně podpory parametrizovaných zanořených cest

4.3 Plnohodnotná webová aplikace

V kapitole 3.3 jsem zmiňoval, že skladník bude aplikaci používat z nativní Android aplikace, která bude obalovat WebView, a vedoucí skladu si aplikaci otevře v běžném browseru - pro obě použití by níže popisovaná konfigurace nebyla potřeba, avšak pokud by někdo nepotřeboval čtečku čárových kódů - což je jediný důvod, proč skladníci používají jako základ nativní Android aplikaci, je možné si otevřít na mobilu běžnou stránku a použít volbu „Přidat na plochu“. Tím vznikne zástupce, který zobrazuje *favicon* webové stránky a po jehož otevření se opět otevře běžný webový prohlížeč.

Pokud je však na stránce definovaný `manifest.json` pro webové aplikace, může se po otevření tohoto zástupce otevřít stránka v režimu celé obrazovky, a to případně i se skrytými ovládacími prvky - vše tak vypadá, jako kdyby se jednalo o nativní nainstalovanou aplikaci. Základní konfigurace tohoto manifestu je vidět v ukázce kódu 4.4.

```
1 {
2   "name": "Swordfish",
3   "icons": [
4     {
5       "src": "favicon/swordfish-192x192.png",
6       "sizes": "192x192",
7       "type": "image/png"
8     },
9     {
10      "src": "favicon/swordfish-512x512.png",
11      "sizes": "512x512",
12      "type": "image/png"
13    }
14  ],
15  "start_url": "/",
16  "display": "standalone",
17  "orientation": "portrait",
18  "background_color": "#009688"
19 }
```

Ukázka kódu 4.4: Manifest pro webové aplikace

Tento soubor je vlastně také první krok k vytvoření PWA - Progresivní webové aplikace, což znamená takové webové aplikace, která do jisté míry může fungovat i bez připojení k internetu. Využívá k tomu Service Worker API [57],

jehož specifikace je v době psaní tohoto textu stále ve stavu návrhu, přestože vzniká již od května roku 2014 [58].

I přes nedokončenou specifikaci se již PWA běžně používají a i ve skladové aplikaci by bylo vhodné toto API využít, to již ale není předmětem této práce a proto jej v tuto chvíli dále nerozvádím.

4.4 Překlady

Novou aplikaci je dnes vhodné hned od počátku psát jako *multijazyčnou* - jako základ tedy například v češtině a angličtině.

Pro překlady Vue.js aplikací je vhodné použít knihovnu `vue-i18n`⁹ [36], jejíž použití je obdobné, jako známe z jiných jazyků. Zatímco například v PHP se často překlady zapisují do `.yaml` nebo `.json`, což jsou formáty, které sice umožňují například zanořování, ale s dalšími funkcemi už si často neporadí, zde se texty zapisují do běžného `.js` souboru (ukázka kódu 4.5), jehož výstupem je JSON. To znamená, že můžeme používat libovolné konstrukce Javascriptu, od standardního zápisu (klíč `close` ve zmiňované ukázce), přes spojování řetězců a používání konstant (podklíče `user`), po proměnné klíče (podklíče `orderState`), jejichž definice se doplní z číselníku načteného klidně i z jiného souboru. Kreativní autor překladů by jistě našel i prostor pro použití cyklů nebo funkcí, mně ale pro potřeby skladové aplikace stačily znázorněné možnosti.

Překlady a uživatelská nápověda. Jelikož výsledná aplikace obsahuje i stručnou nápovědu, která má zatím pouze textový obsah, využil jsem pro zápis jejího obsahu také formát, který používá `i18n` pro překlady. Tyto texty jsem ale neintegroval do seznamu obecných překladů v aplikaci, ale pracuji s nimi odděleně - pomocí vlastního řešení pro zobrazování nápovědy. Tím mohu všechny texty této „sekce“ načíst hromadně a umožnit v nich například vyhledávání, nebo je vkládám přímo k formulářovým atributům, které dokumentují.

4.5 State management pattern

TODO + moje persistentní řešení

⁹název je zkratka slova *internationalization* - číslovka 18 značí počet přeskočených znaků

```
1  const user = "uživatelé";
2
3  {
4    close: "Zavřít",
5    user: {
6      create: "Vytvořit " + user,
7      update: "Upravit " + user
8    },
9    orderState: {
10      [OrderStateEnum.CREATED]: 'Vytvořená',
11      [OrderStateEnum.SENT]: 'Odeslaná',
12    }
13 }
```

Ukázka kódu 4.5: Definice překladů pro i18n, včetně pokročilých funkcí jako například spojování řetězců nebo využití číselníků.

Jako první se zde sluší říci, co to vlastně *State management pattern* je. Ve Vue.js aplikaci máme typicky velké množství komponent, a ty mezi sebou často potřebují komunikovat. Pěkný příklad je například *snackbar message*, někdy nazývaná také *toast message* či jednoduše *oznámení o provedení akce*. To je komponenta, která musí být dostupná z jakékoliv jiné komponenty systému a vždy se musí zobrazovat na stejném místě a stejně se chovat. Je tedy žádoucí zpřístupnit její *stav* a to tak, aby se při změně stavu něco automaticky stalo, a aby změna stavu byla řízena jistými pravidly.

Vuex. Vuex [69] je knihovna, která implementuje *State management pattern* pro Vue.js. Zařizuje jednotný přístup ke stavům komponent a umožňuje jejich řízené změny. Jeho použití v aplikaci skladového systému je vidět na ukázce kódu 4.6, 4.7 a 4.8, kde řeší zobrazování *snackbar message*.

Kód z ukázky 4.8 se mi ovšem v průběhu psaní aplikace přestal líbit, neboť zobrazování těchto zpráv je velmi častá operace, a jeho vyvolání není v ukázce příliš jednoduché. Proto jsem si nastudoval Mixiny ve Vue.js [[vue-mixins](#)] a vytvořil mixin, který zjednodušuje zasílání zpráv do Snackbaru. Použití v komponentě je znázorněné na ukázce kódu 4.9. Kromě tohoto použití je sice navíc potřeba v komponentě nastavit, aby importovala a používala nově vytvořený mixin, který poskytuje právě metodu `snack()`, avšak, to lze vše provést během programování automaticky za pomoci IDE.


```
1 Vue.use(Vuex);
2
3 new Vue({
4   store: new Vuex.Store({
5     modules: {
6       snackbar: {
7         state: {
8           snack: ''
9         },
10        mutations: {
11          setSnack (state, snack) {
12            state.snack = snack
13          }
14        },
15      }
16    }
17  });
18 });
```

Ukázka kódu 4.6: Vuex pro snackbar-message: definice

```
1 ...
2 export default {
3   name: "Snackbar",
4   data: ...
5   created: function () {
6     this.$store.watch(state => state.snackbar.snack, () => {
7       const msg = this.$store.state.snackbar.snack;
8       if (msg !== '') {
9         this.show = true;
10        this.text = msg;
11        this.$store.commit('setSnack', '');
12      }
13    })
14  }
15 }
```

Ukázka kódu 4.7: Vuex pro snackbar-message: Snackbar komponenta

4.6 Hlídání konektivity

V moderních aplikacích, které veškerá data ukládají na API, je vhodné hlídat dostupnost tohoto API.

Prvním krokem k realizaci této funkce bylo zjistit stav připojení - tedy zda je aplikace online, nebo offline. Jako první jsem našel vlastnost prohlížeče

```
1 this.$store.commit('setSnack', '<message to display>');
```

Ukázka kódu 4.8: Vuex pro snackbar-message: použití z jiné komponenty

```
1 this.snack('<message to display>');
```

Ukázka kódu 4.9: Použití mixinu pro zjednodušení zasílání zpráv do Snackbaru

`window.navigator.onLine` [43], která by přesně o tomto měla informovat a navíc poskytuje i možnosti poslouchat její změny pomocí běžných JS eventů. Po hlubším prozkoumání TODO TODO kontrola konektivity (proč nepoužívat browser properties, vlastní check, jeho náročnost na síť)

4.7 Renderování formulářů

Anti-inspirace v jiném projektu. Ještě před tím, než jsem začal tvořit formuláře ve své diplomové práci, jsem shodou okolností potřeboval upravit několik formulářů v jiné aplikaci, která funguje na podobných technologiích: backend je zcela oddělený a poskytuje REST API, frontend je poté napsán v Angularu. Při zjišťování, jak složité se zde generují formuláře jsem ale zjistil, že pro svou práci chci rozhodně vymyslet lepší systém. Niže příkládám seznam, které věci je potřeba ve zmiňovaném projektu upravit, chce-li programátor přidat nový formulářový prvek:

1. přidat atribut do modelové třídy,
2. nakódovat HTML, které atribut vypisuje,
3. přidat atribut do instance formuláře,
4. nastavovat výchozí obsah formuláře při načtení existujících dat z API,
5. nastavovat nový obsah modelu při ukládání nových dat na API,
6. nakódovat HTML, které umožňuje atribut měnit - tj. formulářový vstup.

Celkem se tedy jedná o šest míst, kam je potřeba nový atribut zanést. Zde je ovšem na místě upozornit, že se rozhodně nejedná o problém Angularu, a že ve Vue.js není vše automaticky jednodušší - postup, jak jsem tento počet redukoval, by měl být použitelný v jakémkoliv Javascriptovém frameworku, a s většími úpravami pravděpodobně i v jiných jazycích.

Nový návrh renderování formulářů. Co se mi zejména ve výše zmiňovaném řešení nelíbilo, byl fakt, že *modelová třída* a *instance formuláře* měly totožné atributy, tudíž se vůbec nemusí nastavovat jedna po druhé, ale můžeme použít například `Object.assign()` [45] pro nakopírování hodnot jednoho objektu do druhého. (*Tato metoda sice není podporována v Android WebView, avšak napsat její ruční alternativu je triviální záležitost*). Tím dokážeme odbourat nutnost nastavování konkrétních klíčů mezi instancí formuláře a modelovou třídou - tedy položky 4. a 5. výše uvedeného seznamu.

Nutnost položky č. 2 - vykreslování v HTML - jsem tušil už od začátku. Tomuto je spíše kontraproduktivní se vyhýbat, neboť typicky každý atribut chceme vypsat nějak jinak, celá stránka je nějak strukturována apod. - tuto položku jsem tedy ponechal a smířil se s tím, že se bude u výpisu formulářů vždy kódovat ručně.

Stále ale ještě máme nutnost nastavit atribut v modelové třídě, v instanci formuláře a formulář nějak vykreslovat - tedy na třech různých místech: dvakrát v Typescriptu a jedenkrát v HTML. Má představa o jednoduše konfigurovatelném formuláři se ubírala směrem k vytvoření pouze jednoho konfiguračního souboru, odkud by se všechny tyto 3 věci generovaly, což se mi nakonec podařilo, a seznam jsem tedy stáhl na:

1. nastavit atribut v konfiguračním souboru,
2. nakódovat HTML, které atribut vypisuje.

Důležité je zde zdůraznit, že jsem odebral i nutnost vytvořit HTML kód formuláře: pokud po formulářovém prvku nejsou vyžadovány žádné nestandardní požadavky, jsou všechny atributy formuláře zpracovávány a vykresleny zcela automaticky pomocí komponenty, kterou jsem pro tento účel vytvořil.

V tuto chvíli je na místě projít ukázkou kódu (4.10), která znázorňuje, jak může vypadat *definice formuláře* pro jednoduché skladové umístění.

```
1  const stockLocationForm = {
2    name: '',
3    code: null
4  };
5
6  const stockLocationFormRender = {
7    name: {
8      icon: 'label',
9      max: 50,
10     required: true
11   },
12   code: {
13     icon: 'line_weight',
14     max: 40,
15     hint: 'stocks.locations.codehint'
16   }
17 };
18
19 export {stockLocationForm, stockLocationFormRender};
```

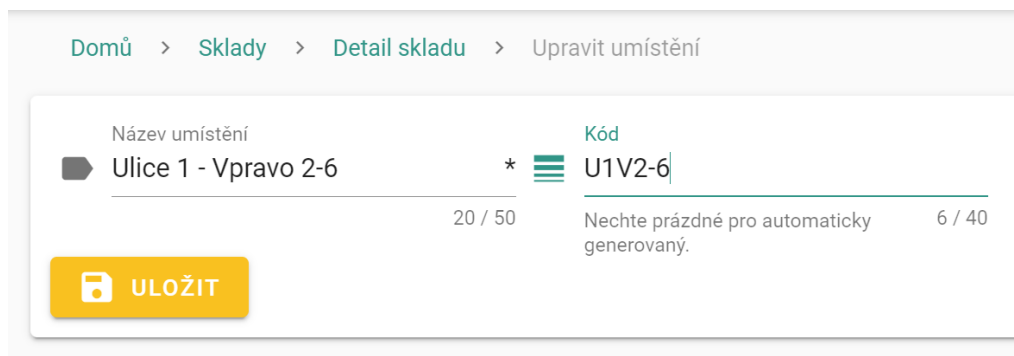
Ukázka kódu 4.10: Příklad definice formuláře: jednoduché skladové umístění

Rozdělení definice na datový a prezenční model. Rozdělení na Form a Render je zvoleno z toho důvodu, aby se oddělila datová (modelová) (Form) a prezenční vrstva (FormRender). Datový model se takto může celý při uložení poslat na API a při načtení se naopak celý přepíše daty z API. Definice zobrazení pak obsahuje informace, jak má samotný formulářový prvek vypadat a chovat se. Někdo by mohl namítnout, že datový model by se také dal generovat automaticky z prezenčního modelu, avšak praxe ukázala, že některé formuláře jsou například rozdělené na více částí co se renderování týká, ale datově jsou společné. Díky rozdělení na data a render pak mohou mít na stránce dvě formulářové komponenty, které sdílí datový model, ale každá má svůj vlastní prezenční model. Někdy se také hodí, že v datovém modelu jsou i prvky, které renderovat nechceme, například hours, do kterého se ukládá strávený čas v úloze, který systém zaznamenává zcela automaticky (viz sekce 4.11). Seznam konfigurovatelných možností pro každý atribut formuláře zahrnuje například:

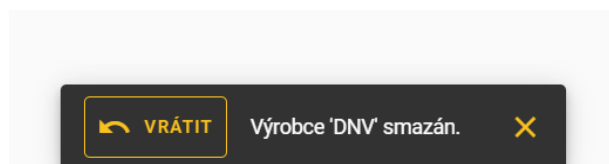
- *label*: název formulářového prvku - pokud není vyplněný, hledá se definice překladu dle názvu klíče atributu,

- *icon*: označení ikony z Material Icons, která se bude zobrazovat vlevo od formulářového prvku,
- *hint*: cesta k překladu textu, který se bude zobrazovat pod formulářovým prvkem,
- *help*: klíč položky nápovědy, která se zobrazí v modálním okně po kliknutí na ikonku otazníku, který se bude zobrazovat vpravo od formulářového prvku,
- *items*: pole s hodnotami, které budou na výběr, jedná li-se o prvek typu `select` nebo `autocomplete`,
- *loading*: booleanovská hodnota, zda má mít prvek načítací stav. Typicky se nenastavuje v konfiguračním souboru, ale může být ovládáno z komponenty, která formulář vykresluje,
- *multiple*: booleanovská hodnota, která určuje, zda prvek typu `select` nebo `autocomplete` může mít více vybraných hodnot současně,
- *readonly*: booleanovská hodnota určující, zda má být prvek pouze pro čtení,
- *rules*: pole pravidel pro validaci prvku (pravidla `max` a `required` je pro zjednodušení možné zadat i napřímo v konfiguraci),
- *createNew*: co se má zobrazit a případně stát, pokud je prvek typu `select` nebo `autocomplete` a vyhledávání přípustných prvků nenalezlo žádnou shodu na uživatelův vstup,
- *autocomplete*: struktura obsahující metody, které se mají zavolat na API, a které následná data zpracují a automaticky tak vytvoří seznam *items*, které budou nabízeny ve formulářovém prvků typu `autocomplete`.

Jedná se tedy o poměrně flexibilní systém, který umí nejen zpracovávat různé typy vstupů, ale také je poměrně bohatě upravovat a přizpůsobovat - pro mé potřeby vykreslování především formulářů na tvorbu či úpravu entit, které se v systému nacházejí: tj. skladů, dodavatelů, produktových karet, odběratelů, umístění apod., a dále formulářů zadávání a schvalování úkolů, které na skladě probíhají, je tato komponenta a její konfigurace naprosto dostačující



Obrázek 4.4: Formulář vykreslený pomocí jednotné formulářové komponenty



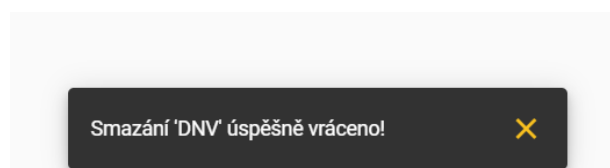
Obrázek 4.5: Undo: Možnost vrácení provedených změn

a do budoucna hlavně velmi snadně rozšiřitelná a konfigurovatelná. Ukázka formuláře vytvořeného přes tuto jednotnou komponentu je vložena přímo v textu jako obrázek 4.4.

4.8 Realizace podpory pro undo

Jak jsme s Pavlem rozhodli v návrhové části (sekce 3.1.1), undo je realizováno přímou podporou u vybraných akcí.

Pro demonstraci celé funkčnosti byla v první fázi vybrána *správa výrobců zboží*, u které je možné procházet kompletní historii změn, a tedy i provádět undo. Základní implementace rozšiřuje běžné *snack messages* přidáním tlačítka „vrátit“ (obrázek 4.5). Takovýto toast message má nastavenou zvýšenou či neomezenou životnost, nezmizí tedy během pár sekund, jako ty ostatní. Po využití funkce vrácení se po úspěšném dokončení zobrazí buďto obrázek 4.6, či případně chybová hláška.



Obrázek 4.6: Undo: Potvrzení vrácení provedených změn

4.9 Generování kódu ze specifikace OpenDoc

Ještě předtím, než jsem začal pracovat na realizaci samotných funkčností aplikace, jsem si připravil rozhraní, přes které budu komunikovat s API.

Při psaní tohoto kódu jsem se snažil co nejvíce omezit opakování kódu, a tak jsem vymyslel i několik funkcí, které kód podstatně zkracují, ale jak jsem zjistil později, tak do něj zanáší neočekávané chování.

Uvedu příklad na funkci pro vytvoření nového odběratele, jejíž hlavička vypadá následovně:

```
create(name, ico, dic, phone, email, website, billing_address, ...
```

Uvnitř těla této funkce potřebuji odeslat na API objekt, který vypadá následovně:

```
{name: <hodnota_argumentu_name>, ico: ...}
```

Na jeho vytvoření samozřejmě existuje jednoduchá cesta: vytvořit objekt postupně ručně, jeden argument po druhém. To je ale zbytečná ruční práce, do které se navíc jednoduše zanesou chyby.

Další možností je vše vytvořit v cyklu, který bude iterovat přes všechny názvy atributů - jenže to znamená že buďto musím znovu udržovat seznam těchto atributů, a nebo využít seznam, který už vlastně *mám vytvořený v deklaraci funkce*.

Parsování názvů argumentů funkce. V Javascriptu není plnohodnotná Reflexe, jak ji známe například z Javy, neboť JS je jazyk *prototypový*. Z toho důvodu není tak snadné vytáhnout si v runtime názvy argumentů funkce. Nejbližší nástroj, který se k tomuto přibližuje, je možné vypsát si textový zápis definice funkce, a z jeho hlavičky argumenty vyparsovat - viz ukázka kódu 4.11, která vychází z kódu Christophera Hermanna [22].

Jak se ale ukázalo později, toto parsování není použitelné v režimu, kdy je aplikace zkompilevaná pro produkční použití, neboť takový kód je minifikovaný

```
1 function getFnArgNames(fn) {  
2   return fn.toString()  
3     .match(/^[^]*\(((\[^\)]*\))\)/)[1]  
4     .split(',')  
5     .map(element => element.split('=')[0].trim())  
6     .filter(element => !(element === ''  
7       || element === null  
8       || typeof element === 'undefined'))  
9   };  
10 }
```

Ukázka kódu 4.11: Parsování názvů argumentů funkce

a tudíž neobsahuje celé názvy argumentů funkce, ale pouze jeden znak - výsledný objekt odesílaný na API tedy vůbec neodpovídá specifikaci, a požadavek logicky selže.

Modely datových typů. Metody parsování argumentů funkce by vůbec nebyly potřeba, kdybych používal standardní modely objektů - tedy předpřipravené struktury - které budou mít potřebné klíče v objektech definované. Tím jsem se ale otočil do kruhu, že nechci nic opisovat a duplikovat. V tu chvíli se ale objevila myšlenka, že když mám od kolegy Pavla připravené krásně zdokumentované API, že bych mohl modely automaticky generovat přímo z něj.

Generování modelů datových typů. Dokumentaci API, kterou mám dostupnou, je zpracovaná ve standardu OpenAPI 3.0 [47], je tedy možné ji zpracovávat automatizovaně různými nástroji, které stručně popíšu:

- *swagger-codegen* [60] Jedná se o velmi robustní nástroj, který dokáže generovat nejen modely, ale také umí vygenerovat kompletní třídy pro komunikace s API. Jeho detaily rozeberu níže.
- *swagger-js-codegen* [13] Tato knihovna třetí strany má podobné možnosti jako *swagger-codegen*, avšak v době psaní této práce není aktualizovaná pro OpenApi specifikace třetí verze, tudíž pro mé potřeby nevhodná.
- *swagger-js* [33] Doplnující nástroj k *swagger-codegen*, který umožňuje veškeré činnosti provádět přímo za běhu programu pomocí Javascriptu.

Pomocí jednoduchého zápisu je například možné i metody spouštět a testovat, avšak pro mé potřeby *generování modelů* je použití taktéž nevhodné.

- *OpenAPI Generator* [59] Na rozdíl od předchozích - víceméně samostatných - nástrojů se v tomto případě jedná o plugin do WebStorm. Interně používá první zmiňovaný - *swagger-codegen*, takže jejich možnosti popíšu společně na dalších řádcích.

Ze čtyř nástrojů zbyly dva padnoucí v úvahu, jeden z nich ale interně používá ten druhý, takže jejich výstupy jsou shodné.

Vyzkoušel jsem tedy generátor spouštět s různými parametry, avšak výsledný kód byl na můj vkus vždy zbytečně dlouhý a nezapadal do zbytku codestyle mé aplikace. Zde je nutno poznamenat, že kdybych s tímto generátorem začal, tak ho pravděpodobně použiji i včetně možnosti vygenerovat prakticky celou vrstvu, která realizuje integraci Javascriptu s API - neboť generovaný kód rozhodně není nekvalitní, je velmi robustní a ušetří spoustu času. Protože ale již mám napsanou vlastní vrstvu, která se o API stará, a zakomponovat do ní pouze část generovaného kódu je poměrně složité, a nechci ani vlastní vrstvu zahodit (řeší totiž i některé další věci, které z generátoru nevypadnou - jako například překlady, automatické zpracování chybových stavů nebo zobrazování loaderu), rozhodl jsem se nakonec tento generátor nevyužít. Budu-li ale někdy pracovat na dalším projektu, ke kterému bude dostupná dokumentace ve specifikaci OpenAPI, rozhodně tento generátor zkusím použít hned na začátku implementace.

Vyřešení modelových tříd pomocí formulářových modelů. Když jsem se rozhodl nepoužívat nakonec generátor kódu z dokumentace API, musel jsem stále ještě vyřešit problém s tím, že má funkce `getFnArgNames` nefunguje v produkčním režimu. Řešení nakonec bylo snadnější, než se zdálo, neboť už vlastně mám připravené vlastní modely pro všechny formuláře, jak jsem popisoval v sekci 4.7 - jedná se o například o konstantu `stockLocationForm` z ukázky kódu 4.10. Někdo by mohl namítat, že tento model nemá řádně definované typy a další náležitosti, a má zcela pravdu. Ovšem pro základní potřebu *neopisovat ve funkcích zařizujících komunikaci s API* všechny atributy je to více než dostačující - ve funkcích jsem přestal řešit, jaké všechny parametry

se do nich dají poslat, a přijímám jednoduše objekt data - jehož obsahem je právě například `stockLocationForm`. Data, která uživatel zadá do formuláře jsou tedy *na přímo, bez jakéhokoliv kopírování* ve stejné formě (samozřejmě po vyřešení serializace, kódování pro URL či převod času do formátu ISO-8601) i odesílána na API - čímž získáváme i zanedbatelné zvýšení efektivity, byť na úkor bezpečnosti (protože nehlídáme typovost obsahu dat), takže to zase taková výhoda není.

Samozřejmě se najde i několik metod, do kterých nelze poslat takto přímo definici formuláře, protože se jedná například o pokyn k přesunu zboží mezi umístěním atp. - které se generuje z napínavání zboží, nikoliv z vyplněného formuláře. V těchto několika případech jsem ručně vypsál přiřazení (kterému jsem se snažil z počátku za každou cenu vyhnout), avšak nikdy se nejedná o více než 3 atributy, takže přehlednost by neměla trpět. Do budoucna budu ještě zvažovat zavedení větší typovosti argumentu data, který často do API metod posílám, rád bych k tomu využil například JSdoc.

4.10 Barevná paleta aplikace

Barvy použité v aplikaci mají velký vliv na to, jak ji budou její uživatelé vnímat. Když jsem začal s implementací, zvolil jsem pouze pár základních barev, protože v tu chvíli jsem potřeboval funkční kostru aplikace. Postupem času jsem ale začal více přemýšlet o volbě barevné palety a jak ji rozumně zvolit.

Kontrast barvy textu a barvy pozadí. Základem pro čitelnost textu je správný kontrast barvy textu a barvy pozadí, na kterém se text nachází. Například klasický černý (`#000000`) text na bílém (`#FFFFFF`) pozadí má kontrastní poměr 21:1, což je nejvyšší možný. Konsorcium W3 vydalo dokument [67], který popisuje, jak by měl vypadat správně dostupný web - jedna sekce popisuje právě i kontrast. Uvedu úryvek z dokumentu:

- *Kontrast (minimum):* Vizuální prezentace musí mít minimální kontrastní poměr 4,5:1, s následujícími výjimkami:
 - *Velký text:* Musí mít kontrastní poměr minimálně 3:1.
 - *Dekoratивní, skryté a doprovodné prvky a logotypy:* Bez požadavků na minimální kontrastní poměr

- *Kontrast (pokročilý)*: Vizuální prezentace by měla mít minimální kontrastní poměr 7:1, s následujícími výjimkami:
 - *Velký text*: Měl by mít kontrastní poměr minimálně 4,5:1.
 - *Dekoratивní, skryté a doprovodné prvky a logotypy*: Bez požadavků na minimální kontrastní poměr

Zde samozřejmě platí, že čím více chceme text na stránce zdůraznit, tím větší kontrastní poměr mu dáme. TODO kontrola použitých kontrastů v aplikaci.

Volba barev akčních a informačních prvků. U informačních prvků je zde poměrně klasické rozdělení: zelená jako potvrzovací, modrá - informativní, žlutá - varovná a červená - chybová. Pro akční tlačítka pak typicky zelená - vytvoření, modrá - zobrazení, žlutá - úprava a červená - smazání. V aplikaci ale většinou nechceme mít tolik barev, moderní aplikace mají čistší, jednodušší design, který když je proveden správně, může snížit zátěž uživatele tolik barvami a zvýšit přehlednost. Velmi často se dnes objevují definice barev jako primary, secondary (občas i tertiary) a accent, které se jednou zvolí a poté se napříč aplikací používají vlastně jako proměnné.

Výběrem těchto tří barev se zabývá mnoho článků a stránek, teorie, na které ale všichni staví, je pořád stejná: Když volíme tři barvy, existují tři nejčastěji používaná barevná rozložení [10]:

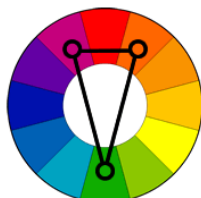
1. *Trojúhelníková*: Rovnoměrné rozdělení barev přes celé barevné schema. Jedná se o velmi výrazné schema, a to i v případě použití nižší barevné saturace. Ideální použití je jedna základní barva a dvě akcentní.



Obrázek 4.7: Barevná paleta aplikace: Trojúhelníkové schéma

2. *Rozděleně-doplňková*: Toto schema vychází z doplňku dvou barev, tedy barev, které jsou v paletě naproti sobě, protější barva je ale rozdělena na

dvě. Oproti trojúhelníkovému schematu je méně výrazné, díky čemuž je jednodušší barvy zvolit tak, aby nebyly pro uživatele příliš útočné. Ideální použití je opět jedna základní barva a dvě akcentní.



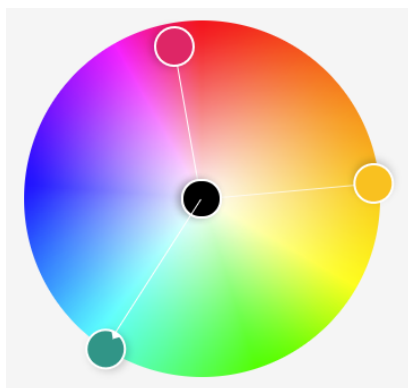
Obrázek 4.8: Barevná paleta aplikace: Rozděleně-doplňkové schema

3. *Analogová:* Zvolené barvy jsou v barevné paletě přímí sousedé a jedná se tak o „klidné“ rozložení, které vytváří jemný design jež je často vidět i v přírodě. Problémem zde může být vhodná volba kontrastu, protože barvy nejsou sami vůči sobě příloš kontrastní a je nutné je od sebe alespoň trochu rozlišit. Ideální použití je jedna dominující barva, druhá doplňková a třetí jako akcentní.



Obrázek 4.9: Barevná paleta aplikace: Analogové schema

Volba barev pro mou aplikaci. Jelikož cílem aplikace je pomáhat při práci ve skladu a upozorňovat na důležité zprávy, vyloučil jsem hned ze začátku *analogové* barevné schema, protože jeho barevné rozložení patří spíše na stránky, které jsou pasivnější, nevyžadují tolik interakce s uživatelem a spíše obsah prezentují, než aby vyžadovali jeho zadávání. Zbyly mi tedy velmi podobná rozložení *trojúhelníkové* a *rozděleně-doplňkové*. Podle osobního pocitu jsem vybral základní klidnou barvu: šedozelenou #009688, a k ní doplňkově-akcentní růžovou #E91A63 a čistě akcentní tmavě žlutou #FFC107, jejichž rozložení na barevné paletě je vidět na obrázku 4.10. Společně tyto barvy tvoří něco mezi dvěma zvolenými schematy - trojúhelník není ani pravidelný, ani se



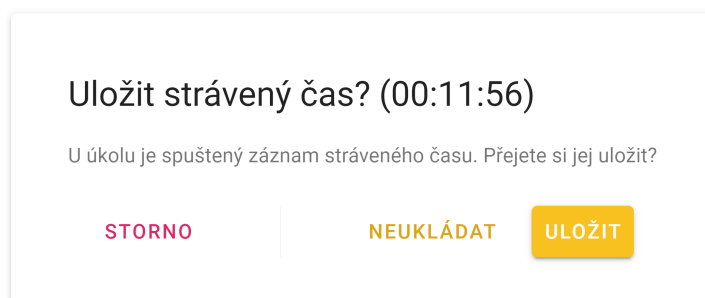
Obrázek 4.10: Barevná paleta aplikace: Zvolené schema pro mou aplikaci

neblíží k doplňku.

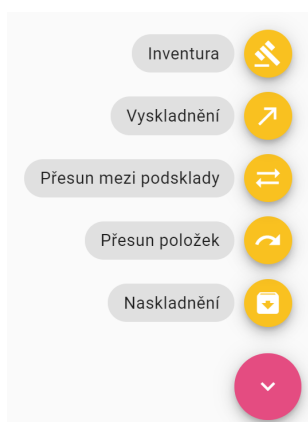
Barvy a idempotence akce. Jelikož mám vybrané dvě akcentní barvy, musel jsem se rozhodnout, kde budu kterou z nich používat. Co se týká zobrazování, je vše poměrně jasné: k základní barvě se volně váže doplňkově-akcentní růžová, a čistě akcentní žlutá barva je ponechána pouze pro důležité položky, které je potřeba speciálně zvýraznit. Jiné je to ale u akcí - tlačítek, odkazů apod. Zde jsem se rozhodl barvu tlačítek volit dle idempotence akce, kterou vyvolávají - tedy zda vyvolaná akce způsobuje vedlejší efekty, nebo ji lze vyvolávat stále dokola. Za vedlejší efekty v aplikaci považuji následující situace:

- navigace na jinou stránku,
- uložení či úprava dat.

Naopak akce, které nemají vedlejší efekt jsou všechny ty, které neukládají žádná data ani nemění navigaci, mohou ale měnit aktuální zobrazení stránky, tedy například něco otevřít, skrýt, přesunout, avšak vždy tak, že se jedná o akci pouze na jednom klientu a zůstávají na jedné stránce. Na obrázcích 4.11 a 4.12 je vidět použití jak idempotentních, tak non-idempotentních akcí: tlačítka *storno* a hlavní akční tlačítko pouze skrývají dialog, respektive rozevírají menu, kdežto ostatních tlačítka a volby vždy buďto odesílají data nebo přecházejí na novou stránku.



Obrázek 4.11: Barvy znázorňující idempotenci akcí: Dialog běžícího času



Obrázek 4.12: Barvy znázorňující idempotenci akcí: Tlačítko tvorby úkolu

4.11 Záznam stráveného času v úkolech

Jedním z požadavků na nový systém byla možnost zaznamenávání stráveného času v úkolech. Jelikož často pracuji v Redmine, což je software pro evidenci požadavků a řízení projektů, kde se čas také zaznamenává, analyzoval jsem, jak se zde se záznamem času pracuje.

Záznam času v Redmine. V čisté instalaci Redmine je možné čas přidávat k úkolům a nebo celkově k projektům. Zatímco záznam času k projektu se musí provést ze samostatné stránky, záznam času k úkolu je možné spojit s jeho aktualizací. Množství stráveného času ale musí vždy zadat sám uživatel - musí si tedy čas buďto pamatovat, nebo zaznamenávat v další externí aplikaci, či vlastních poznámkách. Jelikož je Redmine open-source, tak existuje hned několik pluginů, které umožňují čas strávený v úkolu stopovat a poté ho zapsat automaticky. Problém, který zde identifikuji já osobně spočívá v tom,

že úkolů na Redmine mám často otevřeno i více, nebo naopak žádný, a přesto na některém pracuji například v IDE. Nelze tedy jednoznačně říct, že čas, po který byl úkol otevřen, je stejný jako čas, který byl nad úkolem reálně stráven. Hledal jsem tedy další lepší řešení, a našel vhodnou aplikaci třetí strany.

Záznam času v Toggl. Toggl je webová aplikace, která nabízí i instalaci nativních agentů pro Windows, Mac a mobilní platformy. Díky nainstalovanému agentu je možné spouštět či zastavovat běžící čas klávesovou zkratkou ať už je aktivní browser, IDE, nebo jakýkoliv jiný program. Toto je pro práci, ve které se často přepíná kontext, nebo vznikají pauzy, velmi přínosné. Problém nastává ve chvíli, kdy mám čas uložený v Toggl, ale potřebuji ho zapsat do Redmine. K tomuto účelu jsem napsal vlastní malý doplněk do Google Chrome, který načte časy uložené v Toggl a otevře záložky s předvyplněnými poli, ve kterých čas pouze uložím do Redmine. Vše by šlo řešit i zcela automaticky přes API obou služeb, avšak v době psaní tohoto textu ještě není tato synchronizace na dostatečné úrovni a rád mám kontrolu nad tím, které časy vlastně do Redmine zapisuji.

V odstavcích výše jsem se zamyslel nad záznamem času v systému pro řízení obecných projektů. Aplikace pro správu skladu je však přece jen v něčem specifická - jelikož veškerá manipulace se zbožím musí nutně procházet přes ni - všechny položky musí vždy projít čtečkou / být zadány - mělo by být možné stopovat strávený čas přímo v aplikaci

Záznam času v novém skladovém systému. Pro roli skladníka je poměrně jednoduché automaticky měřit, kolik času strávil v jednotlivých úkolech, a čas odesílat společně s odesláním vyřešení úkolu. U role vedoucího skladu je čas také možné zaznamenávat - zde se jedná o přípravu úkolů nebo jejich kontrolou, stále je ale možné práci provádět i mimo tyto obrazovky, například objednávky se mohou zpracovávat mimo systém a pak se pouze jednorázově zadají. Proto jsem také umožnil vedoucímu skladu přidávat strávený čas i ručně.

Pauzování času. U jakéhokoliv automatického stopování času je nutné mít možnost časomíru pozastavit - typicky například při pauze na oběd či jiné přestávce v práci, během které ale zůstane otevřený jeden úkol. Aplikace vždy zobrazuje, kdy je automatický záznam času spuštěn, a umožňuje ho pozastavit

- přičemž u pozastaveného času není možné v úloze provádět žádné akce, až do opětovného spuštění časomíry.

Navrhované řešení automatického záznamu času je samozřejmě možné zcela vypnout v konfiguraci aplikaci, a také ještě bude před ostrým používáním otestováno uživateli.

TODO popsat nový `vue.config.js` a `vue ui` - dashboard, deps, analyzet atp.

TODO JS Flow

TODO text filtry

<https://itnext.io/yes-this-is-how-to-cache-pages-by-url-with-vue-vue-router-and-keep-alive-component-697ed76896e8>

TODO kontextové menu

TODO exporty do CSV a JS

TODO renderer tabulek

TODO polyfills <https://cli.vuejs.org/guide/browser-compatibility.html>

TODO PR do Vuetify?

TODO persistent store

TODO funkční specifikace jednotlivých úkolů?

4.12 Perličky z vývoje

4.12.1 Špatně importované ikony

Když jsem byl zhruba v polovině tvorby první použitelné verze aplikace, vyšla aktualizace knihovny *Vuetify*, která z verze 1.x poskočila na verzi 2.0. To s sebou neslo poměrně hodně *breaking changes* [52], které jsem ale postupně všechny prošel a aplikaci upravil, takže brzy opět fungovala na nové verzi Vuetify.

Po nějakém čase jsem si ale všiml, že u checkboxů a dalších formulářových prvků chybí některé jejich součásti - například u checkboxu to bylo hodně výrazné - tam chyběl celý zaškrtačací čtvereček a byl vidět pouze *label*. Nejprve

jsem problém ignoroval s tím, že se pravděpodobně jedná o chybu knihovny, a v některé z dalších verzí bude vše opraveno.

Když však ale ani po měsíci nebyly checkboxy stále vidět, začal jsem hledat příčinu problému. Samozřejmě jsem nejprve nahlížel do *Nástrojů vývojáře* v prohlížeči, ale tam jsem nic zajímavého nezjistil - pouze to, že z nějakého důvodu se v mé aplikaci narozdíl od oficiální dokumentace Vuetify [55] (kde checkboxy samozřejmě fungovaly), nerenderuje kus HTML, který je má na starost. Založil jsem si tedy nový lokální projekt s Vue.js + Vuetify, kde checkboxy samozřejmě také fungovaly. Postupně jsem tedy začal odebírat různé závislosti z npm, abych přišel na to, která knihovna tento problém způsobuje. Při tomto procesu jsem rovnou zauditoval, zda opravdu potřebuji všechny dříve používané závislosti, a upravil i některé kusy kódu tak, aby závislosti již nebyly potřebné, a tedy jsem kód vlastně zefektivnil a zmenšil velikost výsledné aplikace. Stále jsem ale nemohl přijít na to, proč nefungují checkboxy.

Teprve asi po 6 hodinách a asi 40x přeinstalovanými všemi závislostmi, jsem se dostal k importu *Material Design Icons* [24]. Vuetify ve verzi 2.0 přidalo do možností své konfigurace klíč, který určuje, který ikonový font se má použít. Při migraci na novou verzi jsem použil ukázkové nastavení této hodnoty, tedy „mdi“. V žádném případě mě totiž nenapadlo, že *Material Design Icons* (*mdi*) a *Material Icons* (*md*) není to samé!

Po chvilce dalšího ladění s importem ikonek vyšlo najevo, že nastavení „mdi“ není kompatibilní s načítáním ikon z CDN Googlu, ale musí se použít balíček z npm. V případě, že chcete načítat ikonky z CDN, musí být hodnota *iconfont* nastavena pouze na „md“. Celý problém, na kterém jsem strávil tolik hodin a nechápavých výrazů šel tedy opravit diffem z ukázky kódu 4.12.

```
1 -   iconfont: 'mdi',
2 +   iconfont: 'md',
```

Ukázka kódu 4.12: Diff nastavení fontu ikonek ve Vuetify

Testování

5.1 Testování aplikace

5.1.1 První testování výsledné aplikace - během vývoje s odbornou osobou

První uživatelské testování výsledné aplikace proběhlo při jejím vývoji - 19. 9. 2019 - v době testu byly připravené základní možnost vkládání a úprav většiny entit v systému (skladové položky, dodavatelé atp.) a necelé čtyři ze stěžených úloh - příjem dodávky, naskladnění, inventura a část přesunu zboží.

Aplikaci testovala osoba, která mimo jiné činnosti pracuje také se starým skladovým systémem Sysel v roli vedoucího skladu.

Testování proběhlo při neformálním setkání v běžné kanceláři, testera jsem instruoval k tomu, aby použil svůj notebook pro zobrazení režimu správce skladu, a mobilní telefon pro roli skladníka. Zatímco na počítači bylo vše v pořádku, neboť byl použit Google Chrome, ve kterém aplikaci spouštím i při vývoji, na mobilním zařízení nejprve nastal malý problém, a to z důvodu použití prohlížeče *Samsung Internet*, který nepodporuje některé moderní Javascriptové konstrukce, na které aplikace spoléhá. Ačkoliv toto nebude pro samotné použití aplikace problém, protože cílové zařízení pro skladníky je jasně dané a aplikace se tam bude otvírat ve WebView, i přesto není na škodu zachovat kompatibilitu i s jinými mobilními prohlížeči, třeba i pro potřeby vedoucích, aby taktéž mohli pracovat z mobilních zařízení. Již před testováním jsem věděl, že je potřeba

5. TESTOVÁNÍ

zavést nějakou detekci prohlížečů a případně uživatele informovat o nekompatibilitě aplikace se zvoleným browserem, ale po této skutečnost jsem ještě zvýšil této úpravě prioritu.

Výstupem z tohoto testování vývojové verze aplikace je seznam postřehů, chyb a návrhů na zlepšení, které jsou k naleznutí v příloze E.

Zde je vhodné napsat, že zhruba pětinu všech požadavků a chyb jsem nějakým způsobem evidoval buďto formou *TODO komentářů* přímo v kódu aplikace, nebo kartami v Trello, avšak pro kompletnost zápisu jsem je ponechal i tam.

Závěr

TODO

Zdroje

1. ALPERT, Sophie. *Change license and remove references to PATENTS* [online]. 2017 (cit. 2018-12-21). Dostupné z: <https://github.com/facebook/react/commit/b765fb25ebc6e53bb8de2496d2828d9d01c2774b#diff-9879d6db96fd29134fc802214163b95a>.
2. ANDRUSHKO, Sviatoslav. *The Best JS Frameworks for Front End* [online]. 2018 (cit. 2018-12-21). Dostupné z: <https://rubygarage.org/blog/best-javascript-frameworks-for-front-end>.
3. *Angular - One framework. Mobile & desktop.* [online]. 2018 (cit. 2018-12-16). Dostupné z: <https://angular.io/>.
4. *Angular - Testing* [online]. 2019 (cit. 2019-01-22). Dostupné z: <https://angular.io/guide/testing>.
5. *Angular - What is Angular?* [online]. 2018 (cit. 2018-12-28). Dostupné z: <https://angular.io/docs>.
6. *Babel · The compiler for next generation JavaScript* [online]. 2019 (cit. 2019-03-01). Dostupné z: <https://babeljs.io/>.
7. *Backbone Test Suite* [online] (cit. 2018-12-28). Dostupné z: <http://backbonejs.org/test/>.
8. *Backbone.js* [online]. 2016 (cit. 2018-12-17). Dostupné z: <http://backbonejs.org/>.
9. *Backbone.js* [online]. 2016 (cit. 2018-12-28). Dostupné z: <http://backbonejs.org/#Getting-started>.

10. *Basic color schemes - Introduction to Color Theory* [online] (cit. 2019-11-23). Dostupné z: <https://www.tigercolor.com/color-lab/color-theory/color-theory-intro.htm>.
11. *Bc. Oldřich Malec / mi-nur-sysel - Gitlab* [online]. 2018 (cit. 2019-01-18). Dostupné z: <https://gitlab.fit.cvut.cz/malecold/mi-nur-sysel>.
12. *Build amazing things* [online]. 2018 (cit. 2018-12-21). Dostupné z: <https://www.npmjs.com/>.
13. CANDILLON, William. *A Swagger Codegen for typescript, nodejs & angularjs* [online]. 2019 (cit. 2019-10-17). Dostupné z: <https://github.com/wcandillon/swagger-js-codegen>.
14. *Classes - JavaScript | MDN* [online]. 2019 (cit. 2019-03-02). Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes>.
15. *Collection: Front-end JavaScript frameworks* [online]. 2018 (cit. 2018-12-17). Dostupné z: <https://github.com/collections/front-end-javascript-frameworks>.
16. *Comparison with Other Frameworks - Vue.js* [online]. 2018 (cit. 2018-12-14). Dostupné z: <https://vuejs.org/v2/guide/comparison.html>.
17. EHRENBURG, Daniel. *Class field declarations for JavaScript* [online]. 2019 (cit. 2019-03-02). Dostupné z: <https://github.com/tc39/proposal-class-fields#private-fields>.
18. *Ember.js - A framework for ambitious web developers* [online]. 2018 (cit. 2018-12-17). Dostupné z: <https://www.emberjs.com/>.
19. *Ember.js Guides - Guides and Tutorials - Ember Guides* [online]. 2018 (cit. 2018-12-28). Dostupné z: <https://guides.emberjs.com/release/>.
20. *Getting Started - React* [online]. 2018 (cit. 2018-12-28). Dostupné z: <https://reactjs.org/docs/getting-started.html>.
21. GOEL, Aman. *Top 10 Web Development Frameworks in 2018* [online]. 2018 (cit. 2018-12-17). Dostupné z: <https://hackr.io/blog/top-10-web-development-frameworks-in-2018>.
22. HERMANN, Christoph. *retrieve-arguments/index.js* [online]. 2014 (cit. 2019-10-17). Dostupné z: <https://github.com/stoeffel/retrieve-arguments/blob/master/index.js>.

23. CHANDRA, Rajdeep. *My experience with Angular 2 , React and Vue* [online]. 2018 (cit. 2018-12-21). Dostupné z: <https://medium.com/@rajrock38/my-experience-with-angular-2-react-and-vue-fb654e3ecf33>.
24. *Icons - Material Design* [online]. 2019 (cit. 2019-09-24). Dostupné z: <https://material.io/resources/icons/>.
25. *Introduction - Enzyme* [online]. 2019 (cit. 2019-01-22). Dostupné z: <https://airbnb.io/enzyme/>.
26. *Introduction - Testing - Ember Guides* [online]. 2019 (cit. 2019-01-22). Dostupné z: <https://guides.emberjs.com/release/testing/>.
27. *Introduction - Vue.js* [online]. 2018 (cit. 2018-12-28). Dostupné z: <https://vuejs.org/v2/guide/>.
28. *Introduction | Vue Router* [online]. 2018 (cit. 2019-02-09). Dostupné z: <https://router.vuejs.org/>.
29. *Introduction | Vue Test Utils* [online]. 2019 (cit. 2019-01-22). Dostupné z: <https://vue-test-utils.vuejs.org/>.
30. *Jagu - Software na míru, webové prezentace, grafika, webhosting* [online]. 2018 (cit. 2018-12-21). Dostupné z: <https://www.jagu.cz/>.
31. JANČA, Marek. *Webpack - moderní Web Development | Ackee blog* [online]. 2017 (cit. 2019-02-11). Dostupné z: <https://www.ackee.cz/blog/moderni-web-development-webpack/>.
32. *JavaScript Error Tracking* [online]. 2019 (cit. 2019-01-21). Dostupné z: <https://sentry.io/for/javascript/>.
33. *Javascript library to connect to swagger-enabled APIs via browser or nodejs* [online]. 2019 (cit. 2019-10-17). Dostupné z: <https://github.com/swagger-api/swagger-js>.
34. *Jest - Delightful JavaScript Testing* [online]. 2019 (cit. 2019-01-22). Dostupné z: <https://jestjs.io/>.
35. JIMBO, Yoshihide. *How to optimize moment.js with webpack* [online]. 2018 (cit. 2019-11-25). Dostupné z: <https://github.com/jmblog/how-to-optimize-momentjs-with-webpack>.

36. KAWAGUCHI, Kazuya. *Vue i18n* [online]. 2019 (cit. 2019-03-04). Dostupné z: <https://kazupon.github.io/vue-i18n/>.
37. KUROSKI, Daniel. *Working an application in Vue.js with TDD* [online]. 2018 (cit. 2019-01-22). Dostupné z: <https://vue-test-utils.vuejs.org/>.
38. LASCIK, V. *Honest look at Ember in the middle of 2018* [online]. 2018 (cit. 2018-12-20). Dostupné z: <https://medium.com/@vlascik/honest-look-at-ember-in-the-middle-of-2018-a0dc2787e506>.
39. *Mixins - Vue.js* [online]. 2019 (cit. 2019-10-03). Dostupné z: <https://vuejs.org/v2/guide/mixins.html>.
40. *Mocha - the fun, simple, flexible JavaScript test framework* [online]. 2019 (cit. 2019-01-22). Dostupné z: <https://mochajs.org/>.
41. *Moment.js / Home* [online] (cit. 2019-11-25). Dostupné z: <https://momentjs.com/>.
42. MOURA, Marcos. *Vue Material - Material Design for Vue.js* [online]. 2019 (cit. 2019-03-02). Dostupné z: <https://vuematerial.io/>.
43. *Navigator.onLine - Web APIs | MDN* [online] (cit. 2019-04-19). Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/API/NavigatorOnLine/onLine>.
44. NIELSEN, Jakob. *Confirmation Dialogs Can Prevent User Errors — If Not Overused* [online]. 2018 (cit. 2019-09-27). Dostupné z: <https://www.nngroup.com/articles/confirmation-dialog/>.
45. *Object.assign() - Javascript | MDN* [online]. 2019 (cit. 2019-09-24). Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/assign.
46. OGDEN, Cody. *Killed by Google - Google Graveyard - A Google Cemetery* [online]. 2018 (cit. 2018-12-23). Dostupné z: <https://killedbygoogle.com/>.
47. *OpenAPI Specification* [online]. 2017 (cit. 2019-10-17). Dostupné z: <https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md>.
48. *Overview | Vue CLI 3* [online]. 2019 (cit. 2019-02-11). Dostupné z: <https://cli.vuejs.org/guide/>.

-
49. *Prototypes, Specifications, and Diagrams in One Tool | Axure Software* [online]. 2018 (cit. 2018-03-07). Dostupné z: <https://www.axure.com/>.
 50. *Protractor - end-to-end testing for AngularJS* [online]. 2019 (cit. 2019-01-22). Dostupné z: <https://www.protractortest.org/>.
 51. *React - A JavaScript library for building user interfaces* [online]. 2018 (cit. 2018-12-16). Dostupné z: <https://reactjs.org/>.
 52. *Release v2.0.0 - vuetifyjs/vuetify* [online]. 2019 (cit. 2019-07-27). Dostupné z: <https://github.com/vuetifyjs/vuetify/releases/tag/v2.0.0>.
 53. ROEMER, Ryan. *Backbone.js Testing* [online]. 2018 (cit. 2018-12-28). Dostupné z: <http://backbone-testing.com/>.
 54. *Routing* [online]. 2018 (cit. 2018-12-21). Dostupné z: <https://vuejs.org/v2/guide/routing.html>.
 55. *Selection control components - Vuetify.js* [online]. 2019 (cit. 2019-09-24). Dostupné z: <https://vuetifyjs.com/en/components/selection-controls>.
 56. *Sentry | Error Tracking Software — JavaScript, Python, PHP, Ruby, more* [online]. 2019 (cit. 2019-01-21). Dostupné z: <https://sentry.io/welcome/>.
 57. *Service Worker API - Web APIs | MDN* [online]. 2019 (cit. 2019-11-25). Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API.
 58. *Service Workers - W3C First Public Working Draft 08 May 2014* [online]. 2014 (cit. 2019-11-25). Dostupné z: <https://www.w3.org/TR/2014/WD-service-workers-20140508/>.
 59. SCHUBERT, Jim. *OpenAPI Generator* [online]. 2019 (cit. 2019-10-17). Dostupné z: <https://plugins.jetbrains.com/plugin/8433-openapi-generator>.
 60. *swagger-codegen contains a template-driven engine to generate documentation, API clients and server stubs in different languages by parsing your OpenAPI / Swagger definition.* [online]. 2019 (cit. 2019-10-17). Dostupné z: <https://github.com/swagger-api/swagger-codegen>.
 61. *Test Utilities - React* [online]. 2019 (cit. 2019-01-22). Dostupné z: <https://reactjs.org/docs/test-utils.html>.

62. *Tilde Inc. - About Us* [online]. 2018 (cit. 2018-12-17). Dostupné z: <https://www.tilde.io/about-us/>.
63. *Unit Testing - Vue.js* [online]. 2019 (cit. 2019-01-22). Dostupné z: <https://vuejs.org/v2/guide/unit-testing.html>.
64. *UX Tool Time* [online]. 2018 (cit. 2018-03-07). Dostupné z: <http://uxtooltime.com/>.
65. VALENTA, Michal. *DBS – transakční zpracování* [online]. 2010 (cit. 2019-09-27). Dostupné z: https://users.fit.cvut.cz/valenta/doku/lib/exe/fetch.php/bivs/dbs_09_transakce.pdf.
66. *Vue.js Material Component Framework — Vuetify.js* [online]. 2019 (cit. 2019-03-02). Dostupné z: <https://vuetifyjs.com/>.
67. *Web Content Accessibility Guidelines* [online]. 2008 (cit. 2019-11-23). Dostupné z: <https://www.w3.org/TR/WCAG20/>.
68. *webpack* [online]. 2019 (cit. 2019-02-10). Dostupné z: <https://webpack.js.org/>.
69. *What is Vuex? | Vuex* [online]. 2019 (cit. 2019-03-04). Dostupné z: <https://vuex.vuejs.org/>.
70. WOLFF, Adam. *Relicensing React, Jest, Flow, and Immutable.js* [online]. 2017 (cit. 2018-12-21). Dostupné z: <https://code.fb.com/web/relicensing-react-jest-flow-and-immutable-js/>.
71. YAOLONG, Dengy. *how to change document.title in vue-router? · Issue #914 · vuejs/vue-router* [online]. 2016 (cit. 2019-02-09). Dostupné z: <https://github.com/vuejs/vue-router/issues/914>.
72. YOU, Evan. *Vue - The Progressive JavaScript Framework* [online]. 2018 (cit. 2018-12-16). Dostupné z: <https://vuejs.org/>.

Seznam použitých zkratk

API	Application Programming Interface
ARES	Administrativní registr ekonomických subjektů
BSD	Berkeley Software Distribution
CDN	Content Delivery Network
CI	Continuous Integration
CLI	Command Line Interface
CORS	Cross-origin resource sharing
CSS	Cascading Style Sheets
DI	Dependency Injection
DPH	Daň z přidané hodnoty
E2E	End-to-end
EAN	European Article Number
ES	ECMAScript
FIT	Fakulta informačních technologií
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IBAN	International Bank Account Number
IDE	Integrated Development Environment
IČO	Identifikační číslo osoby
JS	Javascript
PWA	Progresivní webová aplikace
MIT	Massachusetts Institute of Technology
MI	Magisterská Informatika

A. SEZNAM POUŽITÝCH ZKRATEK

NPM	Node Package Manager
NUR	Návrh uživatelského rozhraní
OOP	Objektově orientované programování
OS	Operační Systém
PDF	Portable Document Format
PHP	PHP: Hypertext Preprocessor
SASS	Syntactically Awesome Style Sheets
SPA	Single Page Application
SVN	Subversion
TDD	Test-driven development
UI	User Interface
URL	Uniform Resource Locator

Slovník pojmů

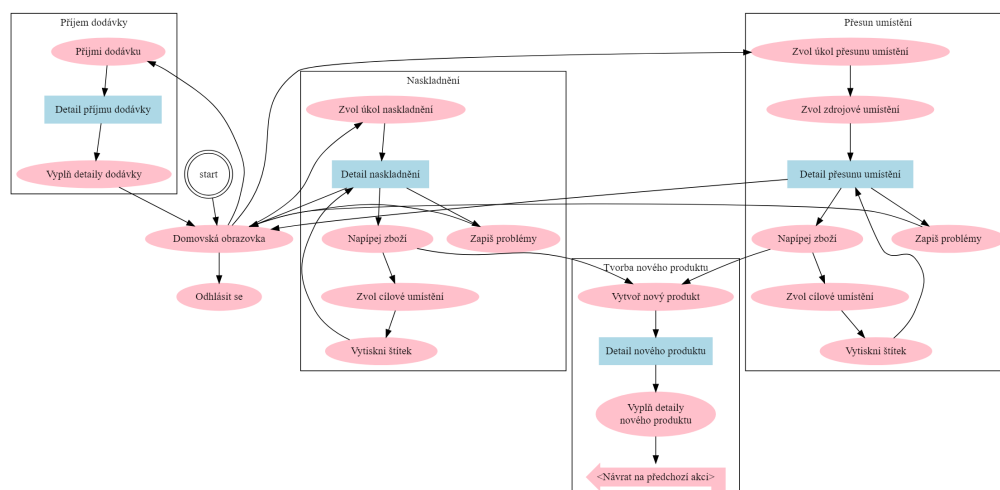
Backend	Část aplikace, která se stará o ukládání dat, jejich zpracování a bussiness logiku. Většinou není přímo přístupná koncovému uživateli, ten k ní přistupuje přes frontend.
Dependency Injection	Technika, která umožňuje "vlození" instance objektu, který poskytuje nějakou službu, do jiného objektu, který pak může danou službu efektivně používat.
EcmaScript	Definice programovacího jazyka, kterou implementuje například Javascript.
Favicon	Ikonka webové stránky, která se ve většině prohlížečů zobrazuje v panelu v horní liště a na dalších místech, které odkazují na daný web.
Framework	Softwarová struktura, která slouží jako podpora pro pohodlnější programování. Může obsahovat podpůrné funkce, knihovny či nástroje pro efektivnější, bezpečnější a pohodlnější vývoj softwaru.
Frontent	Část aplikace, s kterou přímo interaguje koncový uživatel či administrátor, typicky pomocí GUI. Většinou komunikuje s druhou, serverovou částí - backendem.

Git	Git je distribuovaný verzovací nástroj určený zejména pro sdílení a verzování zdrojových kódů aplikací, ale i jiných assetů.
GitHub	GitHub je webová služba podporující vývoj softwaru za pomoci verzovacího nástroje Git.
Heuristická Analýza	Metoda testování softwaru, při které využívá expert své znalosti, aby odhadl možné chování ostatních uživatelů. Typicky prochází několik předem stanovených bodů a zkoumá, zda jsou splněny.
Hot Swapping	Jedná se o způsob zobrazení změn v aplikaci při změně jejího kódu. Při hot-swappingu není potřeba spouštět žádné procesy sestavení aplikace a často ani načíst znovu obrazovku - vše je provedeno automaticky za běhu a změny jsou viditelné ihned.
Idempotence	Idempotence je vlastnost operací. Operace je idempotentní, pokud jejím opakovaným použitím na nějaký vstup vznikne stejný výstup, jako vznikne jediným použitím dané operace. V kontextu programu programu se tedy jedná o akci, kterou lze vyvolávat opakovaně, aniž by to způsobilo vedlejší efekty.
JSdoc	Specifikace zápisu dokumentace Javascriptového kódu.
Middleware	Software realizující integraci mezi dvěma jinými systémy, typicky pomocí API.
Redmine	Software pro evidenci požadavků a řízení projektů.
Reflexe	Popsání programovacího jazyko sebou samým - například možnost zjistit si uvnitř funkce, jaké jsou její parametry.

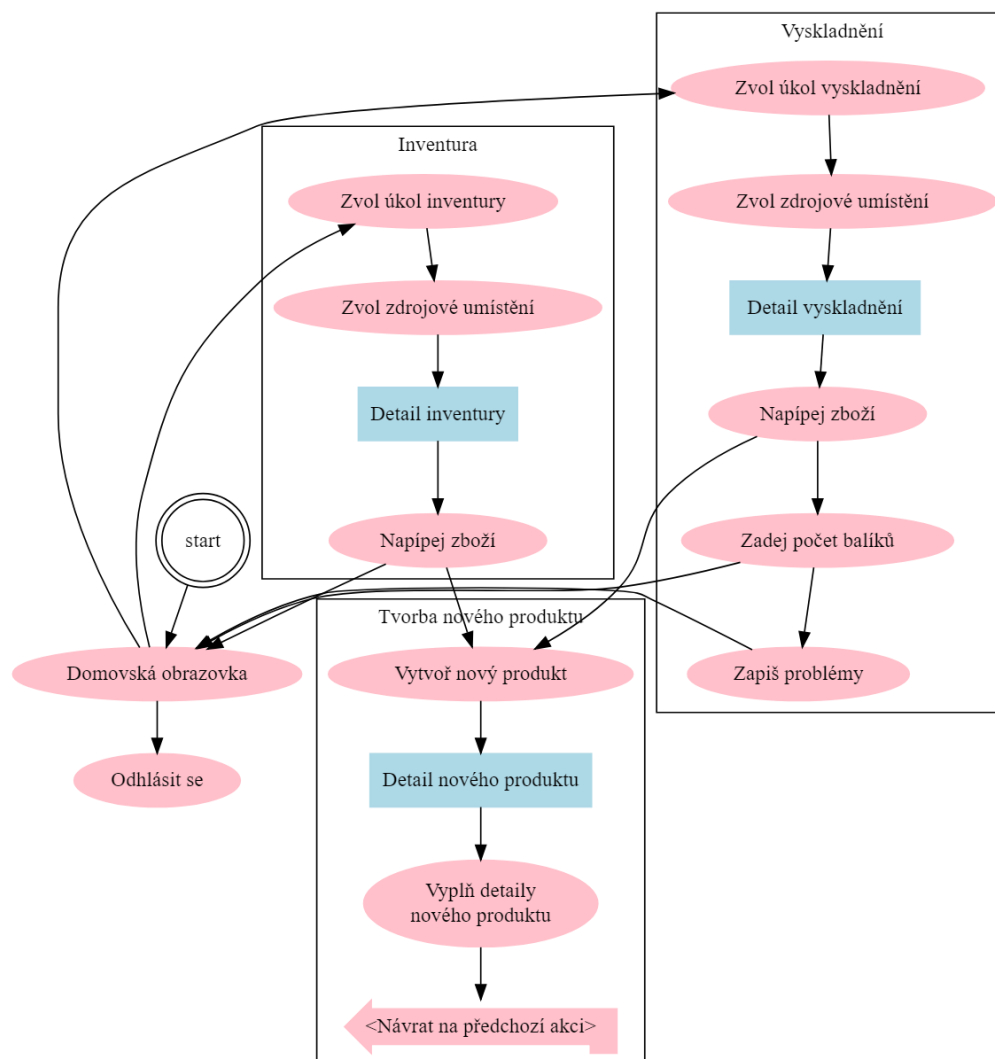
Sentry	Sentry je webová služba pro sledování chyb, které v aplikaci nastanou. Při použití v produkčním prostředí může vývojář díky Sentry o chybě vědět ještě dříve, než ji uživatel nahlásí, a to včetně všech detailů, jako například jaké kroky chybě předcházely, prostředí, ve kterém k chybě došlo a mnoho dalších.
Snackbar	Velmi podobný toast message, narozdíl od čistě informativního toast message umožňuje Snackbar uživateli spustit i programátorem definovanou akci. Zobrazuje se typicky ve spodní části aplikace a informuje o proběhlé akci.
Subversion	Systém pro správu zdrojových kódů a dalších assetů. Dnes se používá zejména pro ne-textové soubory, ty textové jsou většinou verzovány v gitu.
Toast message	Krátká informativní zpráva, který se objevuje ve spodní části aplikace a obsahuje typicky informaci o potvrzení provedení akce.
Trello	Webová služba pro evidenci úkolů, s flexibilním nastavením sloupců, označení atp. ve stylu kanban.
TypeScript	Nadmnožina JavaScriptu, která jej rozšiřuje především o statické typování proměnných a další atributy z OOP.
Use case	Případ použití, neboli seznam kroků či událostí, ke kterým dochází mezi uživatelem aplikace a aplikací samotnou, jejichž účelem je dosažení nějakého cíle.
Vuetify	Knihovna pro Vue.js, která implementuje Material Design a poskytuje základní stavební komponenty, ale i pokročilé bloky jako například datové tabulky.

Vuex	Knihovna pro Vue.js, která se stará o state management.
Webpack	Webpack je software, který zpracovává součásti webových aplikací a tvoří z nich balíčky vhodné pro webové prohlížeče. Primárně je zaměřen na Javascript, ale dokáže zpracovávat i řadu dalších formátů, přes styly v css či sass, obrázky v png, jpeg, svg či konfigurace v json, yaml a dalších.
WebStorm	IDE od společnosti JetBrains pro vývoj webových aplikací především v Javascriptu či Typescriptu.
WebView	Komponenta nativní Android aplikace, která zobrazuje stanovenou URL jako svůj obsah. Používá se zejména v místech, kde je žádoucí zobrazovat obsah z webu, ale je potřeba přístup k funkcím zařízení, ke kterým není možné přistupovat z běžného webového prohlížeče.

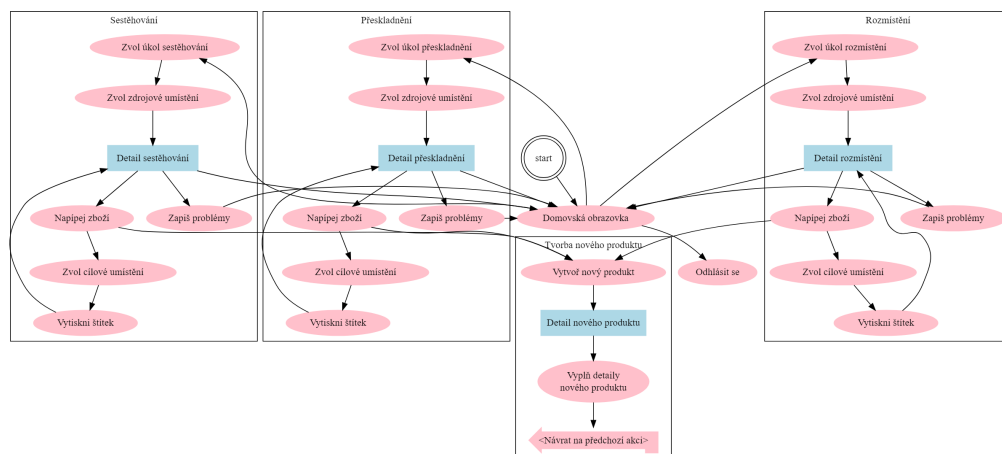
Diagramy průchodu systému skladníkem



Obrázek C.1: Diagram průchodu systému skladníkem: první část



Obrázek C.2: Diagram průchodu systému skladníkem: druhá část



Obrázek C.3: Diagram průchodu systému skladníkem: třetí část

Návrh procesů ve skladovém systému

Slovník pojmů

- Majitel
 - Ve starém syslovi označován jako Sklad.
 - Zákazník skladu, který financuje skladování zboží.
- Projekt
 - Majitel může mít více projektů
- Sklad
 - Ve starém syslu nemá ekvivalent.
 - Budova, ve které se nachází konkrétní umístění.
- Umístění
 - Shodná terminologie se starým systémem
 - Konkrétně umístění v rámci Skladu, může na něm být zboží více Majitelů, nebo může být vyhrazeno pouze pro konkrétního Majitele
- Kód
 - U zboží = EAN
 - U umístění a dalších neartiklových položek = čárový kód, který lze napípnout čtečkou.
- Zákazník
 - Osoba, pro kterou systém vyskladňuje zboží. Je uveden na výdejce v kolonce odběratel.
 - V případě napojení na eshop přenechat evidenci zákazníků na eshopu.

Entity a jejich atributy z pohledu uživatele

- Zboží
 - EAN (i více), název, model, výrobce, popis, prodejní cena, nákupní cena, DPH, stav (aktivní/neaktivní), Stav se využívá při objednávkách - neaktivní zboží nelze objednat.
- Majitel
 - název, adresa, IČ, DIČ, č. bank. účtu, telefon, dodatek dodáku?
- Sklad
 - název, adresa, telefon a email na vedoucího skladu
- Umístění
 - název, nadřazený sklad, [Kód](#), whitelist majitelů
- Výrobce
 - název, zkratka
- Zákazník
 - název, adresa, telefon, email, ič, dič

Procesy shodné pro obě role

1. Přihlásit se
2. Změnit roli (řešeno formou ACL, nesmí být jiný pohled!)
3. Odhlásit se

Procesy role Skladník

1. Příjem dodávky zboží
 - 1.1. Skladník úkol iniciuje sám volbou v systému
 - 1.2. Skladník vyplní informace o dodávce zboží (číslo dodacího listu, číslo faktury, vybere dodavatele) a nafotí dodací list(y)
 - 1.3. K dané úloze může připojit komentář
 - 1.4. Po odeslání je úloha předána vedoucímu ke schválení
2. Naskladnění zboží
 - 2.1. Skladník přijme úlohu.
 - 2.2. Skladník vidí základní informace o naskladnění a komentáře vedoucího.
 - 2.3. Skladník skenuje čárové kódy zboží a umístění pomocí čtečky čárových kódů.
 - 2.4. Když napíše neznámou položku, může [Vytvořit nové zboží](#) nebo [Vytvořit umístění](#).
 - 2.5. U naskenovaného zboží může:
 - 2.5.1. měnit počet kusů,
 - 2.5.2. měnit cílové umístění,
 - 2.5.3. smazat ho,
 - 2.5.4. tisknout pro něj štítky.
 - 2.6. Úlohu celkově může
 - 2.6.1. kdykoliv přerušit a později se k ní vrátit,
 - 2.6.2. [nahlásit problém](#),
 - 2.6.3. dokončit.
3. Vyskladnění zboží
 - 3.1. Skladník přijme úlohu.
 - 3.2. V aplikaci vidí umístění, kde může požadované zboží nalézt.
 - 3.3. Skladník napíše umístění, ze kterého hodlá zboží sebrat.
 - 3.4. Skladník skenuje čárové kódy zboží.
 - 3.5. U naskenovaného zboží může:
 - 3.5.1. měnit zdrojové umístění,
 - 3.5.2. počet kusů.
 - 3.6. Úlohu celkově může
 - 3.6.1. kdykoliv přerušit a později se k ní vrátit,
 - 3.6.2. [nahlásit problém](#),
 - 3.6.3. dokončit.

4. Příprava Expedice
 - 4.1. Skladník přijme úlohu
 - 4.2. Zabalí to, pípá/vyplňuje materiál, který spotřeboval (palety, balíky, folie)
 - 4.3. Vyplní počet balíků
 - 4.3.1. Vytiskne štítky na polep balíků
 - 4.4. Vyfotit zabalené zboží?
 - 4.5. Úlohu dokončí tím, že ji *označí jako zabalenou*
 - 4.6. Úkol se mění na **Připravená expedice**
5. Expedice
 - 5.1. Skladník může v úkolu přepnout do **úpravy balení** a úkol se tím vrací do Expedice
 - 5.2. Nebo může *Označit jako odvezené*
6. Inventura
 - 6.1. Skladník přijme úlohu.
 - 6.2. Skladník napíše umístění, které se chystá skenovat.
 - 6.3. Skladník skenuje čárové kódy zboží.
 - 6.4. U naskenovaného zboží může skladník nastavit počet kusů.
 - 6.5. Skladník vidí zboží, které již zpracoval, a může jej zpětně upravovat či mazat.
 - 6.6. Úlohu celkově může
 - 6.6.1. kdykoliv přerušit a později se k ní vrátit,
 - 6.6.2. [nahlásit problém](#),
 - 6.6.3. dokončit.
7. Přeskladnění zboží (Seskladnění)
 - 7.1. Skladník přijme úlohu.
 - 7.2. Skladník skenuje čárové kódy zboží.
 - 7.3. U naskenovaného zboží může:
 - 7.3.1. měnit počet kusů,
 - 7.3.2. měnit zdrojové umístění,
 - 7.4. Úlohu celkově může
 - 7.4.1. kdykoliv přerušit a později se k ní vrátit,
 - 7.4.2. [nahlásit problém](#),
 - 7.4.3. Dokončit.
8. Rozmístění zboží
 - 8.1. Skladník přijme úlohu.
 - 8.2. Skladník napíše umístění, ze kterého hodlá zboží sebrat.
 - 8.3. Skladník skenuje čárové kódy zboží
 - 8.4. U naskenovaného zboží může:
 - 8.4.1. měnit počet kusů,
 - 8.4.2. měnit cílové umístění,
 - 8.4.3. smazat ho.
 - 8.5. Úlohu celkově může
 - 8.5.1. kdykoliv přerušit a později se k ní vrátit,
 - 8.5.2. [nahlásit problém](#),

- 8.5.3. dokončit.
- 9. Přezásobení
 - 9.1. Skladník přijme úlohu.
 - 9.2. Skladník skenuje čárové kódy zboží.
 - 9.3. U naskenovaného zboží může:
 - 9.3.1. měnit zdrojové umístění,
 - 9.3.2. měnit cílové umístění,
 - 9.3.3. měnit počet kusů,
 - 9.3.4. smazat ho.
 - 9.4. Úlohu celkově může
 - 9.4.1. kdykoliv přerušit a později se k ní vrátit,
 - 9.4.2. [nahlásit problém](#),
 - 9.4.3. dokončit.
- 10. Přesun umístění
 - 10.1. Skladník samovolně zahájí přesun umístění nebo přijme úkol od vedoucího
 - 10.2. Načte čárový kód zdrojového umístění.
 - 10.3. Načte čárový kód cílového umístění.
 - 10.4. Úlohu celkově může
 - 10.4.1. kdykoliv přerušit a později se k ní vrátit,
 - 10.4.2. [nahlásit problém](#),
 - 10.4.3. dokončit.
- 11. Přehledy úloh
 - 11.1. Rozpracované úlohy
 - 11.1.1. Skladník vidí seznam úloh, které má rozpracované.
 - 11.2. Volné úlohy
 - 11.2.1. Skladník vidí seznam úloh, které nejsou přiřazené žádnému skladníkovi. Úlohy jsou v seznamu seskupeny dle typu.
 - 11.3. Uzavřené úlohy
 - 11.3.1. Skladník si může zobrazit přehled všech úloh, které během svého působení vyřešil.
 - 11.3.2. U každé úlohy si může zobrazit podrobnosti
 - 11.4. Úlohy čekající na schválení vedoucím
 - 11.4.1. Skladník může zobrazit přehled úloh, které čekají na zpracování vedoucím skladu. Zobrazeny jsou pouze úlohy, které daný skladník předal vedoucímu.
 - 11.5. Strávený čas
 - 11.5.1. U daného majitele zboží
 - 11.5.2. U daného typu úlohy
 - 11.5.3. atp...

Hlášení problému s úkolem

1. Skladník vyplní důvod problému

2. Skladník vyplní, zda zboží zůstalo ve zdrojovém umístění, u něj, nebo je v cílovém

Tvorba zboží

1. Skladník může vytvořit nové zboží, pokud načte čárový kód, který není v systému evidován.
2. Vyplní doplňující informace o zboží (název, popis, výrobce a model).
3. Nafotí zboží
4. Po uložení jej může ihned používat.

Procesy role Vedoucí

1. Evidence skladů
 - 1.1. Sklad lze vyhledat vyplněním/napípnutím [Kódu](#)
 - 1.2. Tvorba a úprava skladů
 - 1.2.1. Vedoucí otevře stránku pro tvorbu či úpravu skladu.
 - 1.2.2. Vyplní [informace o skladu](#).
 - 1.2.3. Potvrzením formuláře uloží změny do systému.
 - 1.3. Hromadná úprava adres skladů
 - 1.3.1. Úpravy může provádět:
 - 1.3.1.1. přímo v aplikaci,
 - 1.3.1.2. hromadně exportem/importem CSV/XLS.
 - 1.3.2. Aplikace vypíše, jaké změny se chystá aplikovat. Vedoucí může změny potvrdit, nebo odmítnout.
 - 1.4. Nastavení spodních limitů pro automatické doplnění zboží na sklad
 - 1.4.1. Vedoucí otevře stránku, kde může nastavovat limity pro automatické doplnění zboží na sklad.
 - 1.4.2. Z nabídky zboží může libovolně vybírat nové zboží, které má být automaticky doplněno, nebo může stávající limity upravovat či mazat.
 - 1.4.3. Tabulku s limity může filtrovat podle výrobce / model atp.
 - 1.4.4. Z aktivních filtrů může udělat export pro objednávky u dodavatele (xlsx)
 - 1.4.5. Rozhodne-li se změny aplikovat, musí provedené změny nejprve potvrdit.
 - 1.5. Hromadné nastavení spodních limitů pro automatické doplnění zboží na sklad
 - 1.5.1. Vedoucí připraví CSV nebo XLS soubor pro import limitů (ve stanoveném formátu), který nahraje do aplikace.
 - 1.5.2. Aplikace vypíše, jaké změny se chystá aplikovat. Vedoucí může změny potvrdit, nebo odmítnout.
2. Evidence umístění
 - 2.1. Umístění lze vyhledat vyplněním/napípnutím [Kódu](#)
 - 2.2. Tvorba a úprava umístění
 - 2.2.1. Vyplní [informace o umístění](#).
 - 2.2.2. Volitelně může vytisknout štítek pro dané umístění.
 - 2.3. Přehled umístění

- 2.3.1. Vedoucí může filtrovat zboží, které je aktuálně na daném umístění.
 - 2.3.2. Export stavu zboží na umístění = Link do sekce Přehledů
 - 2.3.3. Tvorba úkolu souvisejícího s umístěním = Link do sekce Tvorba úlohy
- 3. Evidence výrobců
 - 3.1. Výrobce lze vyhledat vyplněním/napípnutím
 - 3.1.1. Výrobce lze hledat podle jeho Zkratky, Názvu, nebo napípnutím libovolného zboží, u kterého je přiřazen
 - 3.2. Tvorba a úprava výrobců
 - 3.2.1. Vedoucí otevře stránku pro tvorbu či úpravu výrobce.
 - 3.2.2. Vyplní [informace o výrobci](#).
 - 3.2.3. Potvrzením formuláře uložení změny do systému.
 - 3.3. Přehled výrobce = Link do sekce Přehledy
- 4. Evidence zboží
 - 4.1. Zboží lze vyhledat vyplněním/napípnutím EANu
 - 4.2. Tvorba a úprava zboží
 - 4.2.1. Vedoucí otevře stránku pro tvorbu či úpravu zboží.
 - 4.2.2. Vyplní základní informace o produktu (název, model, výrobce, popis, prodejní a nákupní cenu). Dále může připojit fotografii a nastavit jeden či více EAN kódů.
 - 4.2.3. Volitelně může vytisknout štítek produktu.
 - 4.2.4. Potvrzením formuláře uloží produkt do systému.
 - 4.3. Hromadný import zboží
 - 4.3.1. Vedoucí připraví CSV nebo XLS soubor pro import zboží (ve stanoveném formátu), který nahraje do aplikace.
 - 4.3.2. Aplikace vypíše, jaké změny se chystá aplikovat. Vedoucí může změny potvrdit, nebo odmítnout.
- 5. Evidence uživatelů - vázané na *Angler* (autorizační server)
 - 5.1. Tvorba a úprava uživatelů
 - 5.1.1. Vedoucí otevře stránku pro správu uživatelů.
 - 5.1.2. Vyplní informace o uživateli (jméno a příjmení, uživatelské jméno, heslo, privilegia).
 - 5.1.3. Volitelně může nastavit uživatele jako deaktivovaného.
 - 5.1.4. Potvrzením formuláře uloží změny do autorizačního serveru.
- 6. Evidence majitelů
 - 6.1. Tvorba a úprava majitelů
 - 6.1.1. Vedoucí otevře stránku pro tvorbu či úpravu majitelů.
 - 6.1.2. Vyplní informace o majiteli.
 - 6.1.3. Potvrzením formuláře uloží změny do systému.
- 7. Evidence zákazníků
 - 7.1. Tvorba a úprava zákazníků
 - 7.1.1. Vedoucí otevře stránku pro tvorbu či úpravu zákazníků.
 - 7.1.2. Vyplní informace o zákazníkovi.
 - 7.1.3. Potvrzením formuláře uloží změny do systému.

- 7.2. Export dostupnosti zboží
- 8. Evidence úloh
 - 8.1. Tvorba úlohy
 - 8.1.1. Naskladnění bez dodávky
 - 8.1.1.1. Vedoucí vyplní
 - 8.1.1.1.1. kterého skladu a majitele se naskladnění týká,
 - 8.1.1.1.2. určí prioritu úlohy,
 - 8.1.1.1.3. určí, na které umístění má být zboží umístěno
 - 8.1.1.1.3.1. Pokud si volbou vhodného umístění není jistý, může přenechat volbu na skladníkovi, nebo na rozhodnutí aplikace.
 - 8.1.1.2. Volitelně může zapsat doplňující informace (pokyny), které budou zobrazeny skladníkovi během plnění úlohy.
 - 8.1.1.3. Potvrzením formuláře vytvoří úlohu, čímž ji uvolní ke zpracování skladníkem.
 - 8.1.2. Vyskladnění
 - 8.1.2.1. Vedoucí vyplní
 - 8.1.2.1.1. majitele, kterého se úloha bude týkat,
 - 8.1.2.1.2. zdrojové umístění ve skladu
 - 8.1.2.1.3. cílové umístění pro logistiku (pokud je modul logistiky aktivován)
 - 8.1.2.1.4. určí prioritu úlohy
 - 8.1.2.1.5. Určí zákazníka, pro kterého má být zboží vyskladněno.
 - 8.1.2.2. Specifikuje které a kolik zboží má skladník vyskladnit.
 - 8.1.2.3. Volitelně může zapsat doplňující informace (pokyny), které budou zobrazeny skladníkovi během plnění úlohy.
 - 8.1.2.4. Potvrzením formuláře vytvoří novou volnou úlohu vyskladnění.
 - 8.1.3. Přeskladnění zboží (sestěhování)
 - 8.1.3.1. Vedoucí vyplní
 - 8.1.3.1.1. cílové umístění,
 - 8.1.3.1.2. majitele, kterého se úloha bude týkat,
 - 8.1.3.1.3. určí prioritu úlohy
 - 8.1.3.1.4. určí preference na volbu zdrojového umístění.
 - 8.1.3.2. Specifikuje které a kolik zboží má skladník přesunout.
 - 8.1.3.3. Volitelně může zapsat doplňující informace (pokyny), které budou zobrazeny skladníkovi během plnění úlohy.
 - 8.1.3.4. Potvrzením formuláře vytvoří novou volnou úlohu.
 - 8.1.4. Rozmístění
 - 8.1.4.1. Vedoucí vyplní
 - 8.1.4.1.1. zdrojové umístění
 - 8.1.4.1.2. majitele, kterého se úloha bude týkat,
 - 8.1.4.1.3. určí prioritu úlohy,
 - 8.1.4.1.4. Určí preference na volbu cílových umístění.

- 8.1.4.2. Specifikuje které a kolik zboží má skladník přesunout.
 - 8.1.4.3. Volitelně může zapsat doplňující informace (pokyny), které budou zobrazeny skladníkovi během plnění úlohy.
 - 8.1.4.4. Potvrzením formuláře vytvoří novou volnou úlohu.
- 8.1.5. Přezásobení (Seskladnění + rozmístění)
 - 8.1.5.1. Vedoucí vyplní
 - 8.1.5.1.1. majitele, kterého se úloha bude týkat
 - 8.1.5.1.2. určí prioritu úlohy.
 - 8.1.5.2. Specifikuje které a kolik zboží má skladník najít a přesunout.
 - 8.1.5.3. Volitelně může zapsat doplňující informace (pokyny), které budou zobrazeny skladníkovi během plnění úlohy.
 - 8.1.5.4. Potvrzením formuláře vytvoří novou volnou úlohu.
- 8.1.6. Inventura
 - 8.1.6.1. Vedoucí vybere jakého skladu a čeho se inventura týká. Konkrétně může jít o inventuru dle majitele, výrobce, výrobku nebo jednoho či více umístění.
 - 8.1.6.2. Určí prioritu úlohy.
 - 8.1.6.3. Potvrzením formuláře vytvoří volnou úlohu - inventura.
- 8.1.7. Přesun mezi sklady
- 8.2. Zobrazení probíhající úlohy
 - 8.2.1. Vedoucí vidí seznam nedokončených úloh. Kliknutím na jednu položku seznamu, zobrazí detailní náhled úlohy.
 - 8.2.2. V rámci náhledu vidí
 - 8.2.2.1. kdo úlohu zadal,
 - 8.2.2.2. kdo ji zpracovává,
 - 8.2.2.3. doplňující informace k úloze,
 - 8.2.2.4. průběžný stav (jaké množství zboží je již zpracováno a jaké zbývá, jak dlouho danou úlohu skladník zpracovává).
 - 8.2.3. Pokud není se současným plněním úlohy dostatečně spokojen, může úlohu přeargovat na jiného skladníka, nebo přesunout do volných úloh.
- 8.3. Schválení úlohy
 - 8.3.1. Schválení dodávky zboží
 - 8.3.1.1. Vedoucí vidí informace o přijaté dodávce (datum a čas přijetí, jméno skladníka, který dodávku přijal, číslo dodacího listu, číslo faktury a jméno dodavatele).
 - 8.3.1.2. Vyplní, kterého majitele se naskladnění týká, určí prioritu úlohy a nakonec určí, na které umístění má být zboží umístěno. Pokud si volbou vhodného umístění není jistý, může přenechat volbu na skladníkovi, nebo na rozhodnutí aplikace.
 - 8.3.1.3. Volitelně může zapsat doplňující informace (pokyny), které budou zobrazeny skladníkovi během plnění úlohy.
 - 8.3.1.4. Pokud není s dodávkou spokojen, může ji odmítnout. V opačném případě vytvoří volnou úlohu naskladnění.

- 8.3.2. Obecně (platí pro schválení všech typů úloh)
 - 8.3.2.1. Vedoucí vidí informace o tom, co se v rámci úkolu dělo, kdo jej zpracovával, informace (pokyny), které zadal skladníkovi, a poznámky od skladníka.
 - 8.3.2.2. Úlohu může buď schválit, pokud je s řešením spokojen, nebo ji vrátit k přepracování stejnému nebo jinému skladníkovi. Pokud úlohu vrátí k přepracování, může k ní připojit komentář s dodatečnými instrukcemi.
- 8.3.3. Schválení naskladnění
 - 8.3.3.1. Vedoucí může vyplnit nákupní cenu jednotlivých položek.
- 8.3.4. Schválení vyskladnění
 - 8.3.4.1. Následně se automaticky vytvoří úloha expedice.
- 8.4. Zobrazení uzavřené úlohy
 - 8.4.1. Vedoucí vidí seznam uzavřených úloh. Kliknutím na jednu z nich zobrazí podrobný náhled.
 - 8.4.2. V rámci náhledu vidí, kdo úlohu zadal a kdo ji zpracoval, s jakým zbožím bylo manipulováno a doplňující informace. Pokud se během plnění úlohy objevil nějaký problém, vidí popis tohoto problému, který vyplnil skladník, a vidí, kterého zboží se problém týkal.
 - 8.4.3. U úloh naskladnění a vyskladnění může stáhnout příjemku nebo výdejku ve formátu PDF. Případně může stáhnout souhrn zboží, se kterým bylo manipulováno, ve strojově čitelném formátu (CSV, XLS).
- 9. Přehledy
 - 9.1. Počty kusů zboží dle umístění / majitelů / výrobců / skladů / na cestě mezi sklady (včetně/bez cen)
 - 9.2. Umístění určená pouze pro dané majitele
 - 9.3. Pohyby na skladě
 - 9.4. Stav Logistiky
 - 9.5. Logistické náklady (materiál, čas...)
 - 9.6. Výkon skladníků
 - 9.7. Historie skladu
 - 9.7.1. Podobně jako je na eshopu
 - 9.8. A další...

Angler

Jednotný autorizační server pro komponenty Octopus API.

Požadavky

1. Evidence aplikací

Systém bude evidovat seznam aplikací, které mohou daný autorizační server využívat. U každé aplikace bude určeno, zda obdrží přístupový token na základě ověření přihlašovacích údajů uživatele systému, nebo dle ověření soukromého klíče aplikace. U aplikací, které nedisponují přihlašovacími údaji uživatelů, budou navíc evidovány privilegia.

2. Evidence privilegií

Systém bude evidovat seznam privilegií, které budou sloužit pro řízení přístupu k různým zdrojům v rámci aplikací. Privilegia mohou být sdílená mezi několika aplikacemi, nebo mohou být určena výhradně pro řízení přístupu v jedné aplikaci.

3. Evidence uživatelů

Systém bude evidovat uživatele - jméno, uživatelské jméno, heslo, privilegia a případě další volitelné parametry.

4. Poskytování a validace přístupových tokenů

Systém bude registrovaným aplikacím poskytovat přístupové tokeny. Zároveň bude poskytovat rozhraní pro jejich validaci, obnovu či pro získání informací o již existujících přístupových tokenech.

5. Programové rozhraní pro vzdálenou správu uživatelů

Systém bude umožňovat *vzdálenou* správu uživatelů (z jiných aplikací) pomocí aplikačního rozhraní. Toto rozhraní bude poskytovat služby pro úpravu či tvorbu nových uživatelů a bude přístupné pouze uživatelům, jenž budou mít dostatečná privilegia.

6. Webové rozhraní pro správu aplikací, privilegií a uživatelů

Systém bude poskytovat jednoduché webové rozhraní pro registraci a správu aplikací, tvorbu nových a úpravu stávajících privilegií a uživatelů.

Technologie

- OAuth2 server s podporou introspekce (<https://tools.ietf.org/html/rfc7662>) a Authorization Code Grant s PKCE (<https://tools.ietf.org/html/rfc7636>)
- Backend
 - Symfony aplikace využívající <https://oauth2.thephpleague.com>
 - PostgreSQL databáze + Doctrine ORM
- Frontend
 - Vue, Material

Zápis z uživatelského testování aplikace během vývoje

Seznam požadavků

- Na klik na šedý button formuláře ukázat, co vlastně chybí vyplnit,
- výchozí akce tabulky by měla být aktivní na celém řádku,
- karty na domovské obrazovce řadit rovnou pod sebe - jako v Google Keep,
- tabulky jsou moc řádné, moc splývají,
- přidat k úpravě skladu, co znamená volba "malý sklad",
- přidat ke skladu atribut "popis",
- v malém skladu povolit více umístění,
- přidat skladníkovi možnost "Rychlý přesun mezi umístěními" ,
- na tlačítka která pouze něco rozbalují a nepřidávají, dát jinou ikonku než "+",
- doplnit popisky, že umístění je fyzické a části skladu jsou virtuální,
- přejmenovat "první skladová položka" a "první vlastník" na reálnější údaje,

- přidat možnost "vytvořit nové" i přímo do tabulek, nejen v rohu obrazovky,
- přidat do načítání dat o subjektu nejen základní data z ARES, ale i bankovní účet, plátce DPH, nespolehlivý plátce,
- ve formulářích, kde se vyplňuje i IČO, ho mít na první místě, aby to uživatel vyplňoval jako první a zbytek jen načetl,
- k vlastníkovi skladu přidat pole "oddělení",
- backend by měl vypočítávat IBAN z čísla účtu,
- zákazník má mít možnost mít více doručovacích adres,
- veškeré inputy typu textarea dát jako rámeček, ne jen linku,
- naskladnění - skladník:
 - přidat ke kusům "ks",
 - umožnit zobrazení většího náhledu fotky,
 - výběr počtu štítků k tisku,
 - řešit, že je zboží s výhradou (poškozený obal atp.),
 - skladník možnost přidávat fotky,
 - k položkám ukládat jednotky, ty pak vypisovat v naskladnění (ks, litrů, kg, ...),
- naskladnění - vedoucí (přehled hotového):
 - umožnit rychlý náhled výrobku,
 - zobrazovat fotku dodáku,
 - hledání v přehledu naskladněných položek,
- v záložkách na domovské obrazovce zobrazovat počty úkolů v jednotlivých seznámech,
- zvýraznit priority (možná udělat barevně i ikonky),

-
- přidat vedoucímu možnost vytvořit samotné naskladnění (bez dodávky),
 - inventura:
 - našeptávat umístění k napípaní,
 - zabalit detaily umístění, které nejsou aktivní,
 - přidat autoscroll na napípnuté umístění,
 - přidat fotku výrobku,
 - položky na umístění zalamovat pod sebe,
 - vedle názvu zobrazovat i EAN a kolikakusový EAN to je,
 - přidat stránku přehledu inventury, která bude mít stejný obsah, jako se tiskne do PDF reportu.

Seznam chyb

- nelze změnit název atributu (PUT mění jen value),
- v příjmu dodávky se nevypisuje, kdo ji přijal, je tam "nikdo",
- nelze dokončit naskladnění,
- při odmítnutí řešení úkolu se špatně odesílají poznámky.

Seznam otázek ke zjištění

- Mají tabákové výrobky jiný EAN pro jiné šarže? (možná mění jen kolky),
- zjistit, jak se dá na dodávky tisknout datová informace o položkách (QR kód s linkem na api),
- jak se řeší vyskladňování šaržovaných výrobků.

Návrhy na pokročilé funkce

- Kategorie produktů,
- rozpis vytíženosti skladu: k tomu je potřeba řešit počet pozic na umístění, blokace pozic,

- routování trasy skladníka: nejdřív naber velké věci, pak teprve ty malé,
- řešit umístění u země (upozorňovat, že jsou u země výrobky, které se netočí),
- řešit systémově výrobky, které zabírají více palet (např. 3),
- hromadný aktualizátor dat z Aresu s výběrem diffu,
- příprava na dodávky zboží: vedoucí může zadávat skladníkům avizace dodávek = příprava následujícího úkolu naskadnění,
- skupiny "typů" skladníků + možnost pro vedoucího přiřazovat úkoly přímo skladníkům / skupinám skladníků,
- pokročilé nastavení domovské obrazovky: možnost nastavení množství informací na kartách úkolů,
- víceúrovňový konfigurovatelný Dashboard,
- nový úkol "kompletní inventura skladu": vychází z dokončené jiné inventury, nabízí k inventurizaci pouze to, co je např. jen na špatném umístění, zobrazuje, kde se chybějící zboží pohybovalo,
- nastavování práv skladníků:
 - možnost tvořit nové položky?,
 - možnost tvořit nová umístění?,
 - pokud nemá právo a potřebuje to udělat, tak možnost "zavolej vedoucího".

Obsah přiloženého média

README.md	stručný popis obsahu média
src	
├── thesis	zdrojové soubory textu práce
├── NUR-SYSEL.rpprj	zdrojové soubory Lo-Fi prototypu
├── mi-nur-sysel-master.zip	zdrojové soubory Hi-Fi prototypu
└── processes.pdf	Návrh procesů skladového systému
DIP_Malec_Oldrich_2019.pdf	text práce ve formátu PDF