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 softwarového inženýrství

Diplomová práce

Frontend skladového systému

Bc. Oldřich Malec

Vedoucí práce: Ing. Jiří Hunka

1. ledna 2020

Poděkování

Na těchto řádcích bych chtěl poděkovat všem, bez kterých by tato práce nemohla vzniknout, nebo by nebyla v odpovídající kvalitě. V první řadě se jedná zejména o vedoucího práce - Ing. Jiřího Hunku, který mi poskytl dostatečnou volnost při její tvorbě, ale zároveň zařídil nejdůležitější kroky k jejímu zdárnému vytvoření - zejména velkou zpětnou vazbu ve fázi analýzy a návrhu, a ke konci realizace například zařízení testerů v reálných skladech. Stejně důležité díky patří i mému kolegovi - Ing. Pavlu Kovářovi, který je autorem backendové části řešené aplikace, a bez jehož API by byla má práce pouze jakousi nefunkční šablonou, protože jak řekl klasik[1] - „díky frontendu je to hezké, ale díky backendu to funguje“. Pavlova dokumentace API tak pro mě byla dokonalým zdrojem informací, jaké funkce musím ještě pokrýt a jak se mají chovat, a právě díky tomu jsem se při implementaci mohl soustředit opravdu pouze na frontendové řešení.

S počátky této práce mi velmi pomohl také tým z MI-NUR[2], který se kromě mě skládal právě z Ing. Pavla Kováře, Ing. Martina Kubiše a také Bc. Jakuba Štercla - všem tímto moc děkuji, že se podíleli na návrhu frontendu mé diplomové práce. Prototyp, který v rámci MI-NUR vznikl, testovalo celkem osm osob, a přestože já osobě jsem se setkal pouze s dvěma z nich, všichni si zaslouží poděkování - konkrétně se jedná o Barbaru Novotnou, Kristýnu Miltnerovou, Davida Zahrádku, Lukáše Svobodu, Marka Erbena, Milana Kubiše, Pavla Štercla a Václava Kováře.

[1] Marek Erben

[2] Návrh uživatelského rozhraní - předmět na FIT.

V závěru práce jsem opět oslovil několik testerů, kteří již testovali funkční aplikaci na reálném zařízení, ve svém přirozeném prostředí. I ti si zaslouží velké díky, avšak při testování jsme se domluvili, že nebudu zveřejňovat jejich jména. Pokud se jim však někdy tento text dostane do ruky, oni budou jistě vědět - děkuji i Vám!
Nelze opomenout ani jazykovou a stylistickou korekturu tohoto textu, bez které by práce obsahovala spoustu gramatických chyb, a za kterou vděčím Bc. Markétě Malcové a Bc. Kristýně Miltnerové, také děkuji!

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. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 1. ledna 2020 ……

České vysoké učení technické v Praze

Fakulta informačních technologií

Odkaz na tuto práci

MALEC, Oldřich. *Frontend skladového systému.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020. Dostupný také z WWW:
⟨`https://gitlab.fit.cvut.cz/malecold/master-thesis`⟩.

Abstrakt

Práce se zaměřuje na kompletní proces analýzy, návrhu, vývoje a testování frontendu webové aplikace - skladového systému, s důrazem na praktické použití. Nový systém staví na požadavcích již existující aplikace a klade si za cíl implementovat veškeré její funkcionality ale také další rozšíření, která již není efektivní zapracovávat do staré verze. Návrh uživatelského rozhraní aplikace staví na projektu z předmětu MI-NUR vyučovaného na FIT ČVUT v Praze. Výsledná reálná aplikace využívá Javascriptový framework Vue.js a grafickou knihovnu Vuetify. Frontendová část aplikace, která je předmětem této práce, konzumuje REST API vytvořené v souběžné diplomové práci. V rámci práce byl vytvořen základ nového skladového systému, na kterém je možné dále stavět a rozvíjet jeho funkcionality. Z textu lze vycházet při návrhu obdobných aplikací, zejména co se týká volby použitých technologií a knihoven.

Klíčová slova frontend, webová aplikace, Javascript, Vue.js, Vuetify, skladový systém

Abstract

This thesis focuses on a complete process of analysis, design, implementation, and testing of web-application frontend - storage management system, with emphasis on practical use. The new system builds on the requirements of an existing application, aiming to provide all its current features and some new ones, which are not efficient to incorporate into the old version. The user interface design builds on a project from MI-NUR, a subject taught at FIT CTU in Prague. The final version of the real application utilizes a Javascript framework Vue.js and a graphical library Vuetify. The frontend part of the application, which is a subject of this thesis, consumes a REST API created in a concurrent diploma thesis. As part of this work a base of a new storage management system was created, which is suitable for more improvements. The text can be used as a basis for similar applications, in particular regarding used technologies and libraries.

Keywords frontend, web application, Javascript, Vue.js, Vuetify, storage management system

Obsah

Seznam ukázek kódu

Seznam tabulek

Seznam obrázků

Úvod

Evidence skladovaného zboží je snad jedním z nejčastějších požadavků na informační systém - je potřeba jak v malých skladech spravovaných jedním člověkem, tak i v obřích skladech, které rostou na okrajích velkých měst. Od skladových systémů většina jejich uživatelů očekává především přehled počtů skladovaného zboží, případně evidenci umístění, ale také snadné napojení na externí aplikace. Aplikací řešících tuto problematiku je na trhu nepřeberné množství, avšak z důvodu potřeby nahradit existující skladovou aplikaci provozovanou společností, ve které působím, do které již není vhodné zahrnovat nové funkcionality, byl vytvořen záměr napsat skladový systém načisto od podlahy.

Již v prvotním návrhu bylo rozhodnuto, že aplikace bude striktně rozdělena na serverovou a klientskou část, čímž vznikla dvě témata diplomových prací, z nichž klientskou část řeším právě v rámci tohoto textu. Má motivace pro vytvoření frontendové části nového skladového systému spočívá v zájmu o webové technologie a chuť naučit se pracovat s moderním Javascriptovým frameworkem.

V textu projdu postupně všechny body standardního vývoje aplikace, počínaje analýzou požadavků a konkurenčních řešení, návrhem prototypu a jeho testováním, přes implementaci reálné aplikace, která začíná dlouhou diskusí volby technologie a pokračuje popisem některých klíčových částí kódu. Poslední kapitola obsahuje zápisy a postřehy z uživatelského testování, ze kterého vzešly relevantní požadavky na úpravy, z nichž ty nejpalčivější jsem ještě v rámci této práce zapracoval.

Samotný text je hojně doplněn přílohami, které obsahují rozpracované analýzy, úkoly pro testery či zápisy z testování.

Kapitola 1

Analýza

Cílem nového skladového systému je nahradit a rozšířit funkce stávajícího skladového systému Sysel - proto jsme při analýze požadavků vycházeli[3] 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

[3] Já 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 velikosti 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 ukázku přikládám screenshot úryvku

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 - role skladníka sice typicky na počítači opravdu používána nebude, ale vedoucí skladu by měl mít možnost volně přecházet mezi různými zařízeními.

1.3 Analýza konkurence

Před započetím tvorby nových návrhů UI je vhodné analyzovat existující konkurenční řešení.
V dnešní době existuje na trhu nespočet skladových systémů, jednak veřejně dostupných, které nabízejí buď osekanou verzi zdarma, nebo jsou placené, ale zato mají dostupný popis jejich vlastností, a pak také spousta uzavřených systémů, které si nechal 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 funkcinality 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 varianta zdarma, 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, vyskladnění,

přesun, inventura. Navíc umožňuje pracovat i s objednávkami.
Synchronizace probíhá přes úložiště třetí strany (Dropbox, ...) a to vždy pouze při spuštění či ukončení a nebo na přímé vyžádání uživatelem. Jedná se tedy o systém určený především jedné osobě, neboť při použití ve více lidech současně by mohlo docházek ke kolizím v synchronizovaných verzích. 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 například celou paletu, na které je různé zboží. U každé položky se znovu vyplňuje celý formulář přesunu.

Zajímavé funkcionality:

- Sken čárových kódů běžným fotoaparátem zařízení.
- Možnost konfigurace, které prvky se zobrazují na domovské obrazovce.
- Možnost konfigurace, které atributy skladových položek či objednávek se mají zobrazovat.

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.
- Přepínání aktivních polí formulářů logicky přeskakuje na další, na nových obrazovkách je většinou jako výchozí zvolena položka *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 neotevírá automaticky klávesnici, je tedy stejně nutné znovu do pole tapnout.
- Aplikace nemá menu. Tudíž když se uživatel zanoří hlouběji, musí vícekrát použít tlačítko *zpět*, aby se dostal na domovskou obrazovku.
- Při hromadném vyskladňování nelze pracovat „z jednoho místa“ - u každého produktu se vždy volí znovu umístění - nezůstává ani předvyplněné.
- Celkově se jedná spíše o jednoduchý seznam potřebných informací, UI aplikace není pro uživatele nápomocné ve smyslu navádění, co má být provedeno dále.

1.3.1.2 Simple Stock Manager

Tato aplikace je také dostupná na Google Play zdarma, avšak narozdíl od předchozí testované, tato nenabízí možnost zakoupení plné verze a vždy obsahuje reklamy.
Nabízí pouze základní funkcionalitu naskladnění a vyskladnění (neumí tedy ani evidovat umístění, zdaleka neumí role, synchronizaci, úkoly, dodací listy, faktury, ...).
Provedené transakce lze zpětně upravovat - pro použití této aplikace je to vhodné (je určena pro použití pouze na jednom zařízení), pro nově navrhovaný skladový systém je to funkce nevhodná, neboť je nutné mít přehled o všech pohybech ve skladu - veškerá akce by měla jít vrátit pouze vytvořením nové akce, která bude opakem té první, a obě zůstanou evidovány samostatně.

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. Uživatel se tak vždy jednoduše dostane tam, kam potřebuje.
- 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 její potenciál nevyužitý.

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 rozmýšlet a všechny automaticky potvrzuje, čímž dialog ztrácí smysl [47].
- Chybí auto focus při otevření nových stránek na důležité prvky (vyplnění EAN, hledání…).
- Ovládací prvky UI (potrvzení atp.) jsou často malé a špatně se na ně na dotykové obrazovce trefuje.
- 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 poloviny obrazovky, jinak se menu neotevře.
- V seznamu zboží je pro otevření detailu nutné kliknout na malou akci „Show“, klik na celý řádek produktu nedělá nic.

1.3.1.3 Vyhodnocení provedené analýzy konkurence.

K bodům, které jsem v rámci analýzy konkurence sepsal, jsem se následně několikrát vrátil při návrhu a implementaci reálné aplikace, abych neopakoval zde nalezené chyby, nebo naopak zapracoval do svého řešení i nalezené kladné stránky.

Kapitola 2

Návrh, realizace a testování 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. Tento tým byl vytvořen přímo za účelem realizace prototypu skladového systému, na kterém jsem následně chtěl dále pracovat již v rámci této diplomové práce. Proto jsem byl zvolen vedoucím týmu, staral se o kvalitu našeho týmového výstupu a sledoval pozorně i části, které zpracovávali mí kolegové, protože jejich výstup přímo napomáhal vyšší kvalitě mé navazující práce. V rámci týmu jsme se pro zjednodušení zaměřili především na pohled role skladníka.

První částí návrhu GUI byl mockup, či chcete-li wireframe. Pro jeho tvorbu jsme využili nástroj Axure RP [52] - 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 [69], což je knihovna pro Axure, která nabízí komponenty v *Material designu*. S její pomocí jsme vytvořili návrhy na základní procesy skladníka.

Obrázek 2.1: Domovská obrazovka aplikace: Lo-Fi prototyp

Domovská obrazovka. Právě ve fázi návrhu Lo-Fi prototypu jsem rozhodl o novém základním rozložení obrazovek v systému - vertikální řazení veškerých položek, které již bylo znázorněno v předchozí kapitole na obrázku 1.1 bylo přepracováno do více pohledů, které se na velké obrazovce mohou zobrazovat vedle sebe, na mobilním zařízení pak mezi nimi bude možné přecházet posunem - zárodek tohoto řešení je vidět na obrázku 2.1, jedná se o texty „Přehledy“, „Volné“ a „Všechny“ v horní části obrazovky. Obsah, který se bude na jednotlivých pohledech zobrazovat, by mohl být konfigurovatelný a uživatel by si tak mohl nastavit přesně ty seznamy, které ho zajímají. Kromě toho je také na vložené ukázce vidět první verze znázornění priorit - levý okraj karet úkolů nabývá zelené, žluté a červené barvy podle nastavené priority úkolu.

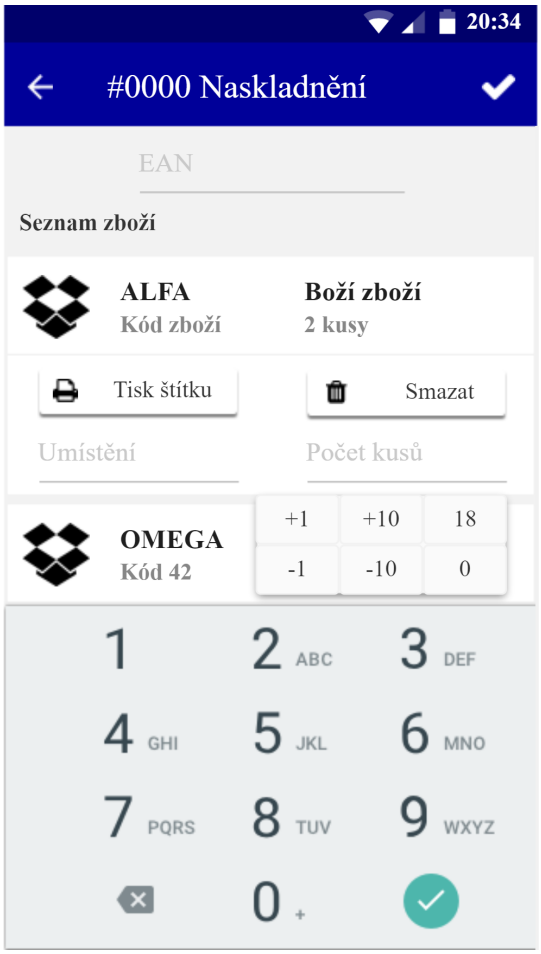
Obrázek 2.2: Rozhraní skladníka pro naskladňování položek: starý skladový systém

Naskladnění. Další ukázkou z této fáze je obrazovka naskladnění - do textu je vložena jako obrázek 2.3, pro porovnání je vložena i ukázka starého skladového systému - obrázek 2.2. V novém systému je v této fázi vidět ještě poměrně silná inspirace rozpoložení ve starém systému a návrh je tak spíše pouze oblečen do modernějšího designu. Dokončení úlohy je v pravém horním rohu, kde je sice stále na očích, ale, jak jsem zjistil později a také při zpětném pohledu na analýzu konkurenčních řešení, toto řešení není příliš vhodné kvůli své nevýraznosti, a tak bylo ještě v další iteraci přepracováno.

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

Zadávání počtu kusů. Novinkou, která se objevila již v Lo-Fi prototypu je snadnější zadávání počtu kusů zboží, pro které jsem vložil samostatnou ukázku jako obrázek 2.4, umožňující jak přímo zadání počtu kusů, tak i možnost přeskakovat po jednom nebo deseti kusech, ve směru nahoru i dolů. Tento výběr se zobrazuje při aktivním poli „Počet kusů“, které bohužel na vloženém obrázku nevypadá aktivně, protože to Lo-Fi prototyp nepodporoval.

Kompletní zdrojové kódy Lo-Fi prototypu jsou k dispozici na přiloženém médiu. Lo-Fi prototyp byl zkontrolován odbornou osobou a na základě získané zpětné vazby proběhla další iterace - návrh Hi-Fi prototypu.



Obrázek 2.4: Naskladňování - rychlé zadání počtu kusů zboží: Lo-Fi prototyp

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.
Hi-Fi prototyp jsme stále 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 použitá 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 já jsem chtěl pro naše řešení mít co nejméně závislostí, rozhodl jsem se vše napsat v čistém Javascriptu (pouze za použití dnes de facto standardní knihovny jQuery, která se dá jednoduše načíst z CDN). Základem je jeden výchozí HTML soubor, viz ukázka kódu 2.1.

```
<!DOCTYPE html>
<html>
<head>
  <title>SYSEL</title>

  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta charset="UTF-8">

  <!-- Material components all-in-one -->
  <link rel="stylesheet" type="text/css" href="https://unpkg.com/
  material-components-web@latest/dist/material-components-web.min.css"/>
</head>
<body>
  <div id="main-content" class="homepage-include"></div>

  <!-- Material design -->
  <script src="https://unpkg.com/material-components-web@latest/dist/
  material-components-web.min.js"></script>
  <!-- jQuery-->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/
  2.2.4/jquery.min.js"></script>
  <!-- Custom scripts -->
  <script src="js/main.js"></script>
</body>
</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.

```
function loadDynamicContent(selector = '[class$="-include"]', callback){

  $(selector).each(function (e) {
    // load all custom components
    let parent = $(this);
    $(this).html('');
    $(this).load("components/" + $(this).attr("class")
      .replace("-include", "") +  ".html", function() {

      // load sub-components
      $(this).find('[class$="-include"]').each(function() {
        loadDynamicContent(this, function(){
            insertParameters(parent)
        });
      });
      if ($(this).find('[class$="-include"]').length === 0) {
        insertParameters(parent);
      }
      callback(this);
    });
  });
}

function insertParameters(parent) {
    let parameters = parent.attr('data-include-content');
    if (parameters !== undefined) {
        $.each(JSON.parse(parameters), function(key, value){
            parent.html(parent.html().replace('{' + key + '}', value));
        }
    }
}
```

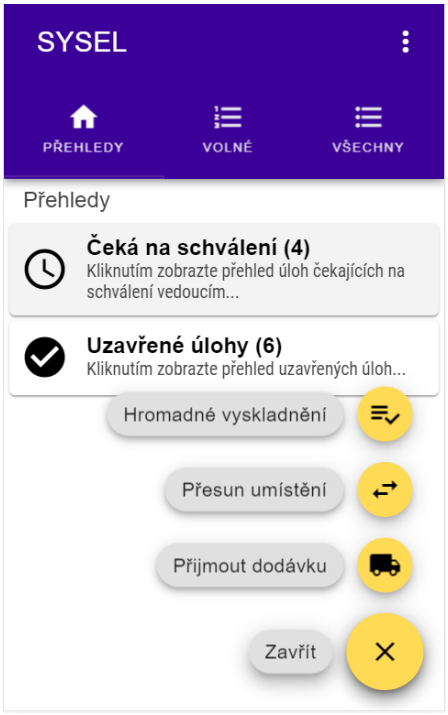
Ukázka kódu 2.2: Funkce pro načtení obsahu komponent Hi-Fi prototypu, včetně podkomponent

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 `foo-include` nahrazuje obsahem souboru `foo.html` ze složky `components/`. Díky tomu, že Javascriptu může být umožněn přístup na filesystém, lze takto dynamicky načítat obsah souborů z disku a renderovat ho přímo v již načteném

obsahu. Tato technika je v některých browserech zablokována kvůli CORS[4], a tak pro použití například v Google Chrome je nutné zmiňovaný `index` spouštět přes libovolný webserver, nelze jej přímo otevřít z filesystému. Naopak například v Mozilla Firefoxu je CORS v tomto ohledu volnější, a celý tento mikroframework tak lze používat bez jakéhokoliv lokálního serveru, stahování závislostí atp. - stačí pouze otevřít `index.html`.

2.2.2 Realizace Hi-Fi prototypu

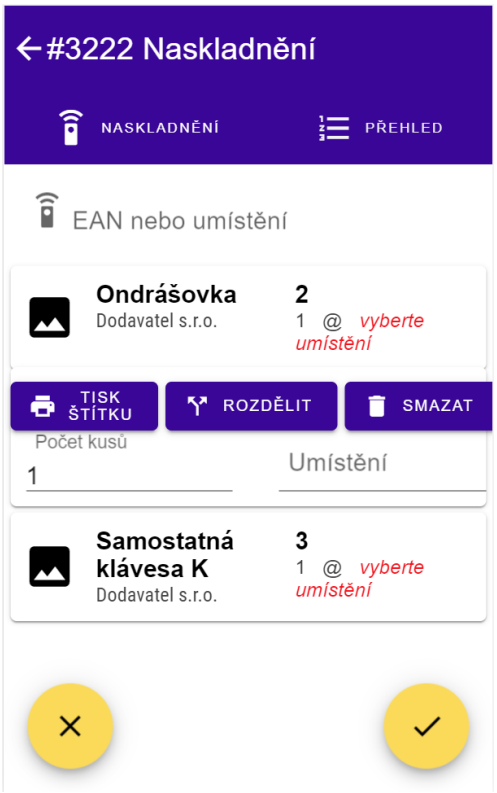
Podobně jako přípravu technologie, i některá důležitá rozhodnutí týkající se designu UI jsem navrhoval samostatně bez týmu a budu je proto probírat v textu níže.
Kompletní zdrojové kódy prototypu jsou dostupné na přiloženém médiu a také v repozitáři na Gitlab FIT [11].



Obrázek 2.5: Rychlé menu v pravé dolní části obrazovky: Hi-Fi prototyp

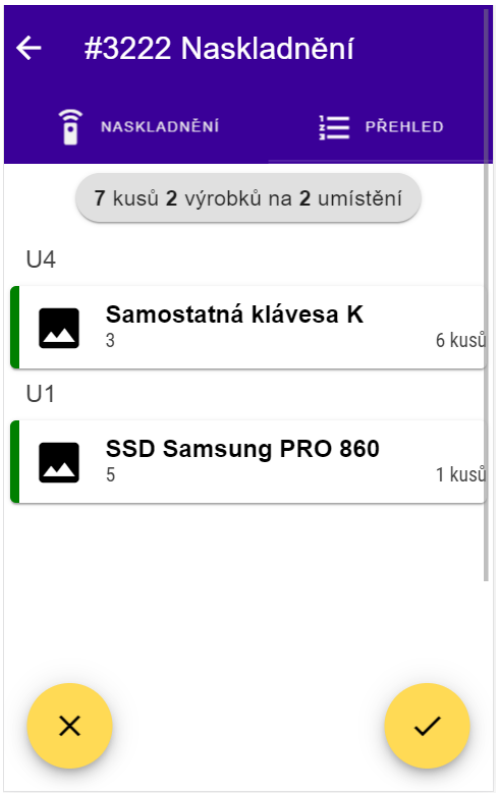
[4]Cross-origin resource sharing - ochrana před načítáním nezabezpečeného obsahu

Rychlé menu. Ve druhém prototypu jsem poprvé použil akční tlačítko plovoucí v pravém dolním rohu aplikace, přes které se tvoří nové položky. Je-li na stránce možné tvořit více různých věcí, otevře se z něj menu, jaké známe převážně z Google aplikací - jak je znázorněno na obrázku 2.5. Toto tlačítko již v aplikaci vydrželo i do reálné verze a jak na něj uživatelé reagovali se dočtete v sekci 5.2.



Obrázek 2.6: Rozhraní skladníka pro naskladňování položek: Hi-Fi prototyp

Naskladnění. Stejně jako u předchozí verze prototypu, i nyní je do textu vložena ukázka obrazovky naskladnění - je to z toho důvodu, že většina úkolů má ve svém nitru obdobný průběh, což vzešlo z analýzy v sekci 1.1. V této verzi prototypu je však již úkol ve svém detailu rozdělen na dvě obrazovky, mezi kterými je - stejně jako na domovské obrazovce - možné přejet posunutím nebo kliknutím v záhlaví aplikace, čímž je pro uživatele snadnější zobrazit si vždy tu informaci, kterou hledá - v případě úkolu naskladnění se jedná o rozdělení na samotnou realizaci *naskladňování* (obrázek 2.6) a na *přehled* (obrázek 2.7). I na



Obrázek 2.7: Rozhraní skladníka pro naskladňování položek - dokončení: Hi-Fi prototyp

tyto obrazovky se dostalo akční tlačítko, navržené pro zobrazení menu, avšak zde mělo funkci dokončení, případně odmítnutí úkolu, což se v uživatelském testování projevilo jako matoucí, a tak byla tato tlačítka - viditelná ve spodní části obrazovek - v reálné aplikaci odstraněna a nahrazena běžnými tlačítky s popisky, která se zobrazují na stránce *přehled*.

2.3 Testování Hi-Fi prototypu

Vzniklý Hi-Fi prototyp byl testován v rámci předmětu MI-NUR společně mnou, Pavlem Kovářem, Martinem Kubišem a Jakubem Šterclem. Abych nevyužíval 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 (screenshoty 2.6 a 2.7), 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 úkolu je jeho řešitel ihned nastaven na právě přihlášeného uživatele. 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 odešlou, nebo se teprve zobrazí nějaký formulář a potvrzení.[5]

4. *Shoda s použitou platformou a obecnými standardy*:
   V pořádku (material design).

5. *Prevence chyb*:
   Úkol nelze dokončit, pokud některé zboží nemá vyplněno umístění.[6] Jiné chyby na této stránce (z aplikačního hlediska) prakticky nelze udělat => OK.

6. *Kouknu a vidím*:
   Při větším množství položek může být seznam dlouhý a nepřehledný.

7. *Flexibilita a efektivita*:
   OK - zkušený může stále pípat, nový může zkoumat dostupné vstupy.

8. *Minimalita (Klapky na očích)*:
   OK - obsah karet bude ve finální aplikaci konfigurovatelný[7].

[5] V 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.

[6] V reálné aplikaci byla kompletně otočena logika označování umístění: nejprve je nutné zvolit umístění, a pak na něj teprve ukládat zboží. Popsaná situace tedy nemůže nastat.

[7] V době psaní textu se jedná o jeden z návrhů na budoucí vylepšení aplikace

9. *Smysluplné chybové hlášky*:
   Jediná chybová hláška o vyplnění umístění je jasná.

10. *Help a dokumentace*:
    Dokumentace není[8].

2.3.2 Uživatelské testování prototypu

Hi-Fi prototyp aplikace byl již otestován více potenciálními uživateli, pro které jsme připravili seznam úkolů k řešení. Tyto úkoly prováděli na mobilních zařízeních, avšak bohužel bez podpory čtečky čárových kódů, neboť zařízení se zabudovanou čtečkou jsem měl k dispozici pouze jedno a testování probíhalo na více místech současně. Zmiňovaný seznam úkolů a také výstupy testování těch uživatelů, které jsem testoval já osobně, jsou dostupné v příloze D. Problémy nalezené v prototypu byly buďto řádně opraveny ještě v rámci prototypu, nebo s nimi bylo počítáno při realizaci reálné aplikace.

[8]Reálná aplikace již základní nápovědu / dokumentaci nabízí

Kapitola 3

Návrh funkcí a architektury reálné aplikace

Po dokončení návrhu GUI, popisovaného v předchozí kapitole, bylo ještě před započetím implementace potřeba rozhodnout o několika pokročilých funkcích, které vyžadují spolupráci frontendu s backendem. Kromě toho bylo také nutné dokončit soupis procesů pro roli vedoucího skladu, neboť ten byl v rámci MI-NUR přeskočen, zvolit použitou technologii a vybrat barevnou paletu aplikace - všechny tyto body budu probírat na následujících řádcích.

3.1 Návrh pokročilých funkčností aplikace

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 chvilku zastavit.

Potvrzovací dialogy. Všichni je známe - modální okna, typicky s otázkou a dvěma akcemi: *opravdu smazat/odeslat/...* a *storno*. Jsou používány tam, kde je jejich potvrzení většinou nevratné a dojde k provedení něčeho důležitého. Co když je ale provádění důležitých akcí stěžejní funkcí 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 [47].

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 Kovářem zanalyzovali několik možností:

1. **Fronta ve frontendu**: Frontend 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 stanovené 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 se může tvářit, 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.

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, zobrazí 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 druhému zmiňovanému - mění se to, 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 výrobku 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 zavrhli.

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 [70].

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 nepřejí. Pro potřeby undo jednoho klienta by tak musely být provedeny mnohé úpravy, a ty by byly neúměrně náročné. Z důvodu robustnosti potřebných úprav tedy tuto variantu taktéž zavrhujeme.

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 - 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 jsme se rozhodli 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 *napípáno*, systém 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 4.9.

3.1.2 Zkratky v systému za použití čtečky čárových kódů

Nápad na tuto funkci vychází z faktu, že většina skladníků, kteří budou se systémem pracovat, budou mít v ruce zařízení podporující skenování čárových kódů.
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 doraziv̌ších dodávek zboží by mohl být nalepen kód, který v aplikaci spustí úlohu *Příjem dodávky*. V místě, kde se vyzvedávají palety, vozíky či cokoliv jiného, pomocí 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 po načtení kódu stát, by ideálně měl být konfigurovatelný přímo ze systému, aby byl skladový systém použitelný pro různé typy skladů.
V rámci návrhu bylo stanoveno, že kód může být libovolný textový řetězec a aplikace bude zajišťovat možnost načítat kódy kdykoliv - čímž se bude lišit od současného řešení, které dokáže čárové kódy načítat pouze do formulářových polí, do kterých musí uživatel navíc nejprve kliknout.
Výsledný stav návrhu této funkce popisuji v sekci 4.10.

3.2 Návrh procesů v novém skladovém systému

Před započetím samotné implementace bylo nutné dokončit návrh procesů v systému, za jejímž účelem vznikl 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, ho nebudu vkládat přímo do textu, ale je možné jej nalézt jako přílohu E nebo v souboru `processes.pdf` na přiloženém médiu.

3.3 Volba technologie

Jelikož je aplikace rozdělena na backend - řešený v práci Bc. Pavla Kováře [39] a frontend, který je předmětem této práce, je žádoucí věnovat jistou část textu volbě vhodné technologie.

Cílová platforma. Aplikace je navrhována s ohledem na očekávané hardwarové vybavení skladů, na které nová aplikace cílí. Tamní skladníci jsou často 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, 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] [22] front-endové frameworky či knihovny Angular [3], React [54], Vue.js [77], Ember.js [19] a Backbone.js [8].

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

3.3.1 Datum vydání

První dva zmíněné jsou v současnosti nejčastěji porovnávanými frameworky. Vue.js je z této pětice vybraných nejmladší, nabírá ale velké obliby. Ember.js a Backbone.js jsou poté lehce upozaděny z důvodu svého 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

[a] V 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

Angular a React jsou vyvíjeny velkými společnostmi: první Googlem a druhý Facebookem, které zná každý, Ember.js je vyvíjen společností Tilde Inc. [67], která také není žádným startupem. Na druhé straně Vue.js a Backbone.js by se daly nazvat *komunitními projekty*, neboť jsou vytvořeny převážně jedním autorem (Evan You v případě Vue.js a Jeremy Ashkenas v případě Backbone.js) 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 své projekty poměrně často *zabíjejí*, stačí se podívat například na seznam projektů, které ukončil Google [49]. 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 tohoto textu licencováno pod MIT licencí [66], 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 škody potenciálně spáchané tímto softwarem.

Licencování Reactu. Facebook původně vydal svůj React pod BSD licencí [65] spolu s dalšími patenty, avšak 24. září 2017 byl React převeden pod MIT licenci [1, 75].
Jelikož jsou všechny frameworky licencovány stejně, neprobíhá v tabulce 3.3 opět 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 [25], ze které vychází hodnocení v tabulce 3.4.

Ember.js: Tento framework je dle V. Lascika [41] vhodný spíše pro projekty, na kterých pracuje velké množství vývojářů, a to z důvodu své komplexnosti. Proto jej, jako nevyhovující mé vznikající aplikaci, hodnotím pouze jedním bodem.

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 nebo nevhodně. 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
bodový zisk	*1*	*2*	*4*	*1*	*4*

Tabulka 3.4: Volba frameworku: Obtížnost

[a] za 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 [21] 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 [29], která je podobně jako u Angularu velmi bohatá a stromově strukturovaná.

Ember.js: Oficiální manuál Ember.js [20] je taktéž poměrně obsáhlý a strukturou 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
bodový zisk	*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, vyvíjet 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* [53]. Jeho testovatelnost tedy hodnotím jako velmi dobrou.

React: Tato knihovna ve své dokumentaci na první pohled testování příliš neprosazuje, avšak stránku dedikovanou této kratochvíli [64] 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 [36], také je ale zmiňován Enzyme [27] 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 [68] popisuje spouštění Unit testů, pomoci vestavěných příkazů, které interně používají buďto již zmiňovaný Jest nebo Mocha [43], a dále odkazuje na vlastní kompletní testovací knihovnu [31]. Také je možné použít velké množství různých testových frameworků a aplikace ve Vue.js lze vyvíjet i pomocí TDD [40].

Ember.js: Tento framework se ve své dokumentaci věnuje testování poměrně intenzivně a zasvětil mu rovnou několik stránek [28]. 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* [56], která již zmiňuje i zde dříve probírané frameworky jako *Mocha*. Ve výsledku tak bude pravděpodobně testování Backbone vypadat obdobně jako u ostatních frameworků, ale kvůli přístupu oficiální dokumentace to není tak snadné pochopit.

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.

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

Tabulka 3.6: Volba frameworku: Testování

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
bodový zisk	*2*	*4*	*3*	*1*	*0*
Hodnocení (z 5)	3.9	4.2	4.7	4.8	4.5
bodový zisk	*1*	*2*	*3*	*3*	*3*

Tabulka 3.7: Volba frameworku: Devtools

[a] Stav 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
bodový zisk[b]	*2*	*4*	*5*	*0*	*1*

Tabulka 3.8: Volba frameworku: Počet hvězdiček na GitHubu

[a] Stav k 17. 12. 2018

[b] Hodnocení přeskakuje bodový zisk 3, aby bylo zhodnoceno i absolutní množství hvězdiček, nejen pořadí.

3.3.9 Počet npm balíků

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

	Angular	React	Vue.js	Ember.js	Backbone.js
Počet npm balíků	26,6k	73,4k	20,6k	6,4k	1,5k
bodový zisk	*2*	*3*	*2*	*1*	*0*

Tabulka 3.9: Volba frameworku: Počet npm balíků

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 odpovídá a přitom 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 *zodpovězených* otázek	86k	72k	18k	17k	16k
bodový zisk	*3*	*3*	*1*	*1*	*1*

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 je tento projekt realizován. Firma se specializuje především na webové aplikace a middlewary na zakázku [32], 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
bodový zisk	*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
bodový zisk	*4*	*4*	*2*	*1*	*2*

Tabulka 3.12: Volba frameworku: Počet vývojářů na LinkedIn

3.3.13 Integrace se Sentry

Sentry [59] 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 [34], 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	13	13

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.

3.4 Volba barevné palety aplikace

Barvy použité v aplikaci mají velký vliv na to, jak ji budou její uživatelé vnímat. Důležitý je jak kontrast barvy textu a pozadí, tak i volba barev akčních či informativních prvků.

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 [72], 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.
  - *Dekorativní, 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.
  - *Dekorativní, 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.

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 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]:

Obrázek 3.1: Barevná paleta aplikace: Trojúhelníkové schema

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.

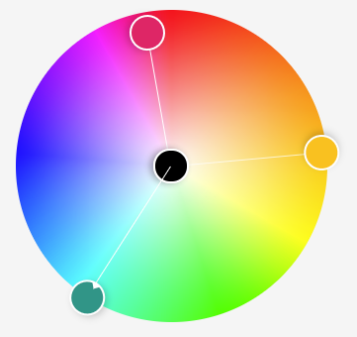
Obrázek 3.2: Barevná paleta aplikace: Rozděleně-doplňkové schema

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 3.3: Barevná paleta aplikace: Analogové schema

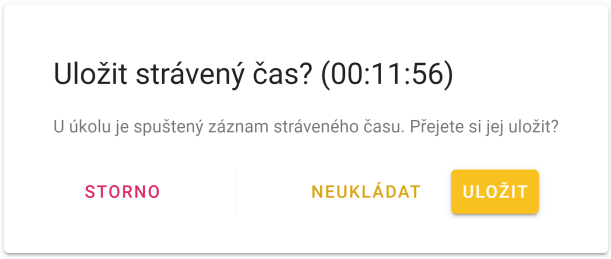
3. *Analogová*: Zvolené barvy jsou v barevné paletě přímí sousedé a jedná se tak o „klidné“ rozložení, které vytváří jemný design, jenž je často vidět i v přírodě. Problémem zde může být vhodná volba kontrastu, protože barvy nejsou samy vůči sobě příliš kontrastní a je nutné je od sebe alespoň trochu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

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 vyžadovaly jeho zadávání. Zbyla 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 3.4. Společně tyto barvy tvoří něco mezi dvěma zvolenými schematy - trojůhelník není ani pravidelný, ani se neblíží k doplňku.

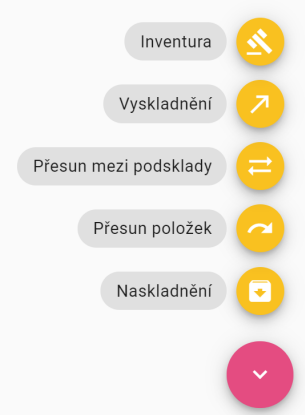


Obrázek 3.4: Barevná paleta aplikace: Zvolené schema pro mou aplikaci

Barvy a idempotence akce. Jelikož mám vybrané dvě akcentní barvy, musel jsem se rohodnout, 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



Obrázek 3.5: Barvy znázorňující idempotenci akcí: Dialog běžícího času



Obrázek 3.6: Barvy znázorňující idempotenci akcí: Tlačítko tvorby úkolu

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 3.5 a 3.6 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.

Kapitola 4

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 implementací strávil nezanedbatelné množství času. Tento popis je již totiž dostupný ve formě analýzy a návrhu, kterým odpovídá.

4.1 Podpůrné nástroje vývoje

4.1.1 Webpack

Webpack [73] 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 řídící se standardem ES6 či novějším. S pomocí Webpacku lze nastavit, aby se při zpracování některých souborů použil *loader*, který například převede SASS na CSS, konstrukce ES6 na ES5 apod. [33].

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*[9]. 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

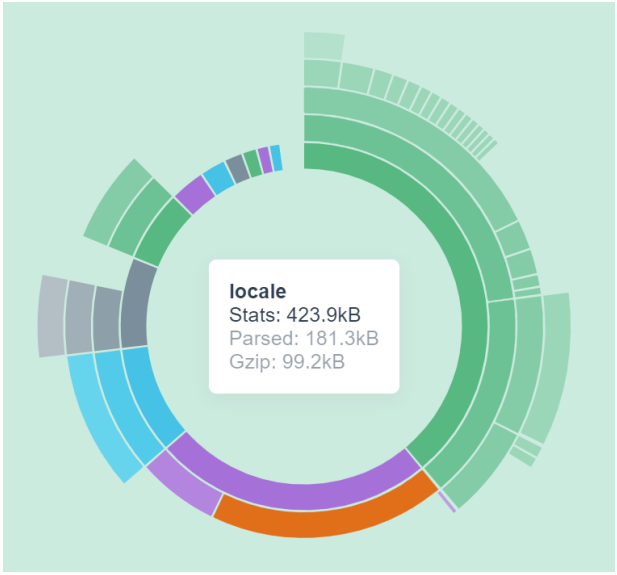
V předchozím paragrafu jsem popisoval výhodu použití Vue-CLI, ke kterému je ale vhodné říci i několik dalších slov, jelikož se jedná o velmi prakticky nástroj, usnadňující mnoho běžných požadavků na správu projektu.
Zaměřit bych se chtěl především na možnost spuštění grafického prostředí, pomocí příkazu `vue ui` z terminálu, které 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. Jedná se například o pluginy pro samotné `vue-cli`, `eslint` nebo třeba knihovnu `vuetify`, která poskytuje grafické

[9] Nejedná 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.



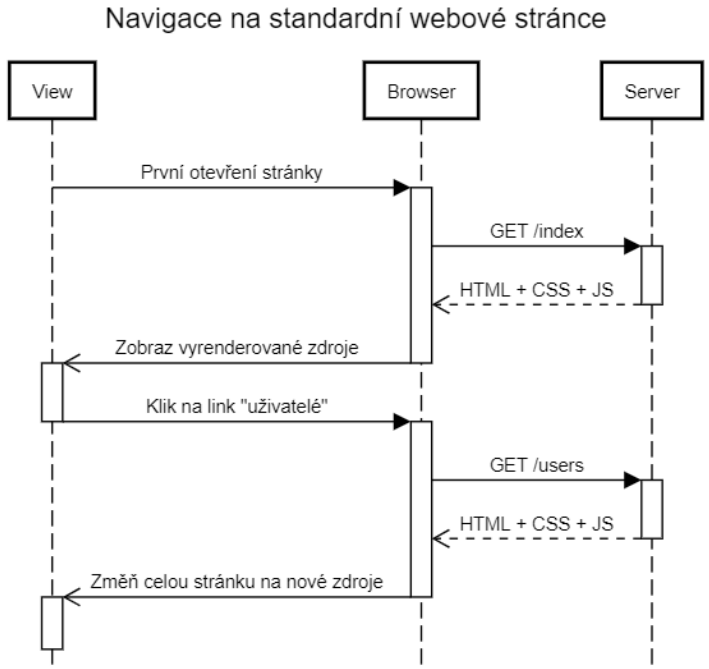
Obrázek 4.1: Analýza velikosti výsledné aplikace: překlady moment.js (na obrázku popsáno jako locale a označeno oranžově) zabírají velké procento z celkové velikosti souboru.

Optimalizace velikosti aplikace odebráním zbytečných překladů. Jednou ze závislostí, kterou v projektu používám, je knihovna *moment* [44], která se stará o parsování a zobrazování data a času, 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 velikostu výsledných souborů pomocí Vue-UI jsem odhalil, že tyto překlady

zabírají téměř *15% celkové 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 [37], 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*.

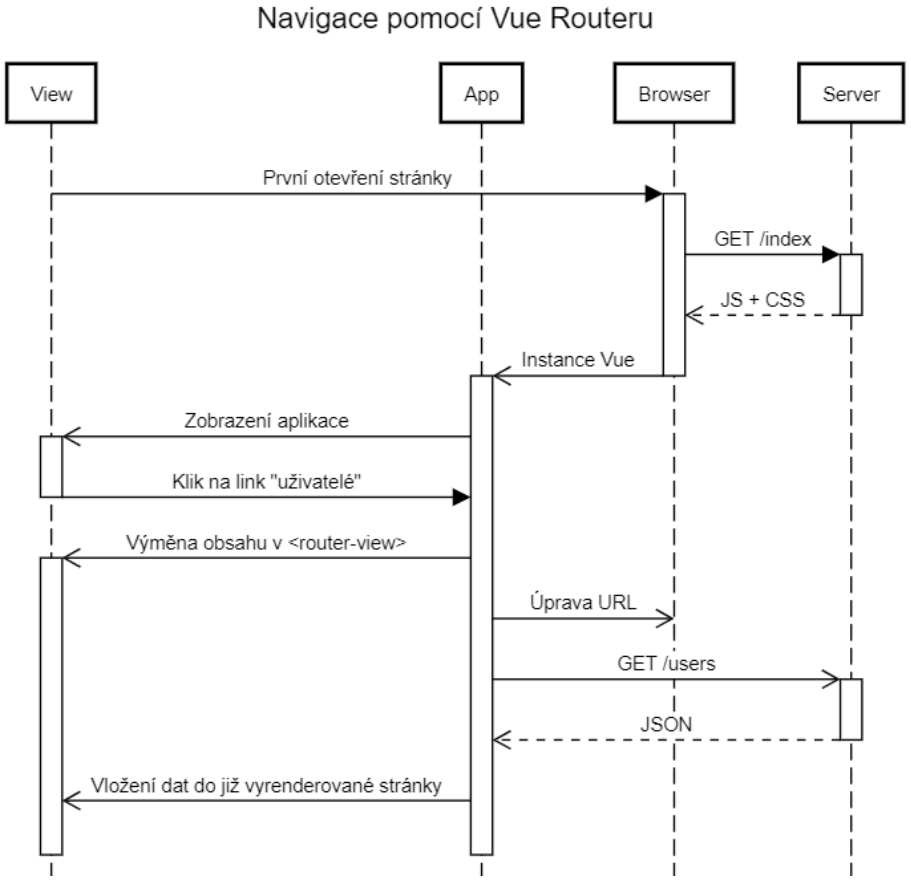
4.2 Router



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

Jednou z prvních komponent, kterou jsem k čistému Vue.js přidal, byl oficiální Vue Router [30]. Koncept routeru je dobře znám z jakékoliv jiné aplikace, která nějak pracuje s prohlížečem uživatele.
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.



Obrázek 4.3: Sekvenční diagram 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.

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 pomocí routeru vůbec nemění. K vyřešení tohoto problému sice

```
{
    path: '/',
    component: Homepage,
    meta: {
        title: 'Swordfish'
    }
}
```

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

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

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

ve Vue Routeru neexistuje nativní podpora, ale řešení je opravdu snadné - je popsáno v jednom z *issues* v repozitáři na GitHubu [76] 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 (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.

```
function buildPath(pathWithPlaceholders, parameters) {
    let routePath = pathWithPlaceholders.replace(/\([^/]+\)/g, '');
    const matches = routePath.match(/(:[a-zA-Z]+\/)|(:[a-zA-Z]+$)/g);
    if (matches !== null) {
        for (const match of matches) {
            const matchParam = match.replace(/\//, '').replace(/:/, '');
            routePath = routePath
                        .replace(match, parameters[matchParam] + '/');
        }
    }
    return routePath;
}
```

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

4.3 Moderní 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.

```
{
  "name": "Swordfish",
  "icons": [
    {
      "src": "favicon/swordfish-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "favicon/swordfish-512x512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ],
  "start_url": "/",
  "display": "standalone",
  "orientation": "portrait",
  "background_color": "#009688"
}
```

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 [60], jehož specifikace je v době psaní tohoto textu stále ve stavu návrhu, přestože vzniká již od května roku 2014 [61].
I přes nedokončenou speficikaci 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`[10] [38], jejíž použití je obdobné, jako známe z jiných jazyků. Zatímco například v PHP se často překlady zapisují do `.yaml` nebo `.neon`, 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 standarního 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.

```
1  const user = "uživatele";
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.

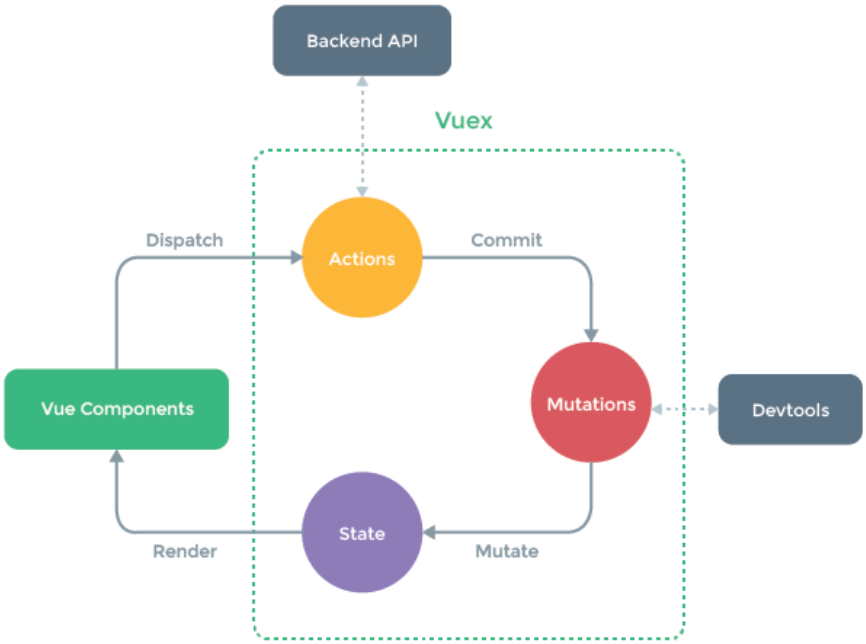
Překlady a uživatelská nápověda. Jelikož výsledná aplikace obsahuje i stručnou nápovědu, která má zatím pouze textový obsah, využil jsem pro zápis jejího obsahu také formát, který používá `i18n` pro překlady. Tyto texty jsem ale neintegroval do seznamu obecných překladů v aplikaci, nýbrž s nimi pracuji odděleně - pomocí vlastního řešení pro zobrazování nápovědy. Tím mohu všechny

[10] název je zkratka slova *internationalization* - číslovka 18 značí počet přeskočených znaků

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

Jako první se zde sluší říci, co to vlastně *State management pattern* je. Ve Vue.js aplikaci máme typicky velké množství komponent, které mezi sebou často potřebují sdílet nějaký *stav*. Nabízí se tedy vytvoření centrálního bodu, který bude stav udržovat a umožní jej měnit. Na změny by v ideálním případě měly dotčené komponenty reagovat automaticky, a tyto změny by také měly být nějak řízeny, aby nebylo možné nastavit nevalidní hodnoty. Přesně toto nabízí knihovna *Vuex* [74], jejíž interní cyklus popisuje diagram 4.4.



Obrázek 4.4: Diagram použití Vuex

Nebudu zde rozepisovat detaily jejího použití, ty jsou dostupné kdykoliv online v její kvalitní dokumentaci. Popíšu ale své vlastní zkušenosti s touto knihovnou

a jak jsem implementoval některá vylepšení. Začněme však od kraje - jako první jsem Vuex použil pro zobrazování snack message - tedy krátkých informačních zpráv, které se zobrazují uživateli ve spodní části obrazovky, především jako potvrzení jeho akce. K tomu jsem použil Vuex, do jehož stavu jsem ukládal obsah zprávy a případně její trvání. Zaslání takové zprávy znázorňuje ukázka kódu 4.6.

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

Ukázka kódu 4.6: Vuex pro snackbar-message: použití z jiné komponenty

Výše uvedený kód se mi ovšem v průběhu psaní aplikace přestal líbit, neboť zobrazování těchto zpráv je velmi častá operace a její vyvolání není v ukázce příliš jednoduché. Proto jsem si nastudoval mixiny ve Vue.js [42] a vytvořil mixin[11], který zjednodušuje zasílání zpráv do Snackbaru. Použití v komponentě vybavené takovým mixinem je znázorněné na ukázce kódu 4.7.

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

Ukázka kódu 4.7: Použití mixinu pro zjednodušení zasílání zpráv do Snackbaru

Persistentní úložiště. Když už máme efektivně vyřešené, jak data ukládáme, a aby se neměnila *jen tak*, ale vždy řízeně, může být v některých případech škoda, že v javascriptovém frameworku stačí jedno obnovení stránky pomocí `F5`, `ctrl-r` či chcete-li tlačítkem v prohlížeči, a všechna tato data se ztratí, neboť bylo vše uloženo pouze v paměti stránky, která se touto akcí celá přepíše. I v aplikaci pro správu skladu jsem se dostal k tomu, že některé věci bych potřeboval uložit tak, aby se nepřemazaly. Projdu nyní jednotlivé moduly, které ve Vuex v době psaní tohoto textu používám, a okomentuji jejich možnosti persistence:

- *Stav připojení k API:* modul udržující informaci, zda je dostupné API backendové části aplikace.

[11] Mixin je možné použít pro definici dat a metod, kterými má komponenta disponovat, a které se do ní vloží při jejím vytvoření. Výhodou je, že mixin je znovupoužitelný a může být vložen do libovolného počtu komponent.

  - Ukládání tohoto modulu persistentně nemá příliš smysl, protože drží vždy pouze aktuální informaci.

- *Loader:* tedy ukazatel, že se něco stahuje. V aplikaci rozlišuji dva typy loaderů: jeden nenápadný ve spodní části obrazovky, a druhý znemožňující provádět jakékoliv akce, dokud se obsah nenačte. U prvního zmiňovaného je navyšován počet požadavků, které běží, a teprve při doběhnutí všech se loader skryje.

  - Vzhledem k možnému vyššímu počtu běžících požadavků se nabízí, že by tato data mohla být persistentní, ale ve výsledku je to také nevhodné, protože při obnovení stránky se všechny běžící požadavky zahodí, a jejich potenciální výsledek je ignorován.

- *Cache:* místní úložiště dat z API

  - Tato část úložiště je detailněji popsána v sekci 4.7 níže.

- *Autorizace:* Jelikož se uživatelé do aplikace přihlašují, a backendové API je samozřejmě zabezpečené proti neoprávněnému přístupu, je potřeba na klientském zařízení udržovat informace o přihlášení. Aplikace implementuje protokol OAuth2 [23], který na klientském zařízení pracuje s dočasnými identifikátory `access_token` a `refresh_token` plus některými dalšími položkami, tyto dvě jsou ale nejkritičtější. Ukládání těchto tokenů by se teoreticky mohlo hodit, aby se uživatel nemusel při znovunačtení stránky znovu přihlašovat.

  - Jelikož znalost tokenu prakticky opravňuje kohokoliv vystupovat pod jménem uživatele, který token při přihlášení získal, nesmí být ani jeden z nich ukládán kamkoliv jinam než do aplikace samotné. Z toho důvodu není doporučováno ukládat tokeny persistentně, existuje však zde jedna výjimka. Během vývojového procesu, pokud se pracuje s API, které je zabezpečené, používáme tyto tokeny také. V takovém případě je možné tokeny ukládat persistentně, protože s nimi pracuje pouze vývojář, který používá testovací uživatele a testovací data - nehrozí tedy žádný únik či poškození dat reálných uživatelů.

- *Snackbar:* modul pro zobrazování krátkých zpráv uživateli.

  - Z logiky věci zde vyplývá, že nemá smysl po přenačtení stránky zobrazovat stále stejnou jednorázovou zprávu, modul tedy není nastaven jako persistentní.

- *Záznam času stráveného plněním úkolu:* tento modul ukládá, kdy bylo spuštěno měření času stráveného plněním úkolu a další spjaté informace.

  - Tuto informaci je naopak velmi důležité ukládat persistentně, neboť při přenačtení stránky je důležité, aby čas nebyl resetován.

- *Uživatelská konfigurace aplikace:* zde jsou uloženy informace jako například jazyk, tmavý režim, povolení záznamu času apod.

  - Tyto informace bych rád ukládal na API, avšak to zatím takové položky nepodporuje. Proto je také ukládám lokálně a modul je tedy persistentní.

Obsah těch modulů, které jsem vyhodnotil jako vhodné k persistenci, ukládám pomocí vlastního řešení do *LocalStorage* webového prohlížeče. V definici modulu pro to stačí nastavit příznak `persistent`, případně `devPersistent` pro ukládání pouze ve vývojovém režimu, a o zbytek se postará můj globální handler nastavený pro všechny akce Vuex - který při zapnutí aplikace existující stav načte a jakoukoliv změnu zapíše právě i do *LocalStorage*.

4.6 Hlídání konektivity

V moderních aplikacích, které veškerá data ukládají na API, je vhodné hlídat dostupnost tohoto API.
Prvním krokem k realizaci této funkčnosti bylo zjišťovat aktuální stav připojení - tedy zda je aplikace online, nebo offline. Jako první jsem našel vlastnost prohlížeče `window.navigator.onLine` [46], která by přesně o tomto měla informovat a navíc poskytuje i možnosti poslouchat její změny pomocí běžných JS eventů. Po hlubším prozkoumání jsem ale zjistil, že tento stav odpovídá pouze dostupnosti *místní sítě*, tj. například dostupnost nejbližšího routeru. Pokud bude fungovat spojení mezi prohlížečem a routerem, ale router samotný nebude mít přístup k internetu, bude tento stav mít hodnotu `true`, což ale už pro mé potřeby není vhodné - potřebuji znát dostupnost backendového API.

Po tomto zjištění jsem tedy připravil vlastní řešení, které periodicky zasílá dotazy na výchozí endpoint API, ve kterém kontroluje pouze návratový kód. Kontrola probíhá v minutových intervalech a při odpojení od sítě častěji. Při zaimplementování tohoto řešení mě napadlo, zda nebude problém s tím, že by funkce využívala zbytečně moc dat, provedl jsem proto zběžné výpočty: Velikost jednoho požadavku pro zjištění konektivity je 280 B. Při jednom požadavku za minutu to dělá 16.8 kB za hodinu, což je za osmihodinovou směnu 134,4 kB - tedy zcela zanedbatelná velikost.

4.7 Cache

Při psaní aplikace jsem častokrát narazil na problém, že mnohokrát za sebou potřebuji načíst stejná data. Nejprve bylo možné tento problém řešit efektivně, data jsem načetl při vytvoření komponenty, a její součásti je poté načítaly jednotně. Problém ale přišel ve chvíli, kdy jsem chtěl vytvořit obecnou komponentu, která bude s takovými daty pracovat, a nechtěl jsem, aby byla závislá na externích datech - žádoucí bylo, aby si je načetla sama. Příkladem je například komponenta zobrazující informace o umístění, která jako parametr přijímá jeho ID. Cílem komponenty je načíst si o umístění informace, jako jméno nebo kód, a ty vypsat uživateli. Někdy se ale stává, že je na stránce potřeba vypsat i více stejných umístění, a tak je na ní několik komponent se stejným ID umístění. Každý si pak samostatně načítá zcela stejná data z API, což je nejen neefektivní z hlediska využití sítě, ale také pomalé.

Vue keep-alive. Jednou z prvních objevených možností je použití wrapperu `<keep-alive>`[17], která není tak zcela o cachování dat z API, ale o cachování celých Vue komponent, což při správném použití může vést ke stejnému výslednému efektu. Bohužel při hromadném aplikování na celý projekt začalo docházet k nežádaným vedlejším efektům - například odeslaný formulář byl při znovuotevření vyplněn starými daty, nebo na jiných stránkách byl špatně načten nový stav z API a zobrazoval se ten starý. Nakonec jsem `keep-alive` přestal používat, ale nechávám ho ještě jako otevřený v tipech k dalšímu vývoji aplikace.

Vlastní cache. V současnosti používané řešení spočívá ve vlastím aktivním cachování vybraných položek, které se ukládají do persistentního Vuex store popsaného v sekci 4.5. Toto řešení jsem testoval ke konci vývoje aplikace a projevilo se jako poměrně flexibilní, bez vedlejších efektů, avšak za cenu vlastní správy cachovaných položek, včetně jejich ručního zápisu apod. Náročnost na psaní kódu je tedy poměrně vysoká, což by mohlo do budoucna přinášet spíše problémy, a proto budu se této problematice ještě věnovat.

Axios cache. Další alternativou by bylo využít cache, která by byla napojená přímo na HTTP klienta - Axios. Tuto možnost ještě musíme řádně prozkoumat a vyhodnotit, zda by byla pro projekt přínosná.

4.8 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 složitě se zde generují formuláře, jsem ale zjistil, že pro svou práci chci rozhodně vymyslet lepší systém. Níže přikládám 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()` [48] 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 je ale ještě nutné 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 ukázku kódu (4.8), která znázorňuje, jak může vypadat *definice formuláře* pro 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

```
const stockLocationForm = {
    name: '',
    code: null
};

const stockLocationFormRender = {
    name: {
        icon: 'label',
        max: 50,
        required: true
    },
    code: {
        icon: 'line_weight',
        max: 40,
        hint: 'stocks.locations.codehint'
    }
};

export {stockLocationForm, stockLocationFormRender};
```

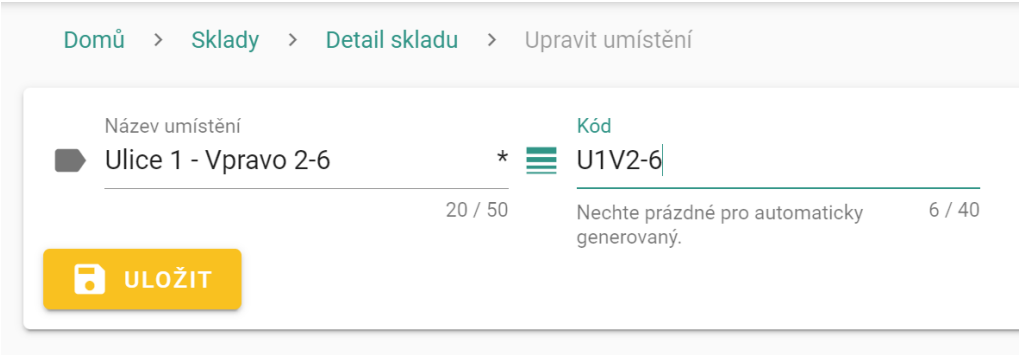
Ukázka kódu 4.8: Příklad definice formuláře: jednoduché skladové umístění

API a při načtení se naopak celý přepíše daty z API. Definice zobrazení pak obsahuje informace, jak má samotný formulářový prvek vypadat a jak se má chovat. 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 mohu 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`, který ukládá čas strávený v úkolu, který systém zaznamenává zcela automaticky (viz sekce 4.12).
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 prvku typu `autocomplete`.

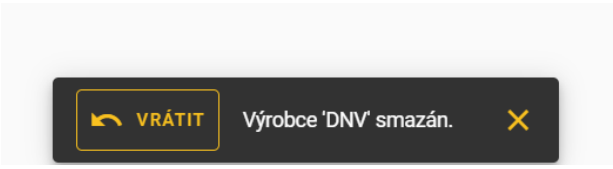
Jedná se tedy o poměrně flexibilní systém, který umí nejen zpracovávat různé typy vstupů, ale také je poměrně bohatě upravovat a přizpůsobovat - pro mé potřeby vykreslování formulářů na tvorbu či úpravu entit, které se v systému nacházejí: tj. skladů, dodavatelů, produktových karet, odběratelů, umístění apod., a dále formulářů zadávání a schvalování úkolů, které na skladě probíhají, je tato komponenta a její konfigurace naprosto dostačující a do budoucna hlavně velmi snadně rozšiřitelná a konfigurovatelná. Ukázka formuláře vytvořeného přes tuto jednotnou komponentu je vložena přímo v textu jako obrázek 4.5.



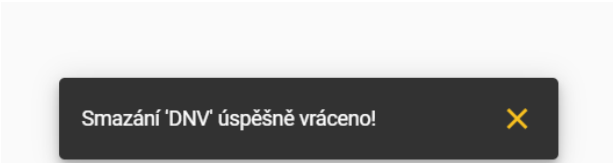
Obrázek 4.5: Formulář vykreslený pomocí jednotné formulářové komponenty

4.9 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 4.6). Takovýto snack 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 4.7, či případně chybová hláška.



Obrázek 4.6: Undo: Možnost vrácení provedených změn



Obrázek 4.7: Undo: Potvrzení vrácení provedených změn

4.10 Realizace zkratek v systému pomocí čtečky čárových kódů

Tato funkce, navrhovaná v sekci 3.1.2, má ve výsledné aplikaci připravenou obecnou podporu, což znamená, že je možné načítat čárové kódy na jakékoliv obrazovce bez nutnosti klikat do formulářového prvku. Vstup z čtečky se zpracovává ve funkci, kterou si může libovolná komponenta aplikace přetížit. Na druhé straně ale nebyla z časových důvodů realizována samotná možnost nastavit si, jakou akci má který kód spouštět, aplikace tedy prozatím reaguje pouze na standardní napípnutí umístění či EANu skladové položky, do budoucna ale není problém realizovat i kompletní funkčnost, jak byla navržena.

4.11 Generování kódu ze specifikace OpenDoc

Ještě předtím, než jsem začal pracovat na realizaci samotných funkčnostech 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, zanášejí do něj 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 zanese chyba.
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 vypsat si textový zápis definice funkce, a z jeho hlavičky argumenty vyparsovat - viz ukázka kódu 4.9, která vychází z kódu Christophera Hermanna [24].

```
function getFnArgNames(fn) {
    return fn.toString()
        .match(/[^(]*\(([^)]*)\)/)[1]
        .split(',')
        .map(element => element.split('=')[0].trim())
        .filter(element => !(element === ''
            || element === null
            || typeof element === 'undefined')
        );
}
```

Ukázka kódu 4.9: Parsování názvů argumentů funkce

Jak se ale ukázalo později, toto parsování není použitelné v režimu, kdy je aplikace zkompilovaná pro produkční použití, neboť takový kód je minifikovaný, 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 začal točit v kruhu, protože 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. Dokumentace API, kterou mám dostupnou, je zpracovaná ve standardu OpenAPI 3.0 [50], je tedy možné ji zpracovávat automatizovaně různými nástroji, které stručně popíšu:

- *swagger-codegen* [63] Jedná se o velmi robustní nástroj, který dokáže generovat nejen modely, ale i 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 je pro mé potřeby nevhodná.
- *swagger-js* [35] Doplňující 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* [62] 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, které připadají v úvahu, jeden z nich ale interně používá ten druhý, takže jejich výstupy jsou shodné.
Zkoušel 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 nakonec rozhodl nepoužívat 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 4.8 - jedná se o například o konstantu `stockLocationForm` z ukázky kódu 4.8. Někdo by mohl namítat, že tento model nemá řádně definované typy a další náležitosti, a měl by naprostou 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).
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 napípávání zboží, nikoliv z vyplněného formuláře. V těchto několika případech jsem ručně vypsal 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.

4.12 Záznam času stráveného plněním úkolu

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 platofrmy. 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 já mám raději, když 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 na 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 kontrolu, 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 aplikace, a také ještě bude před ostrým používáním otestováno uživateli.

4.13 Perličky z vývoje

4.13.1 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* [55], 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 zaškrtávací 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 na rozdíl od oficiální dokumentace Vuetify [58] (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 čtyřicátém přeinstalování všech závislostí, jsem se dostal k importu *Material Design Icons* [26]. 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 šel tedy opravit diffem z ukázky kódu 4.10.

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

Ukázka kódu 4.10: Diff nastavení fontu ikonek ve Vuetify

Kapitola 5

Testování

5.1 První testování - během vývoje s odbornou osobou

První uživatelské testování výsledné aplikace proběhlo při jejím vývoji - 19. 9. 2019. V době testu byly připravené základní možnosti vkládání a úprav většiny entit v systému (skladové položky, dodavatelé atp.) a necelé čtyři ze stěžejních úloh - příjem dodávky, naskladnění, inventura a část přesunu zboží.
Aplikaci testovala osoba, která má zkušenosti 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 zavést nějakou detekci prohlížečů a případně uživatele informovat o nekompatibilitě aplikace se zvoleným browserem, ale po této skutečnosti jsem

ještě zvýšil této úpravě prioritu.
Výstupem z tohoto testování vývojové verze aplikace je seznam postřehů, chyb a návrhů na zlepšení, které jsou k nalezení v příloze F.
Zde je vhodné poznamenat, že zhruba pětinu všech požadavků a chyb jsem již nějakým způsobem evidoval i před testováním - buďto formou *„TODO“ komentářů* přímo v kódu aplikace, nebo kartami v Trellu, avšak pro kompletnost zápisu jsem je ponechal i tam.

5.2 Druhé testování - alfa verze kompletní aplikace

Dne 6. 12. 2019 proběhlo druhé uživatelské rozhraní, které bylo na rozdíl od prvního testování mnohem obsáhlejší. Vydal jsem se do dvou skladů menších rozměrů, ve kterých jsem otestoval celkem čtyři osoby zde pracující.
Průběh testování probíhal s každou testovanou osobou stejně, podle následujícího rozpisu:

1. Pokud byl ve skladu používán již nějaký skladový systém, pozoroval jsem chvíli, jak se s ním pracuje a jaké jsou nejdůležitější úkony.
2. Položil jsem testované osobě několik otázek dle *Dotazníku před testováním* (viz příloha G).
3. Provedli jsme samotné testování aplikace, dle připravených úkolů (viz příloha H).
4. Položil jsem testované osobě několik dalších otázek, dle *Dotazníku po testování* (viz příloha) G).
5. Jako poděkování jsem nabídl vánoční perníčky.

V tomto textu budu vypisovat pouze nejdůležitější odpovědi a poznatky z testování, zápis požadavků a problémů zpracovaný do strukturovaného seznamu je dostupný v příloze I. Také nebudu zmiňovat celá jména testovaných osob, neboť to pro výstup testu není podstatné.
Během samotného testování aplikace jsme používali reální zařízení Zebra TC-25, na kterém jsem měl předinstalovanou aplikaci s podporou čtečky čárových kódů. Také jsem měl vytištěno několik čárových kódů, kterými se identifikují

umístění ve skladu a dále několik kódů reprezentující skladové položky. Kromě Zebry - na které jsem testoval především roli *skladníka*, jsme vždy využili i notebook, který je běžně ve skladu používán - na tom se naopak testovala role *vedoucího skladu*.

5.2.1 Testování v prvním skladu

Nejprve jsem začal ve skladu e-shopu, jehož sortimentem je převážně obuv, přípravky pro péči o tělo a zdraví, výbava domácnosti a také elektronika. Sklad se nachází v samostatné místnosti v průmyslové budově a obsahuje odhadem kolem čtyř set umístění, přičemž „umístění“ je krabice polepená štítkem, kterých je na sobě mezi čtyřmi až šesti, a jsou vyskládány v řadách.
V tomto skladu je aktuálně používán skladový systém Sysel, ze kterého vycházela i analýza požadavků pro nově tvořený systém, takže se nabízela možnost přímého srovnání obou systémů.
Ještě před započetím testování jsem si zavedl do nového systému dvě reálná umístění tohoto skladu a jeden konkrétní výrobek. Bohužel kvůli povaze skladovaného zboží jsem nenašel více stejných skladových položek, a tak jsme nemohli efektivně simulovat práci s více kusy stejné položky.

5.2.1.1 Postřehy z testování s první osobou

Žena, 42 let, dříve zdravotní sestra, nyní majitelka eshopu.

Dotazník před testováním. Z dotazníku provedeného před samotným testováním vyplynulo, že první testovaná osoba od skladového systému nejvíce očekává informace o umístění požadovaného kusu zboží během vyskladňování. Dále jí vyhovují režimy pípání při načítání kódů - krátké pípnutí jako potvrzení a dlouhé jako chyba. Také považuje za důležité mít v systému funkci průchodu historie umístění položky - pro případ, že na deklarovaném umístění se položka ve skutečnosti nenachází. Naopak jako problematické označila například nutnost posouvat se na obrazovce až na konec v případě, že něco naskladňuje, protože nejnovější položka se přidá na konec seznamu. Také jí vadí absence evidence kapacity umístění, kterou si vede v separátním souboru uloženém na notebooku, který je ve skladu.

Testování nové aplikace. Co se týká testování nového systému, přistupovala k němu spíše s rozvahou, často tápala, kudy má pokračovat, nebo co má zadávat. Toto bylo částečně způsobeno tím, že na rozdíl od Sysla je v novém systému prohozeno pořadí napípávání kusu zboží a umístění, ze kterého je sebráno, či na které je uloženo. Největší problémy dělalo hledání přidávání nových věcí - ať už skladových položek nebo úkolů, které je řešeno pluskem ukotveném v pravém dolním rohu obrazovky. Během testování mi bohužel příliš nesdělovala své myšlenky, které by byly přínosné pro případné úpravy, největších problémů jsem si ale i přesto všiml a poznamenal jsem si je.
Některé úkony prováděla poprvé, neboť je nedělá ani v současném skladovém systému - jednalo se například o založení nové skladové položky. Po nalezení příslušné akce bez problémů vyplnila potřebná pole, bohužel ale nedokázala určit, která pole jsou povinná[12].

Obrázek 5.1: Označení povinných polí formuláře: Název je povinný a uživatel v něm již měl kurzor, nebo se pokusil formulář odeslat. Model je povinný, ale uživatel do něj ještě neklikl. Výrobce je výběr ze seznamu, ale povinný není.

Častokrát jsme také narazili na to, že některé úkony jsou v novém systému složitější, než ve starém - již například zmiňovaná nutnost nejprve napípnout umístění. Starý systém nabízel automatické doplnění umístění, pokud byl napípnut výrobek, který je v systému pouze na jednom místě. Tato funkce ale někdy nebyla stoprocentně spolehlivá, a tak jsem se rozhodl mechanismus předělat tak, aby se vždy nejprve volilo umístění, od čehož jsem si sliboval snížení chybovosti především ve větších skladech. Rozumím ale argumentu, že pro sklad, který víceméně celý spravuje jeden člověk, je toto zbytečné ztížení, avšak již v době psaní této práce je vypsáno nové téma, které bude na tuto práci navazovat a jehož cílem bude právě optimalizace procesů na malých skladech - považuji tedy za korektní svůj současný přístup, a to, že v systému lze provést vše, kvalitně, avšak někdy zbytečně zdlouhavě, a efektivnější procesy se mohou dále optimalizovat.

[12]Povinnost polí je ve formulářích označována hvězdičkou a při chybě jsou pole označena červeně, viz obrázek 5.1. Toto označení považuji i po tomto testu za dostatečné.

Dotazník po testování. Po skončení testování jsem se vyptal na několik věcí přímo se týkajících nového systému. Jako největší slabinu označila především menší písmo a pro ni nižší přehlednost, na druhé straně uvítala zobrazování fotografií skladové položky. Nejvíce jí chyběla možnost zadat přímo počet kusů položky, napípávání všech jednotlivých kusů postupně jí nevyhovovalo. Na otázku, zda by jí dávalo smysl namísto pípání, nebo jako doplněk k němu, používat vibrace zařízení, reagovala, že jí to připadá jako zbytečnost, protože zvukový signál prý vnímá lépe.

5.2.1.2 Postřehy z testování s druhou osobou

Muž, 40 let, dle svých slov „majitel největšího e-shopu v ČR s mizerným skladovým systémem“[13].

Dotazník před testováním. Již z dotazníku provedeného před testování bylo zřejmé, že se jedná o člověka, který se se skladovými systémy již setkal, a to i ve větším měřítku než pouze ve skladu eshopu. Co se konkrétně aktuálně používaného systému - tedy Sysla - týká, vyhovuje mu především propojení s eshopem, rychlost a individuální řešení. Na druhé straně si ale stěžuje na některé kostrbaté funkce, jako příklad uvedl, že když přiřadí EAN k nové položce, ale ten stejný EAN už byl dříve použit, tak se ze staré položky odebere, aniž by o tom byl uživatel informován. Od skladového systému očekává především efektivní využití zaměstnanců, jedná-li se o velký sklad, a také snížení chybovosti. Za nejpalčivější procesy současného řešení označil expedici a inventuru.

Testování nové aplikace. V systému se pohyboval velmi rychle a většina zádrhelů byla způsobena spíše tím, že si chtěl něco upřesnit, nebo mi k tomu dát zpětnou vazbu. Efektivně využíval našeptávací pole, ve kterých začal vždy psát a klávesou enter potvrzoval zvolené položky, na rozdíl od předchozí testované osoby, která více používala myš. S přidáváním nových věcí pomocí šipky či pluska v pravém dolním rohu problém neměl, pouze na domovské obrazovce, kde se tento prvek používá pro tvorbu nových úkolů, mu přijdou dva kliky zbytečné a radši by měl seznam dostupný přímo, aby úkoly tvořil jen jedním kliknutím. Při naskladňování byl lehce zmatený nutností pípat nejprve umístění

[13] První část tvrzení není pravda, druhou nemohu objektivně posoudit.

a teprve poté skladové položky, ale rychle se nový postup naučil. Při otázce, jak ze systému zjistí, kde požadovanou položku nalezne, poměrně překvapivě šel hledat do zadání, přestože tato informace se vztahuje ke konkrétní položce. Až když v zadání tuto informaci nenašel, zkusil rozkliknout skladovou položku, kde už požadovaná informace byla. Během testování zanadával pouze jedenkrát, a to ve chvíli, kdy zjistil, že při přesunu položek mezi umístěními musí nejprve všechny položky napípat při jejich zvednutí, a znovu při jejich položení na cílové umístění. Tento proces mi přišel od stolu poměrně logický, protože pouze tak je možné zaručit, že skladník položí opravdu všechny položky, ale uživatel to vidí jinak - tento konkrétní tester při tomto zjištění aplikaci pojmenoval „skladový systém buzerant“.

Dotazník po testování. Po testování jsem se jako v prvním případě zeptal na několik dopřesňujících otázek. Na novém systému oceňoval nový a moderní design, přišlo mu ale že většina důležitých informací je vždy orientována až moc do levé horní části stránky. Uvítal novou možnost nastavit jednomu účtu role skladníka i vedoucího současně, u práce se skladovými položkami mu ale chyběla možnost přímo nastavit jejich množství. Co se konfigurovatelnosti domovské obrazovky týká, navrhuje i možnost zobrazovat úkoly ne ve sloupcích, ale v řádcích. Také má zkušenosti se skladováním zboží více různých majitelů, naopak šarže mu blízké nejsou, ale rád by o nich zjistil více, protože jimi označené zboží plánuje do budoucna také skladovat.

5.2.2 Testování v druhém skladu

Po otestování prvních dvou osob jsem se přesunul do druhého skladu, který na rozdíl od toho prvního žádný skladový systém nepoužívá, přestože počet skladovaných produktů také není nejmenší. Osoby pracující v tomto skladu si ale veškerá umístění pamatují, a umístění by zde ani nebylo zcela jednoduché zřídit, neboť se jedná víceméně o chodbu a jednu místnost v přízemí rodinného domu, kde je na první pohled vše uloženo tam, kam se to zrovna vešlo. Jelikož se ale i přesto o zavedení skladového systému do budoucna uvažuje - pravděpodobně již v nových prostorech, kam by se měl sklad přemístit, bylo testování na místě.
Kvůli nižším teplotám v samotném skladu jsme testování prováděli v zázemí, u notebooku, a umístění jsme simulovali mými čárovými kódy, které jsem měl

předem vytištěné. Jako skladové položky jsme poté využili běžné zboží, které bylo dostupné v kuchyni, a od kterého jsme měli k dispozici více než jeden kus.

5.2.2.1 Postřehy z testování s třetí osobou

Žena, 30 let, živnostnice, skladník na expedici

Dotazník před testováním. Zkušenost se skladovými systémy zde byla nulová, čímž odpadla i většina dalších otázek, které jsem v dotazníku před testováním pokládal. Co se požadavků na takový systém týká, očekávala by především časové usnadnění práce ve smyslu zlepšení orientace, případně i poskytování vhodných statistik jako například trendů ve vyskladňování zboží - doporučování k objednání apod. Během této diskuze jsme se dostali až k bodům, které se již týkají spíše systému e-shopu, jako například nabízení slev na produkty, které se dlouhodobě neprodávají, ale přesto jsem si je poznamenal.

Testování nové aplikace. Během samotného testování projevila poměrně velkou komunikativnost, sdílela své pocity a myšlenkové pochody, což bylo pro účely uživatelského testování naprosto skvělé. Z jejího testování tedy mám nejdelší zápis a nejvíce návrhů na zlepšení, o kterých budu dále přemýšlet. Stejně jako se již objevilo dříve, měla poměrně problém s pluskem v pravém dolním rohu, avšak jakmile jej jednou nalezla, tak již věděla, co v něm očekávat a práce se rapidně zrychlila. Mezi nejzajímavější návrhy ke zlepšení patří možnost zasílat skladníkům přímé notifikace, že jim byl přiřazen nový úkol. Poměrně problematické pro ni bylo pochopit, že nejprve je vždy nutné naskenovat kód umístění a teprve poté kód skladové položky - i když již tento proces procházela poněkolikáté, tak skoro vždy provedla skenování v opačném pořadí, přestože systém poskytuje hlášky, jaké skenování je očekáváno. Z toho tedy vyplynulo, že se hlášky musí ještě zlepšit - prostor pro to je a hned při testování jsem si zapsal návrhy na konkrétní úpravy. Podobný problém se zobrazovanými informacemi na stránce nastal i při vyskladňování, ve kterém ji mátla informace o umístění, na které má vyskladňované zboží přemístit - místo toho si myslela, že na uvedeném umístění najde zboží k vyskladnění. I na tento bod jsem měl již při testování zapsaný návrh na zlepšení.
Diskuze proběhla také nad funkcí záznamu stráveného času, u kterého marně

hledala funkcionalitu pauzy, kterou jsem hodlal do aplikace tak jako tak přidat. Matoucí ale byl dialog při odchodu z rozpracované úlohy, který se ptá, zda chce uživatel uložit čas - v testerce vyvolal dojem, že když čas uloží nyní, tak už nikdy později nepůjde přidat čas další.

Dotazník po testování. // TODO

5.2.2.2 Postřehy z testování s čtvrtou osobou

Žena, 32 let, živnostnice, majitelka e-shopu.

Dotazník před testováním. Jelikož se jedná o stále stejný sklad, ani poslední testovaná nemá zkušenosti se skladovými systémy, a proto byla většina otázek ze sekce „před testováním“ přeskočena. Od skladového systému očekává, že jí řekne, kde se skladová položka nachází, další nápady jsem z ní ale v této části testování nedostal.

Testování nové aplikace. Při testování aplikace se projevila jako učenlivá, ale dosti skeptická k novým věcem, největší problém bylo vždy požadovanou věc v systému nalézt, ale po jejím prvním projití byl již druhý průchod velmi rychlý. Z tohoto testování také pochází některé citace, které nešlo do textu nezahrnout:

- *Během hledání, kudy se zakládá nová skladová karta:* „Tady nikde není nový produkt… Tohle? Sorry jako ale takovýhle plus? … Ježiš marja zase dole v pravym rohu!“
- *Během vyskladňování typu „expedice“ (což vlastně znamená přemístění na jiné místo, kde se bude balit):* „Já si to teda vezmu z U2[14] a přemístím to na U3 a to je uplně dementní, na tom strávíme hrozně moc času!“

Při dotazu, zda nějak naloží s informací, že daná skladová položka je někdy v krabici, která má vlastní kód, a uvnitř je deset kusů zboží, prohlásila, že namísto aby si vytvořila v systému nový kód této položky, který bude reprezentovat 10 kusů, tu krabici rozbalí a po jedné položce naskladní. Na rozdíl od předchozích testerů neměla žádné problémy s pochopením pořadí

[14] označení umístění

pípání umístění a položek, v tomto se zorientovala velmi rychle. U dialogu se záznamem stráveného času namítla poměrně věcný bod, že čas by se mohl ukládat vždy, protože takto by pro skladníka bylo jednodušší s časem manipulovat - načež jsem usoudil, že tuto funkcionalitu budu muset ještě znovu promyslet.

Dotazník po testování. Při pokládání otázek po testování jsem byl opět svědkem odmítavého přístupu k zavedení podobného systému, a to větou „Já bych to nepoužívala ani náhodou, přijde mi to hrozně zdlouhavý, že bych strávila víc času, než prostě jen vezmu, zabalim, nalepim štítek.“. Systém jako takový ale špatně nedopadl, veškeré potřebné informace zobrazoval, dokonce jí přišlo jako dobrý nápad umožnit i vibrace zařízení jako doplněk k pípání, a stejně tak jí dávalo smysl spouštění rychlých akcí za pomoci načtení kódu odkudkoliv z aplikace.

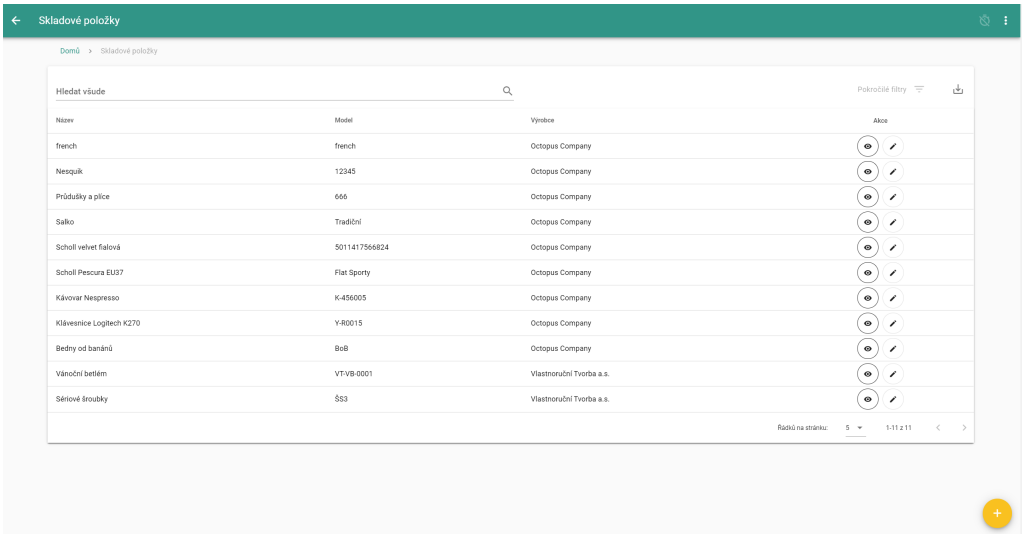
5.2.3 Shrnutí testování a největší problémy

Uživatelské testování jako takové považuji za velmi úspěšné. Aplikace byla použitelná, všechny potřebné věci v ní testeři byli schopni nakonec provést, byť v některých případech k tomu vedla trnitá cesta.
Při testování se vyskytlo pouze několik aplikačních chyb, které jsme byli schopni obejít a pokračovat v testu, žádná z chyb nebyla zcela kritická.

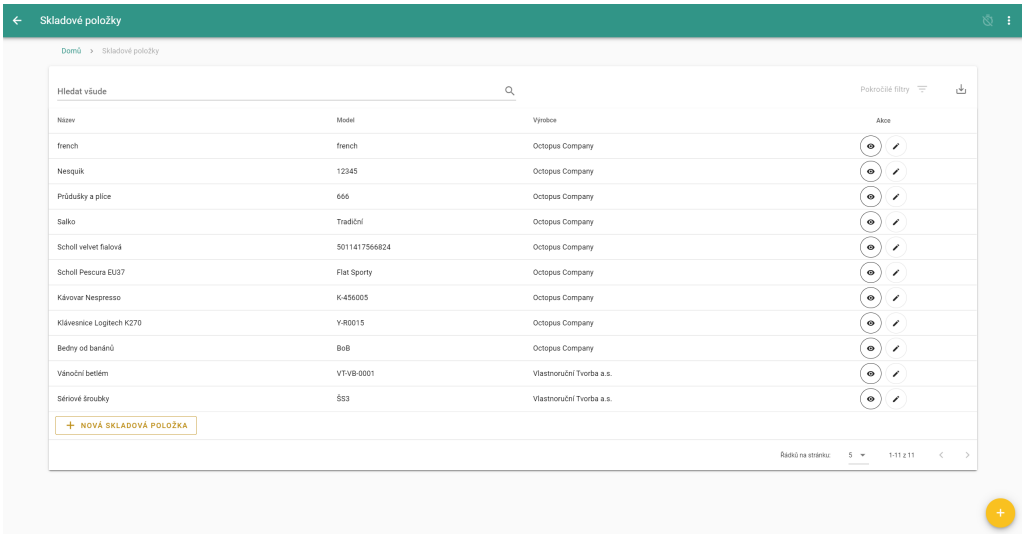
Jelikož nejpalčivější problémy uživatelského rozhraní se často opakovaly u více testerů, popíšu je nyní hromadně:

Tvorba nových věcí pomocí pluska v pravém dolním rohu. Na obrázku 5.2 je vidět detail seznamu skladových položek, odkud jsem po testerech chtěl založit novou skladovou položku. Většina z nich měla problém s nalezením žlutého „plus“ v pravém dolním rohu, přes které se nová položka přidává. Při příštím použití tohoto tlačítka již vše probíhalo svižně, ale i přesto jsem se rozhodl v reakci na tento problém přidat možnost vkládání položek i na konec tabulky, jak je vidět na obrázku 5.3. Tato volba se zobrazuje pouze na tabletech a větších zařízeních, na mobilních zařízeních je plusko dostatečně v zorném úhlu, a navíc

se na zařízeních s Androidem jedná o standardní komponentu, na kterou jsou uživatelé zvyklí.



Obrázek 5.2: Tvorba nové skladové položky: stav před uživatelským testováním



Obrázek 5.3: Tvorba nové skladové položky: úprava po testování

Přesun mezi umístěními a dvojité pípání položek. Dalším velmi palčivým problémem, na který si uživatelé stěžovali, byla nutnost znovu zvolit všechny položky, které se chystají přesunout na cílové umístění při přesunu položek, či při vyskladňování typu „expedice“. Při těchto úkolech je potřeba nejprve

napípat zboží, které skladník *sbírá*, a poté jej napípat znovu, při *pokládání* na cílové umístění. Při přesunech, které mají pevně dané cílové umístění může být ale druhé pípání velmi otravné, a tak jsem se rozhodl do aplikace přidat *volitelnou* možnost, která po označení cílového umístění umožní rychle na něj přesunout vše, co má skladník aktuálně „u sebe".

Ruční nastavení počtu kusů zboží. Funkce, kterou všichni testeři shodně žádali, je umožnění ruční manipulace s počtem načteného zboží, což umožní jednodušeji zadat větší množství položek, avšak může to také zvýšit chybovost. Přesto jsem se rozhodl nastavení přímé kusovosti uživatelům umožnit a v případě, že by to nějakému provozovateli skladu vadilo, může být tato komponenta zobrazována na základě nastavení konkrétní instance skladového systému.

Zavření detailu skladové položky. Detaily o skladové položce se během práce na úkolu zobrazují v komponentě nazývané „bottom sheet", což je vlastně modální okno, které vyjíždí ze spodní části obrazovky a zabírá celou její šířku. Uživatelské testování ukázalo, že uživatelé mají často problém tento pohled zavřít, což se provádí dotykem (či kliknutím) kamkoliv mimo. Po zvážení těchto nejasností jsem se rozhodl přidat do jeho pravého horního okraje i tlačítko na jeho zavření, viz obrázek 5.4.

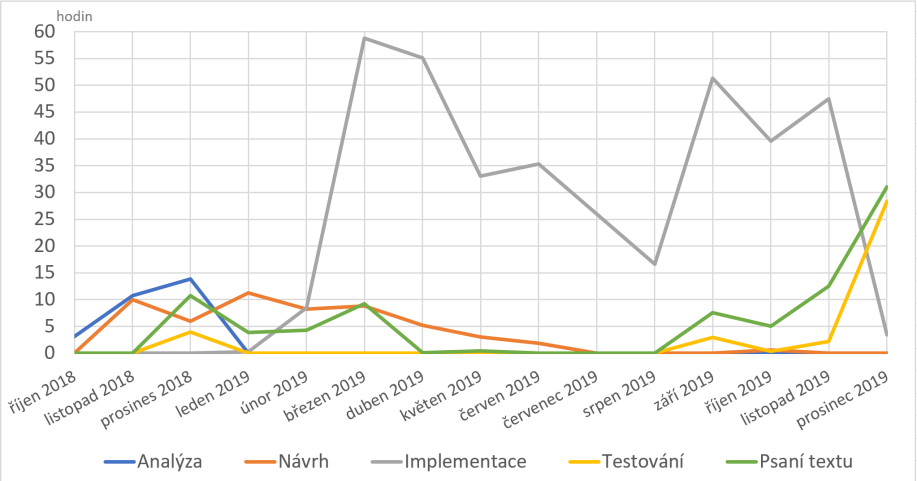
Obrázek 5.4: Zavření přehledu skladové položky: křížek v pravém horním rohu byl přidán po uživatelském testování.

Závěr

Závěrem lze říci, že vytvořená aplikace má před sebou ještě dlouhou cestu, avšak její základ je vytvořen na stabilním podkladu, ze kterého bude možné ji dále efektivně rozvíjet. Návrhy na další funkce, které buďto nebyly zcela dokončeny v rámci této práce, nebo které byly objeveny až při jejím uživatelském testování, jsou evidovány v systému Trello.

Během realizace jsem se naučil alespoň na určité úrovni používat framework Vue.js, který se mi během tohoto procesu velmi zalíbil, a jistě jej použiji i v jiných projektech. Kromě této technologie jsem také více pronikl do architektury moderních frontendových aplikací, u kterých může být výzvou řešení konektivity, rychlosti načítání obsahu nebo jejich kompatibilita se zastaralými prohlížeči.

Výsledná aplikace, tak jak je k dispozici při dokončování tohoto textu, je reálně použitelná - obsahuje nezbytná zabezpečení jako například OAuth2 přihlašování nebo podporu čtečky kódů. Přesto ještě před jejím reálným nasazením doporučuji vylepšit některé klíčové funkce, u kterých z uživatelského testování vzešly podněty ke zlepšení. V průběhu celé práce jsem si zaznamenával čas strávený realizací tohoto projektu, který nyní jako měsíční souhrny přikládám v grafu 5.5.



Obrázek 5.5: Čas strávený realizací projektu, rozdělený dle typu aktivity.

Při zpětném ohlédnutí na tento graf vidím poměrně velkou převahu implementační části, která je způsobena tím, že analýzu jsme prováděli ještě ve dvou - společně s Pavlem Kovářem, a její efektivita byla poměrně vysoká. Naopak zejména počátky implementace, které spočívaly převážně v učení se pro mě zcela nové technologie, přinesly pochopitelnou neefektivitu samotného vývoje a tudíž nárůst stráveného času.
Testování eviduje rapidní nárůst až v závěru, což je způsobenou velkou časovou náročností uživatelského testování ve skladech. Přesto bylo testování realizováno i v průběhu vývoje, avšak oproti závěru to bylo ve velmi malé míře.

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. *Dynamic & Async Components* [online]. 2019 (cit. 2019-12-20). Dostupné z: https://vuejs.org/v2/guide/components-dynamic-async.html#keep-alive-with-Dynamic-Components.

18. EHRENBERG, Daniel. *Class field declarations for JavaScript* [online]. 2019 (cit. 2019-03-02). Dostupné z: https://github.com/tc39/proposal-class-fields#private-fields.

19. *Ember.js - A framework for ambitious web developers* [online]. 2018 (cit. 2018-12-17). Dostupné z: https://www.emberjs.com/.

20. *Ember.js Guides - Guides and Tutorials - Ember Guides* [online]. 2018 (cit. 2018-12-28). Dostupné z: https://guides.emberjs.com/release/.

21. *Getting Started - React* [online]. 2018 (cit. 2018-12-28). Dostupné z: https://reactjs.org/docs/getting-started.html.

22. 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.

23. HARDT, Dick. *RFC 6749 - The OAuth 2.0 Authorization Framework* [online]. 2012 (cit. 2019-11-28). ISSN 2070-1721. Dostupné z: https://tools.ietf.org/html/rfc6749.

24. HERMANN, Christoph. *retrieve-arguments/index.js* [online]. 2014 (cit. 2019-10-17). Dostupné z: https://github.com/stoeffel/retrieve-arguments/blob/master/index.js.

25. 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.

26. *Icons - Material Design* [online]. 2019 (cit. 2019-09-24). Dostupné z: https://material.io/resources/icons/.

27. *Introduction - Enzyme* [online]. 2019 (cit. 2019-01-22). Dostupné z: https://airbnb.io/enzyme/.

28. *Introduction - Testing - Ember Guides* [online]. 2019 (cit. 2019-01-22). Dostupné z: https://guides.emberjs.com/release/testing/.

29. *Introduction - Vue.js* [online]. 2018 (cit. 2018-12-28). Dostupné z: https://vuejs.org/v2/guide/.

30. *Introduction | Vue Router* [online]. 2018 (cit. 2019-02-09). Dostupné z: https://router.vuejs.org/.

31. *Introduction | Vue Test Utils* [online]. 2019 (cit. 2019-01-22). Dostupné z: https://vue-test-utils.vuejs.org/.

32. *Jagu - Software na míru, webové prezentace, grafika, webhosting* [online]. 2018 (cit. 2018-12-21). Dostupné z: https://www.jagu.cz/.

33. 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/.

34. *JavaScript Error Tracking* [online]. 2019 (cit. 2019-01-21). Dostupné z: https://sentry.io/for/javascript/.

35. *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.

36. *Jest - Delightful JavaScript Testing* [online]. 2019 (cit. 2019-01-22). Dostupné z: https://jestjs.io/.

37. 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.

38. KAWAGUCHI, Kazuya. *Vue i18n* [online]. 2019 (cit. 2019-03-04). Dostupné z: https://kazupon.github.io/vue-i18n/.

39. KOVÁŘ, Bc. Pavel. *Backend skladového systému*. Diplomová práce. Czech Technical University in Prague, Faculty of Information Technology, 2019.

40. KUROSKI, Daniel. *Working an application in Vue.js with TDD* [online]. 2018 (cit. 2019-01-22). Dostupné z: https://vue-test-utils.vuejs.org/.

41. 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.

42. *Mixins - Vue.js* [online]. 2019 (cit. 2019-10-03). Dostupné z: https://vuejs.org/v2/guide/mixins.html.

43. *Mocha - the fun, simple, flexible JavaScript test framework* [online]. 2019 (cit. 2019-01-22). Dostupné z: https://mochajs.org/.

44. *Moment.js | Home* [online] (cit. 2019-11-25). Dostupné z: https://momentjs.com/.

45. MOURA, Marcos. *Vue Material - Material Design for Vue.js* [online]. 2019 (cit. 2019-03-02). Dostupné z: https://vuematerial.io/.

46. *Navigator.onLine - Web APIs | MDN* [online] (cit. 2019-04-19). Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/NavigatorOnLine/onLine.

47. 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/.

48. *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.

49. OGDEN, Cody. *Killed by Google - Google Graveyard - A Google Cemetery* [online]. 2018 (cit. 2018-12-23). Dostupné z: https://killedbygoogle.com/.

50. *OpenAPI Specification* [online]. 2017 (cit. 2019-10-17). Dostupné z: https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md.

51. *Overview | Vue CLI 3* [online]. 2019 (cit. 2019-02-11). Dostupné z: https://cli.vuejs.org/guide/.

52. *Prototypes, Specifications, and Diagrams in One Tool | Axure Software* [online]. 2018 (cit. 2018-03-07). Dostupné z: https://www.axure.com/.

53. *Protractor - end-to-end testing for AngularJS* [online]. 2019 (cit. 2019-01-22). Dostupné z: https://www.protractortest.org/.

54. *React - A JavaScript library for building user interfaces* [online]. 2018 (cit. 2018-12-16). Dostupné z: https://reactjs.org/.

55. *Release v2.0.0 - vuetifyjs/vuetify* [online]. 2019 (cit. 2019-07-27). Dostupné z: https://github.com/vuetifyjs/vuetify/releases/tag/v2.0.0.

56. ROEMER, Ryan. *Backbone.js Testing* [online]. 2018 (cit. 2018-12-28). Dostupné z: http://backbone-testing.com/.

57. *Routing* [online]. 2018 (cit. 2018-12-21). Dostupné z: https://vuejs.org/v2/guide/routing.html.

58. *Selection control components - Vuetify.js* [online]. 2019 (cit. 2019-09-24). Dostupné z: https://vuetifyjs.com/en/components/selection-controls.

59. *Sentry | Error Tracking Software — JavaScript, Python, PHP, Ruby, more* [online]. 2019 (cit. 2019-01-21). Dostupné z: https://sentry.io/welcome/.

60. *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.

61. *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/.

62. SCHUBERT, Jim. *OpenAPI Generator* [online]. 2019 (cit. 2019-10-17). Dostupné z: https://plugins.jetbrains.com/plugin/8433-openapi-generator.

63. *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.

64. *Test Utilities - React* [online]. 2019 (cit. 2019-01-22). Dostupné z: https://reactjs.org/docs/test-utils.html.

65. *The 3-Clause BSD License* [online]. 2019 (cit. 2019-12-29). Dostupné z: https://opensource.org/licenses/BSD-3-Clause.

66. *The MIT License* [online]. 2019 (cit. 2019-12-29). Dostupné z: https://opensource.org/licenses/MIT.

67. *Tilde Inc. - About Us* [online]. 2018 (cit. 2018-12-17). Dostupné z: https://www.tilde.io/about-us/.

68. *Unit Testing - Vue.js* [online]. 2019 (cit. 2019-01-22). Dostupné z: https://vuejs.org/v2/guide/unit-testing.html.

69. *UX Tool Time* [online]. 2018 (cit. 2018-03-07). Dostupné z: http://uxtooltime.com/.

70. VALENTA, Michal. *DBS – transakcní 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.

71. *Vue.js Material Component Framework — Vuetify.js* [online]. 2019 (cit. 2019-03-02). Dostupné z: https://vuetifyjs.com/.

72. *Web Content Accessibility Guidelines* [online]. 2008 (cit. 2019-11-23). Dostupné z: https://www.w3.org/TR/WCAG20/.

73. *webpack* [online]. 2019 (cit. 2019-02-10). Dostupné z: https://webpack.js.org/.

74. *What is Vuex? | Vuex* [online]. 2019 (cit. 2019-03-04). Dostupné z: https://vuex.vuejs.org/.

75. 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/`.

76. 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`.

77. YOU, Evan. *Vue - The Progressive JavaScript Framework* [online]. 2018 (cit. 2018-12-16). Dostupné z: `https://vuejs.org/`.

Dodatek A

Seznam použitých zkratek

API	Application Programming Interface
ARES	Administrativní registr ekonomických subjektů
BSD	Berkeley Software Distribution
CDN	Content Delivery Network
CI	Continuous Integration
ČR	Česká republika
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
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

Dodatek B

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 ”vložení” 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.

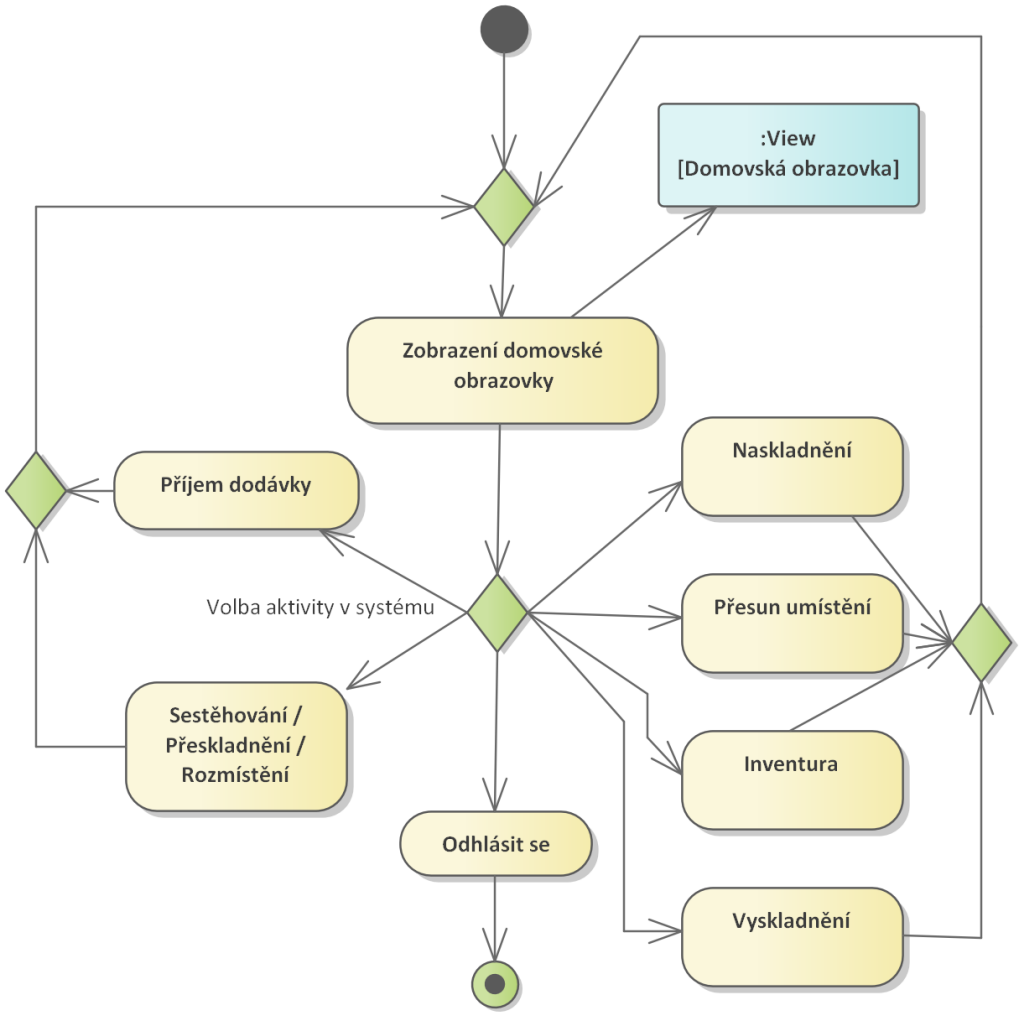
Git	Git je distribuovaný verzovací nástroj určený zejména pro sdílení a verzování zdrojových kódů aplikací, ale i jiných assetů.
GitHub	GitHub je webová služba podporující vývoj softwaru za pomoci verzovacího nástroje Git.
Heuristická analýza	Metoda testování sotfwaru, při které využívá expert své znalosti, 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 jazyka 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, která 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 údělem 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.

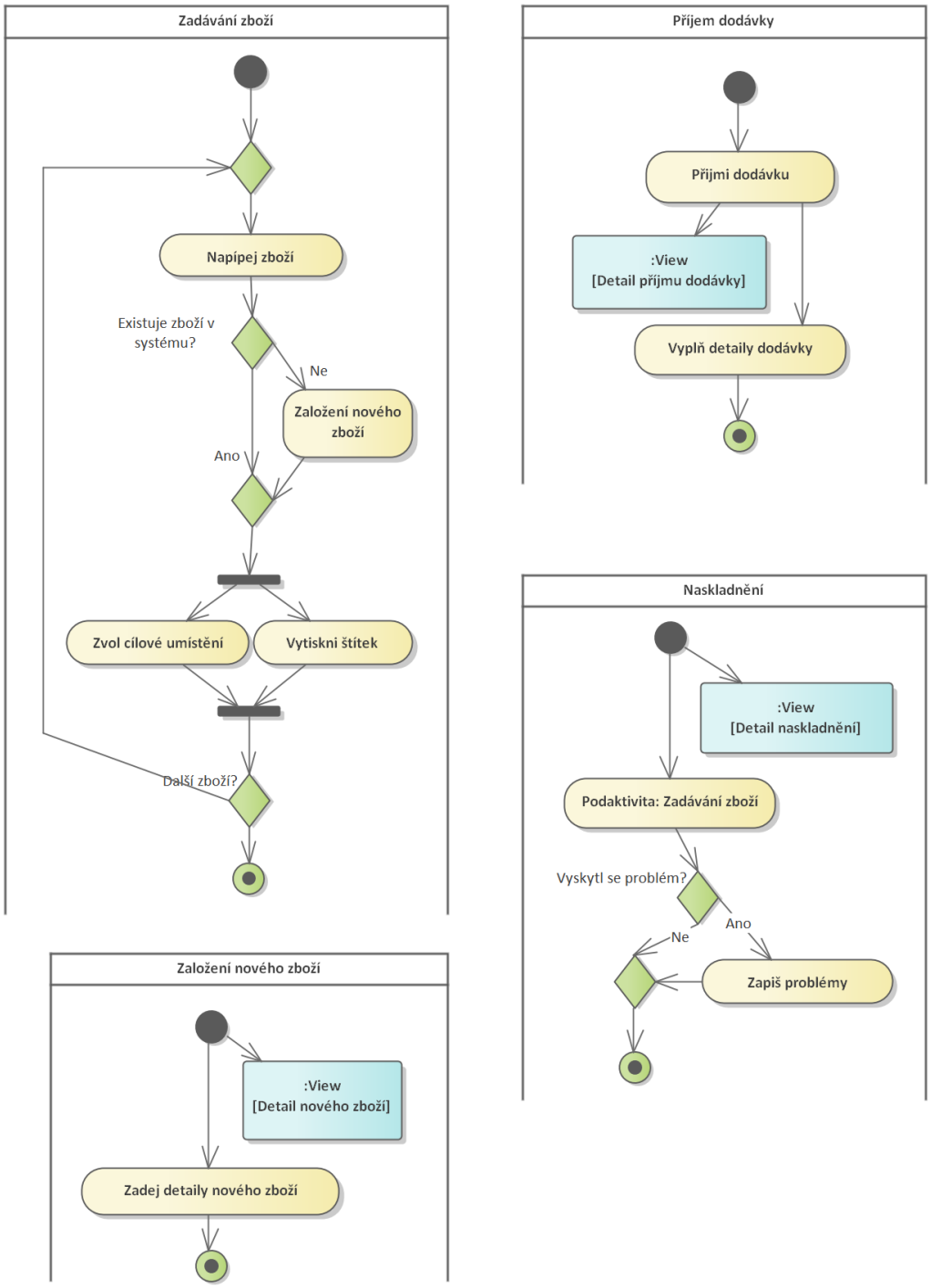
Vuex	Knihovna pro Vue.js, která realizuje state management pattern.
Webpack	Webpack je software, který zpracovává součásti webových aplikací a tvoří z nich balíčky vhodné pro webové prohlížeče. Primárně je zaměřen na Javascript, ale dokáže zpracovávat i řadu dalších formátů, přes styly v css či sass, 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.

Dodatek C

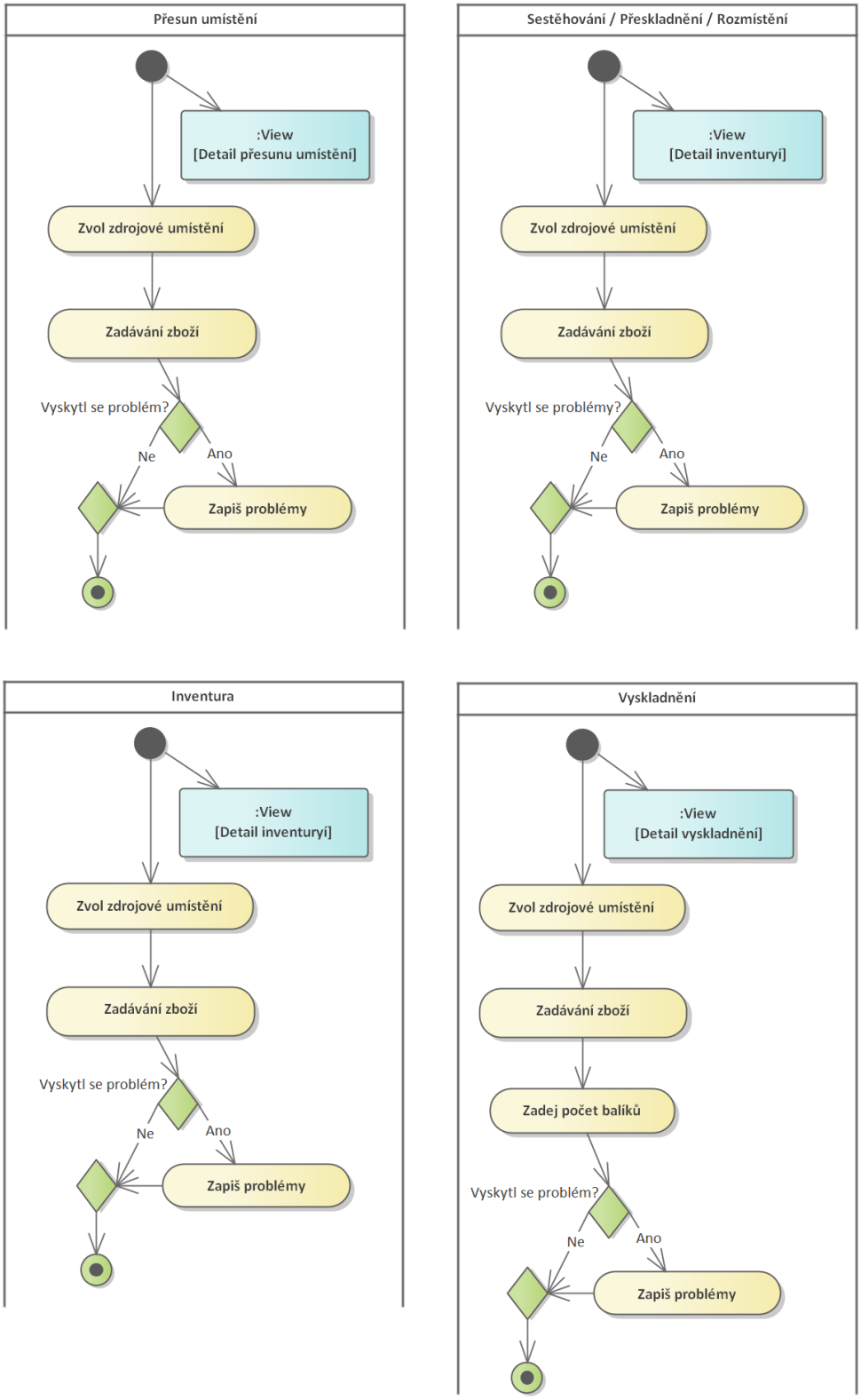
Activity diagramy průchodu systému skladníkem



Obrázek C.1: Activity diagram průchodu systému skladníkem: první část



Obrázek C.2: Activity diagram průchodu systému skladníkem: druhá část



Obrázek C.3: Activity diagram průchodu systému skladníkem: třetí část

Dodatek D

Zadání a zápis z uživatelského testování Hi-Fi prototypu

Úkoly pro testery

Představte si, že máte v ruce mobil, který umí číst čárové kódy pomocí zmáčkutí tlačítka na jeho straně. Touto čtečkou je možné načítat jak kódy balíků, tak názvy umístění. *(chcete-li simulovat čtení čtečkou, řekněte prosím průvodci testem, aby čtečku simuloval)*.

1. Naskladnění
    a. Šéf na Vás právě zakřičel přes půl skladu, že potřebuje vyřešit naskladnění číslo **#3222** - najděte a zvolte tuto úlohu.
    b. Vaším úkolem je načíst zboží, které přijelo, a stanovit každému výrobku pozici na skladě (rozmístění je na Vás - k dispozici jsou umístění A-3, A-4 a B-7). Čísla umístění je možné také načíst čtečkou, viz výše. Řidič vám postupně předává následující zboží:
        i. EAN: 85476, kusů: 11
        ii. EAN: 32486, kusů: 554 - musí být libovolně rozděleno alespoň na dvě umístění
        iii. EAN: 7451, kusů: 2
        iv. EAN: 98653, kusů: 1
        v. EAN: 11227, kusů: 4
        vi. EAN: 89569, kusů: 20
    c. Zkontrolujte, že na každém dostupném umístění (A-3, A-4 a B-7) je umístěn alespoň jeden z nově doručených balíků. Pokud není, nějakou položku přesuňte.
    d. Řidič za Vámi přispěchal s křikem a hlásí, že SSD disk od Samsungu Vám předávat neměl, že to koupil pro syna k Vánocům a omylem Vám ho dal s ostatním zbožím. Zrušte naskladňování této položky.
    e. Dokončete úlohu naskladnění.
2. Přesun umístění
    a. Vaším úkolem je přesunout zboží z umístění s názvem *“Tohle”* na umístění s názvem *“Tamto”*. Nalezněte v aplikaci příslušnou akci, vyplňte formulář a dokončete aktivitu.
3. Uzavřené úlohy
    a. Vaším úkolem je vyhledat úlohu Naskladnění od Firma s.r.o, které 30.11.2018 dělal Honza, najít číslo faktury a zkontrolovat, že byly naskladněny 3 druhy zboží.
    b. Stáhni si PDF naskladňováku.
4. Inventura
    a. Rozhodl ses, že se ti moc chce dělat inventuru. Najdi si inventuru s nejvyšší prioritou a proveď ji.
        i. Na umístění, pro které budeš dělat inventuru najdeš:
            - 30 kusů - “Boží zboží” (*EAN: 1111*)
            - 1 kus - “USB-C nabíječka” (*EAN: 1236*)
            - 50 kusů - “Samostatná klávesa K” (*EAN: 7651*)

ii. Potom, co jsi naskenoval všechny tyhle věci, ses nehorázně unavil a zbytek inventury už se ti prostě dělat nechce (tenhle úkol radši necháme jinému skladníkovi - tohle umístění je pro tebe moc náročné). Inventuru zruš z důvodu únavy řešitele.

5. Příjem dodávky
    a. Představte si, že k vám právě přijela přepravní služba a přivezla vám zásilku, kterou potřebujete zaevidovat do systému. Najděte v aplikaci akci, která vám umožní tuto úlohu provést.
    b. Dopravce vám předá dodací list, na kterém najdete číslo dodacího listu, jméno dodavatele, poznámku pro příjemce a další informace. Do aplikace zadejte následující data:
        i. Číslo dodacího listu: **145AC634-12ER43**
        ii. Dodavatel: **BBC**
        iii. Poznámku pro příjemce: Křehké zboží
    c. Nakonec dodací list vyfoťte.
    d. Protože jste pečlivý zaměstnanec, zkontrolujete, zda jsou všechny údaje správně vyplněné. Náhle však zjistíte, že jste špatně přečetli jméno dodavatele a místo dodavatele **BBB** máte v aplikaci dodavatele **BBC**. Odeberte dodavatele **BBC** a nahraďte jej dodavatelem **BBB**.
    e. Údaje opět zkontrolujete a zjistíte, že se již shodují. Dokončete příjem zboží.

Výstupy uživatelského testování

Záznamy obrazovek všech testování
https://drive.google.com/drive/u/1/folders/1NN7UjoTtebEuXvUVzC5MzpHGlj0OBeyW

Marek Erben

- erbenma1
- Student FIT, první ročník magisterského studia

1) Naskladnění:

- v inputu “umístění” chybí našeptávání,
- po skončení vyplňování inputu uživatel hledá, jak akci ukončit,
- formulář pro úpravu položky by se mohl rozevírat přímo u ní - nemusí skákat na první pozici. Po enteru očekává, že by se form mohl zase sbalit,
- je nejasné, že na druhé obrazovce je přehled. Přidat aktivní hlavičku podobnou jako na HP, jen menší,
- Usnadnit rozdělení napípnutých 540 položek na dvě umístění - možná rovnou z jedné karty?

2) Přesun umístění:

- největší problém bylo akci najít, tester netušil, že karty tam tvoří systém a nebo vedoucí. Po vysvětlení, že se jedná o “jeho iniciativu”, položku našel v menu vpravo nahoře.

3) Uzavřené úlohy:

- v kartách by uvítal více informací (datum, autor?).

4) Inventura:

- pochopil, že barvy jsou priorita, ale navrhuje přidat mezi jednotlivé barvy karet i malý header “střední priorita” atp.

5) Příjem dodávky:

- “Můžu ten dodák napípnout?”,
- “Je dodavatel povinná položka?”,
- “Můžu mít víc dodavatelů? - Myslím si, že ano, protože jsou u nich ty křížky, podobně jako u fotek.

Kristýna Miltnerová

- Žena, 21 let, studentka HR

1) Naskladnění:

- po vyplnění počtu kusů hned "tapne mimo" pro zavření klávesnice,
- na zpět možná dát potvrzovací dialog (chtěla se tak vrátit z druhé obrazovky a tím si přerušila úlohu,
- z druhé obrazovky netuší, jak se dostat zpět na první,
- nepochopeno, že položek naskladnění může být více v jednom tasku,
- kartu chce smazat přes dlouhý tap - což není špatný nápad.

2) Přesun umístění:

- bez problémů nalezeno.

3) Uzavřené úlohy:

- uzavřené úkoly hledá v seznamu všech, pak v menu nahoře,
- "je to nepřehledný. Vůbec netuším, co kde hledat",
- "proč tam není datum, když hledám dle datumu?",
- nenapadlo jí, že může použít hledání,
- pdf hledá v detailech, pak hned najde v tečkách.

4) Inventura:

- do modálu, který požaduje confirm EANu umístění, vyplňuje EAN zboží,
- zrušení bez problémů.

5) Příjem dodávky:

- akci najde bez problémů,
- “Ty čísla dodáků bych vyplňovat nechtěla - doufám, že to takhle nebude muset někdo používat” = znovu zjistit, zda jdou dodáky nějak napípávat...

Celkové hodnocení:

- “Pro někoho, kdo to nikdy neviděl, je to hrozně nepřehledný”,
- “Kdybych v tom dělala déle, tak to může být fajn”.

Dodatek E

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 syslem
  - 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 napípne neznámou položku, může Vytvořit nové zboží nebo Vytvořit umístění.
    2.5. U naskenovaného zboží může:
        2.5.1. měnit počet kusů,
        2.5.2. měnít 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 napípne 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 napípne 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 napípne 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 přeřadit 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

Dodatek F

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 fádní, 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 fotku,
  - 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 seznamech,
- zvýraznit priority (možná udělat barevně i ikonky),

- přidat vedoucímu možnost vytvořit samotné naskladnění (bez dodávky),
- inventura:
  - našeptávat umístění k napípání,
  - 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 dodáky 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 inveturizaci 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”.

Dodatek G

Dotazník pro testery alfa verze aplikace

Dotazník před testováním

1. Uveďte prosím základní údaje:
    - Pohlaví:
    - Věk:
    - Zaměstnání:
2. Pracujete běžně s nějakými skladovými systémy? S jakými?
3. Pokud nějaké ano, v čem vám dané systému vyhovují / nevyhovují?
4. Co nejvíce od skladového systému očekáváte?
5. Na čem trávíte ve skladovém systému nejvíce času?

Dotazník po testování

1. Co vám v novém systému nejvíce chybí?
2. Co se vám naopak líbí?
3. Jak se stavíte k možnosti buďto rozdělených rolí na skladníka a vedoucího, nebo podporu obou rolí v jedné osobě?

4. Vadí vám nutnost vše pípat, bez možnosti zadání přímo počtu kusů či zvolení produktu?
5. Využili byste vibrace zařízení při důležitých upozorněních / potvrzení?
6. Využili byste zkratky v systému pomocí skenu kódů kdekoliv v aplikaci? Např. načtením kódu spustit tvorbu úlohu příjmu dodávky apod.
7. Využili byste možnost nastavit si na hlavní stránce seznamy karet podle sebe?
8. Co vám přijde ve skladových systémech navíc, nebo že to většinou neumí pořádně?
9. Máte ve skladu zboží více různých zákazníků / majitelů?
10. Používáte ve skladu označená umístění?
11. Používáte šarže nebo sériová čísla skladových položek?

Dodatek H

Zadání úkolů pro testery alfa verze aplikace

1. Práce rozdělená na vedoucího a skladníka – naskladnění
    - Na PC nebo mobilu se přihlaste jako test_vedouci / test_vedouci.
    - Přišel Vám nový produkt, který ještě není v jiných systémech. Založte mu novou skladovou kartu:
        - vyplňte údaje, které běžně u skladové karty potřebujete,
        - přidejte i jeho čárový kód,
        - tento produkt je někdy i zabalen v kartonu, ve kterém jsou 4 kusy.
    - Vytvořte nový úkol „naskladnění“, čímž zaúkolujete skladníka, aby přijal toto nové zboží.
    - Na mobilu se přihlaste jako test_skladnik / test_skladnik.
    - Najděte před chvílí vytvořený úkol naskladnění a začněte na něm pracovat.
    - Naskladněte několik různých kusů výrobků na více umístění (podle konkrétního zadání) – použijte k tomu čtečku kódů.
    - Po dokončení úkolu se odhlašte.

- Přihlaste se jako vedoucí, a zkontrolujte tento úkol, tedy že naskladněné zboží odpovídá reálné situaci ve skladě, pokud ano, úkol schvalte, jinak ho vraťte a za skladníka chybu opravte.

2. Práce vedoucího a skladníka – vyskladnění
    - Jako vedoucí založte nový úkol vyskladnění
    - Způsob předání zvolte „Expedice“
    - Zadejte zboží k vyskladnění podle zadání (bude upřesněno)
    - Jako skladník tento úkol přijměte a posbírejte požadované zboží k sobě, zatím ho ale nepřemisťujte na umístění určené k expedici.
    - Vraťte se na seznam úkolů a dejte si oběd.
    - Po pauze na oběd se vracíte k práci, a úkol vyskladnění, jehož zboží máte stále u sebe, chcete dokončit – proveďte to.
    - Po dokončení se přihlaste jako vedoucí a zkontrolujte, kolik bylo na úkolu stráveno času.
3. Práce vedoucího a skladníka v jedné osobě – naskladnění
    - Přihlaste se jako test_master / test_master
    - Znovu vytvořte úkol naskladnění, dle zadání (bude upřesněno)
    - Rovnou po vytvoření začněte na úkolu pracovat, a dokončete ho bez nutnosti schvalování vedoucím – protože vy jste vedoucí.
4. Přesun mezi umístěními
    - Zvolte si, zda budete úkol zadávat skladníkovi, nebo použijete uživatele s oběma rolemi.
    - Vytvořte úkol „Přesun položek“ – dle konkrétního zadání (bude upřesněno)
    - Úkol realizujte a dokončete

Dodatek I

Zpracovaný výstup z uživatelského testování alfa verze aplikace

Seznam požadavků

- Expedice:
  - přidat možnost zadat váhu balíku,
  - upozorňovat, že dělám balení skladové položky, která je i v jiné expedici, která byla ale zadána dřív a je ještě nezpracovaná (FIFO),
- možnost dokončit vyskladnění částečně, tj. nedokončené položky by se vložily do nového úkolu,
- do webview aplikace přidat možnost rozsvítit světlo blesku přímo z aplikace,
- mít možnost u skladové položky vyplnit jednak oficiální název ale také skladový název, podle kterého se např. bude lépe položka hledat na skladě,
- přehled produktů může být moc dlouhý, když jich je na skladě hodně - rozumně omezit,
- k „může mít šarži“ přidat vysvětlení (nápovědu),

- mít možnost přidávat nové položky i pod tabulkou, které je vypisují,
- přidávání čárových kódů skladových položek by mělo být i v šipce vpravo dole, výchozí kódy rovnou v tabulce, ne až na prokliku. Stejná tabulka by mohla být i v úpravě produktu,
- při naskladňování položky bez fotky nabídnout, zda chce skladník fotografii přidat,
- při kolizi čárového kódu umožnit zobrazení kolizní položky,
- umožnit vlastní řazení sloupců na přehledu na domovské obrazovce,
- ikonku čárového kódu otočit o 90°,
- vedle šipky vpravo dole přidat nadpis „nový úkol“,
- ze seznamu umístění ve skladu mít akci, která zobrazí položky na konkrétním umístění,
- možnost nastavit si pořadí tvorby úkolů v šipce vpravo dole,
- umožnit zobrazit nahranou přílohu,
- umožnit nahrát více souborů postupně,
- umožnit měnit prioritu úkolu i po jeho vytvoření,
- v naskladnění zpřístupnit i přehled podle produktů a kam jsou naskladněny, ne pouze přehled umístění, a co na ně bylo umístěno,
- při dokončování úkolů mít možnost přiřadit i fotku,
- při volbě položek k vyskladnění či přesunu vypisovat jejich stručné informace ve volném místě vpravo,
- u umístění mít možnost evidovat, zda se na něm dá provádět expedice,
- snack zpráva o nově vytvořeném úkolu by měla obsahovat jeho ID a odkaz na něj,
- seznam umístění, kde lze požadovanou položku nalézt, řadit pod sebe, pouze jedno na řádek,

- přidat na bottom sheet křížek pro jeho zavření,
- automatické obnovování seznamů na domovské obrazovce,
- možnost zvolit komu úkol přiřadím, skupiny skladníků,
- po navrácení na domovskou stránku ponechat aktivní tu záložku, ze které byl úkol otevřen,
- při manipulaci se skladovými položkami v úkolech umožnit změnu jejich počtu i rozklikem položky, ne nutně napípnutím,
- při naskladňování přidat volitelně možnost použití skladníkova inventáře, aby viděl, co ještě musí umístit,
- při výběru umístění ve formulářích vypisovat i jeho kód
- při výběru položek k vyskladnění či přesunu umožnit je přidávat kliknutím na jejich název (ve výpisu „položek na skladu“),
- potom, co skladník zvolí, že bude pracovat na úloze, přepnout rovnou na druhou záložku úkolu,
- více zvýraznit, když se seznam úkolů načítá,
- informace o tom, co by se mělo skenovat, by měla být nejen pod vstupním polem, ale i přímo v něm,
- možnost aktivovat umístění, na které naskladňuji, pouhým kliknutím na něj, bez nutnosti skenovat kód,
- při přesunu položek na cílové umístění povolit rychlou akci, která přesune vše.

Seznam chyb

- Název umístění v přehledu hotových úloh je moc nevýrazný,
- překlopení Zebry na bok ukončí aktivitu - aplikace se poté musí znovu přihlásit,
- ve webview nefunguje otevření jiné přílohy než png, jpg,

- hláška o načítání na umístění je nejasná: mělo by být spíš „Načtěte produkty k umístění do …“,
- když mám přesouvat produkt, který už je i cílovém umístění, tak stále vypisuje chybu a nelze přesunout.

Návrhy na pokročilé funkce

- Umožnit zadat termíny, kdy pravidelně přijíždí konkrétní dopravce přebírající zásilky. Díky tomu by mohl systém upřednostňovat expedici podle termínu, kdy musí být balíky připravené.
- Umožnit současně pracovat na více úkolech.
- Preferování různých umístění při naskladnění: do zcela volných umístění apod.
- Podpora pro výměnu vratných obalů (sklenice, sodastream, apod.).
- Vedoucímu umožnit nastavit k výrobku speciální upozornění pro skladníka, které se mu zobrazí (např. zkontroluj, že je na krabici nějaký nápis apod.).
- Notifikace na nové úkoly.
- Vratky zboží, řešit především návaznosti šarží - tj. trvanlivost apod.,
- hromadná avíza: umožnit zadání, které zboží přijede a bude naskladněno. To ale bude najíždět postupně, a mělo by být vidět, co vlastně ještě chybí a má přijet.
- Umožnit skladníkovi zadávat si mikro-úkoly například na expedici, když se mu něco na umístění nezdá a chce to zkontrolovat.
- Kapacity umístění.

Dodatek J

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