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

24. září 2019

Poděkování

TODO Tým z NURů

Prohlášení

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

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

V Praze dne 24. září 2019 .....

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, 2019. Dostupný také z WWW: ⟨`https://gitlab.fit.cvut.cz/malecold/master-thesis`⟩.

Abstrakt

TODO

Klíčová slova TODO

Abstract

TODO

Keywords TODO

Obsah

Seznam ukázek kódu

Seznam tabulek

Úvod

TODO

Kapitola 1

Analýza požadavků

Cílem nového skladového systému je nahradit a rozšířit funkce stávajícího skladového systému Sysel - proto jsme při analýze požadavků vycházeli[1] 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 i realizace podpůrného systému. Výhodou tohoto řešení je možnost použití systému i v menších skladech a to případně i o velikostí pouze jednoho správce.

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

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 XY TODO.

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. TODO screen?

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

Kapitola 2

Návrh

2.1 Návrh uživatelského rozhraní - mockup

V poděkování jsem zmiňoval, že při prvotním návrhu UI jsme spolupracovali jako tým v předmětu MI-NUR. V rámci naší semestrální práce jsme navrhovali nový skladový systém vycházející ze Sysla, a zaměřili jsme se pouze na pohled skladníka.
První částí návrhu GUI byl mockup, či chcete-li wireframe. Pro jeho tvorbu jsme využili nástroj Axure RP [39] - 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 [50], což je knihovna pro Axure, která nabízí komponenty v *Material designu*. S její pomocí jsme vytvořili návrhy na základní procesy skladníka.

TODO https://zho703.axshare.com/

2.2 Návrh uživatelského rozhraní - HiFi prototyp

Dalším krokem návrhu uživatelského rozhraní byla tvorba HiFi 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.

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

2.2.1 Technologie 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 chtěl jsem pro naše řešení mít co nejméně závislostí, rozhodl jsem se vše napsat v čistém Javascriptu (pouze za použití dnes de-facto standardní knihovny jQuery). Základem je jeden výchozí HTML soubor, viz ukázka kódu 2.1.

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

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

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

```
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
      $(is).find('[class$="-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.html` z ukázky kódu 2.1) a veškeré výskyty tříd

`něco-include` se snaží nahradit obsahem souboru `něco.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[2], a tak pro použití například v Google Chrome je nutné zmiňovaný `index.html` spouštět přes libovolný webserver, nelze jej přímo otevřít z filesystému. Naopak například v Mozilla Firefoxu je CORS v tomto ohledu volnější, a celý tento mikro-framework tak lze používat bez jakéhokoliv lokálního serveru, stahování závislostí atp. - stačí pouze otevřít `index.html`.

2.3 Návrh pokročilých funkčností aplikace

2.3.1 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 jejích potvrzení je většinou nevratné a dojde k provedení něčeho důležitého. Co když je ale provádění důležitých akcí stěžejní funkčností systému? V příkladu skladového systému jsou veškeré přesuny zboží, naskladnění, vyskladnění atp. důležité akce, u kterých by si měl skladník zkontrolovat, že realita odpovídá stavu v systému. Potvrzovací dialogy na všech těchto akcích ale mohou uživatele i zdržovat - akce, která je v systému vykonávána s vysokou frekvencí, by neměla stále dokola vyžadovat dvojité potvrzení. Nejen že to bude uživatele zdržovat, ale i samotná četnost zapříčiní, že potvrzovací dialog ztratí svůj účel - uživatel bude zvyklý jej *ihned odkliknout*, aniž by se zamyslel nad tím, zda akci chce opravdu provést. (TODO zdroj?)

Undo Možnost vrátit akci zpět je svým způsobem volnější než potvrzovací dialog - nezdržuje dvojitým potvrzením, ale v případě nutnosti umožňuje špatné

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

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[3] zanalyzovali několik možností:

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

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

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

   Zhodnocení řešení: Jedná se o řešením, 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.

[3] spoluautor celého řešení - zabývá se backendovou částí aplikace

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

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

   Zhodnocení řešení: Třetí řešení je velmi podobné tomu druhému, změnou je, že když si bude frontu udržovat přímo backend, může ten s čekajícími požadavky již počítat při vracení dat jiným klientům (soft lock) - a tím předcházet nekonzistencím v datech: Když uživatel A vyskladní poslední kus výrobku, ale tento požadavek ještě čeká ve frontě, uživateli B se již bude zobrazovat, že je na skladě 0 kusů tohoto výrobků a nemůže ho tak vyskladnit duplicitně.
   Problémem tohoto řešení je velmi velká náročnost na zpracování těchto zámků a poměrně vysoká šance na neúmyslné chyby v kódu. Z toho důvodu jsme nakonec tuto variantu také 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 (TODO zdroj?).

   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 vůči výsledku. Z důvodu robustnosti 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, a požadavek na vrácení změn by byl zpracován jako zcela nová akce.

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

Po zhodnocení všech variant jsem se 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*, systému mu ani úkol dokončit nedovolí. U některých vybraných akcí půjde undo provést, a u zbylých, které vyhodnotím jako důležité a málo časté, kde nepůjde realizovat ani prevence chyb, ani undo, skončím u běžných potvrzovacích dialogů.

TODO možnost papínávat kódy pro rychlé akce v systému (IFTTT?)

TODO další návrhy funkčností

2.4 Volba technologie

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

Cílová platforma Aplikace je navrhována s ohledem na hardwarové vybavení skladu, ve kterém bude poprvé nasazována: zdejší skladníci jsou vybaveni mobilními telefony *Zebra TC25BJ*, které disponují OS Android 7.1 a vestavěnou čtečkou čárových kódů. Kromě skladníků by měla být aplikace použitelná také

z tabletu či stolního počítače pro účely vedoucích pracovníků. Z důvodu jednoduchosti vývoje, testování, možností aktualizací a obecně dobré zkušenosti z jiných projektů bylo hned při úvodním návrhu určeno, že aplikace bude tvořena formou webové služby, která bude na klientských zařízeních zobrazována ve WebView v jednoduchém kontejneru chovajícím se jako nativní aplikace. Řídící pracovníci budou naopak moci využít přístupu odkudkoliv, kde budou připojeni k internetu pouze pomocí běžného webového prohlížeče.
Z tohoto důvodu jsou v následující rešerši zhodnocovány frameworky či knihovny, které usnadňují vývoj *webových aplikací*.

Frameworky a knihovny V době psaní této práce patří mezi nejpopulárnější [12] [18] front-endové frameworky či knihovny Angular [3], React [41], Vue.js [56], Ember.js [15] a Backbone.js [8].

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

2.4.1 Datum vydání

Zatímco v současnosti nejčastěji porovnávanými frameworky jsou první dva zmíněné, Vue.js je z této pětice vybraných nejmladší, nabírá ale velké obliby. Ember.js a Backbone.js jsou poté lehce upozaděny z důvodu jejich stáří. Přehled prvního vydání jednotlivých frameworků je v tabulce 2.1

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

Tabulka 2.1: 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.

2.4.2 Zázemí

Zatímco Angular a React jsou vyvíjeny velkými společnostmi: Googlem, respektive Facebookem, které zná každý, Ember.js je vyvíjen společností Tilde Inc. [48], která také není žádným startupem. Vue.js a Backbone.js by se naopak daly nazvat *komunitními projekty*, neboť jsou vytvořeny převážně jedním autorem (Evan You, respektive Jeremy Ashkenas) a rozvíjeny a udržovány komunitou vývojářů.
Na první pohled by se mohlo zdát, že z tohoto hodnocení budou vycházet lépe ty frameworky, které mají za sebou stabilní firmy, neboť je tím zajištěn jejich kontinuální vývoj. Ve skutečnosti ale velké firmy *zabíjejí* své projekty poměrně často, stačí se podívat například na seznam projektů, které ukončil Google [37]. 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 2.2 se tedy opět zdržuji udělování bodů.

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

Tabulka 2.2: Volba frameworku: Zázemí

2.4.3 Licence

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

Licencování Reactu Facebook původně vydal svůj React pod BSD licencí spolu s dalšími patenty, avšak 24. září 2017 byl React převeden pod MIT licenci [1, 54].
Jelikož jsou všechny frameworky licencovány stejně, neprobíhá v tabulce 2.3 žádné bodování.

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

Tabulka 2.3: Volba frameworku: Licence

2.4.4 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* [19], ze které vychází hodnocení v tabulce 2.4.

Ember.js Tento framework je dle V. Lascika [32] vhodný spíše pro projekty, na kterých pracuje velké množství vývojářů, a to z důvodu své komplexnosti. Proto jej pro použití v mé vznikající aplikaci hodnotím nula body.

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

	Angular	React	Vue.js	Ember.js	Backbone.js
Obtížnost	vysoká	vyšší	nízká	velmi vysoká[a]	nízká
bodový zisk	*1*	*2*	*4*	*1*	*4*

Tabulka 2.4: Volba frameworku: Obtížnost

[a]za předpokladu, že na projektu bude pracovat pouze velmi malé množství vývojářů

2.4.5 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 [17] 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 [23], která je podobně jako u Angularu velmi bohatá a stromově strukturovaná.

Ember.js Oficiální manuál Ember.js [16] 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 2.5: Volba frameworku: Dokumentace

2.4.6 Testování

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

Angular Nejkomplexější z frameworků nabízí přehlednou dokumentaci [4], jak by měl být testován. Popisuje jak používat Unit testy i E2E testy a pro druhý zmíněný typ nabízí i samotný testovací framework: Protractor [40]. Jeho testovatelnost tedy hodnotím jako velmi dobrou.

React Tato knihovna ve své dokumentaci na první pohled testování příliš neprosazuje, avšak stránku dedikovanou této kratochvíli [47] 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 [29], také je ale zmiňován Enzyme [21] 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 [49] popisuje spouštění Unit testů pomoci vestavěných příkazů, které interně používají buďto již zmiňovaný Jest nebo Mocha [33], a dále odkazuje na vlastní kompletní testovací knihovnu [25]. Také je možné použít velké množství různých testových frameworků a aplikace ve Vue.js lze vyvíjet i pomocí TDD [31].

Ember.js Tento framework se ve své dokumentaci věnuje testování poměrně intenzivně a zasvětil mu rovnou několik stránek [22]. 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* [43], která již zmiňuje i zde dříve probírané frameworky jako *Mocha*. Ve výsledku tak bude pravděpodobně testování Backbone vypadat obdobně jako u ostatních frameworků, avšak přístup oficiální dokumentace snižuje jeho pochopení.

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

Tabulka 2.6: Volba frameworku: Testování

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

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

Tabulka 2.7: Volba frameworku: Devtools

[a]Stav k 24. 12. 2018

2.4.8 Počet hvězdiček na GitHubu

Počet hvězdiček na GitHubu lze velmi volně interpretovat jako oblíbenost frameworku mezi vývojáři. Z tohoto důvodu již v tabulce 2.8 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 2.8: 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.

2.4.9 Počet npm balíků

Npm [10] 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 2.9: Volba frameworku: Počet npm balíků

2.4.10 Otázky na Stack Overflow

Stack Overflow je jedním z portálů sítě Stack Exchange, který zná prakticky každý vývojář. Kdokoliv zde může položit otázku a komunita poté odpovídá, zatímco hlasuje o kvalitě odpovědí, aby byla vybrána ta nejlepší.
Z pohledu volby frameworku může být na jednu stranu vhodné, aby bylo na této stránce hodně otázek týkajících se dané technologie, na druhou stranu to ale může znamenat i nekvalitní dokumentaci. Jelikož ale v předchozí sekci nebyla žádná dokumentace vyhodnocena jako vysloveně špatná, budu dále usuzovat, že větší množství otázek je lepší.

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

Tabulka 2.10: Volba frameworku: Otázky na Stack Overflow

2.4.11 Firemní stack

Další zvolenou metrikou je, jak daná technologie zapadá do firemní stacku firmy Jagu s.r.o., ve které bude tento nový projekt realizován. Firma se specializuje především na webové aplikace a middlewary na zakázku [26], 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 2.11 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 2.11: Volba frameworku: Shoda s firemním stackem

2.4.12 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 2.12: Volba frameworku: Počet vývojářů na LinkedIn

2.4.13 Integrace se Sentry

Sentry [46] 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 [28] rychle zjišťujeme, že všech pět zde zkoumaných frameworků je oficiálně podporováno, včetně rychlého návodu na zprovoznění. Z toho důvodu neprobíhá v tabulce 2.13 žádné bodování.

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

Tabulka 2.13: Volba frameworku: Integrace se Sentry

2.4.14 Souhrn průzkumu

V tabulce 2.14 jsou sečteny body z předchozích dílčích hodnocení.

	Angular	React	Vue.js	Ember.js	Backbone.js
bodový zisk celkem	23	25	27	14	12

Tabulka 2.14: 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.

Kapitola 3

Implementace

3.1 Obecné součásti aplikace

Pro instalaci základu projektu jsem vycházel z dokumentace [23] a využil NPM [10]. Tím jsem získal základní kostru aplikace, kterou je možné spouštět ve vývojářském režimu (ten podporuje například hot-swapping, ale pro produkční prostředí se nehodí, neboť není rychlostně optimalizován). Nástroje pro tvorbu optimalizované verze jsou také samozřejmě dostupné - zatím ale nejsou pro mé použití potřeba.

Jelikož se zabývám aplikací středních rozměrů, je vhodné k tomuto základu přidat některé pokročilé funkcionality, které popíšu v následujících sekcích.

3.1.1 Router

Jednou z prvních komponent, kterou jsem k čistému Vue.js přidal, byl oficiální Vue Router [24]. Jelikož Vue.js je zaměřeno na tvorbu SPA - Single Page Application, bez konfigurace routeru by se všechny obrazovky aplikace zobrazovaly pouze na URL `/` - pokročilý uživatel by tak nemohl zadat například adresu `/task/2345`, což je vhodné i při ladění chyb - tester může jednoduše poslat zprávu *„na této adrese se špatně zobrazuje dodavatel“*.

Podobně jako chceme mít hezké URL, pro uživatele, který bude aplikaci ovládat z prohlížeče (tedy ne z aplikace), je žádoucí nastavovat správné titulky stránek. K tomu sice ve Vue Routeru neexistuje nativní podpora, ale řešení je opravdu

snadné - je popsáno v jednom z *Issues* v repozitáři na GitHubu [55] a spočívá v doplnění titulků do meta atributů cest:

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

Ukázka kódu 3.1: Nastavování titulků stránek pomocí Vue routeru - úprava definic

a dále úpravě samotné instance routeru:

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

Ukázka kódu 3.2: Nastavování titulků stránek pomocí Vue routeru - úprava instance routeru

3.1.2 Webpack

Webpack [52] 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.
Primárním důvodem, proč používat Webpack, je možnost rozdělování kódu do jednotlivých souborů. Z jiných programovacích jazyků jsme zvyklí mít oddělenou komponentu starající se o přihlašování, v dalším souboru mít podporu komunikace s externím API atp. Samotný Javascript sice samozřejmě umožňuje kód rozdělit do více souborů, avšak vše se spojuje pouze do jednoho kontextu prohlížeče, a tak může u větších projektů být matoucí, odkud se ta která závislost vlastně bere.

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 assetů použil *loader*, který například převede SASS na CSS, konstrukce ES6 na ES5 apod. [27].

Webpack ve spolupráci s Vue.js Základní instalace Vue.js využívá automaticky Babel [6] - to je kompilátor moderních verzí JS do starších, s kterými jsou kompatibilní prohlížeče. Pokud ale chceme využít pokročilé možnosti webpacku, je potřeba vytvořit soubor `webpack.config.js` kam zaneseme běženou konfiguraci webpacku - tedy seznam assetů, které budou zpracovávány a jednotlivé lodery, které je mají zpracovat.

3.1.3 Vue-CLI

TODO

3.1.4 Proměnné prostředí

TODO

3.1.5 Sentry

Sentry [46] 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.
Integrace s Vue.js je přímo podporována, a tak je integrace se Sentry záležitostí několika řádků kódu. V ukázce kódu 3.3 je vidět třída, kterou v projektu používám.

```
import Vue from 'vue'
import Raven from 'raven-js';
import RavenVue from 'raven-js/plugins/vue';
import Env from '@/service/Environment'

class SentryClient {

    #client = undefined;

    constructor() {
        if (Env.isProduction()) {
            // When this is enabled, there is **no console output**
            this.#client = Raven.config('<url>')
                .addPlugin(RavenVue, Vue)
                .install();
        }
    }

    captureMessage(msg, options) {
        if (Env.isProduction() && this.#client !== undefined) {
            this.#client.captureMessage(msg, options);
        }
    }
}

// TODO už je uplně jinak

const Sentry = new SentryClient()
;
export default Sentry;
```

Ukázka kódu 3.3: Třída pro zasílání aplikačních chyb do Sentry

Používání `class` a `#` v JS kódu V ukázce kódu 3.3 je použito klíčové slovo `class`, což je konstrukce představena až ve standardu ECMAScript 2015 [11] a výsledný kód se chová totožně jako třídní objekty z jiných jazyků. Dále se uvnitř třídy vyskytuje proměnná začínající symbolem `#` - zde se jedná o *proposed feature* [14] do budoucí verze ECMAScript a jedná se o privátní proměnné - opět tak jak je známe z jiných jazyků.

Samotné zavedení třídy z ukázky kódu 3.3 zajistí, že veškeré chyby, které vzniknout v produkčním prostředí, jsou automaticky zaslání do Sentry včetně veškerých detailů. Občas je ale vhodné odeslat i uživatelskou zprávu, která nemusí nutně být chybová, může jít pouze o informativní zprávu. Sentry client

toto umožňuje pomocí metody `captureMessage`, kterou výše zmíněné třída lehce obaluje. Použití v projektu je zaznamenáno v ukázce kódu 3.4.

```
1 import Sentry from "@/service/Sentry";
2
3 ...
4
5 Sentry.captureMessage('Your message');
```

Ukázka kódu 3.4: Zasílání vlastních zpráv do Sentry

3.1.6 Podpora WebApp

V kapitole 2.4 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 konfigurace WebApp nebyla potřeba, avšak pokud by někdo nepotřeboval čtečku čárových kódu - 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 aplikaci. Základní konfigurace tohoto manifestu je vidět v ukázce kódu 3.5.

```
{
  "short_name": "Swordfish",
  "name": "Swordfish - brána do skladového systému Atlantis",
  "icons": [
    {
      "src": "favicon/swordfish-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    }
  ],
  "start_url": "/",
  "display": "fullscreen",
  "orientation": "portrait"
}
```

Ukázka kódu 3.5: Manifest pro webové aplikace

TODO screenshot rozdílu?

3.1.7 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`[4]. [30]
Použití knihovny je pak obdobné jako známe z jiných jazyků. Máme definované klíče, které mohou být i vnořené (viz ukázka kódu 3.6) a ty následně používáme v šabloně (viz ukázka kódu 3.7)

```
{
    close: "Zavřít",
    home: "Domů",
    notFound: "Nenalezeno",
    lang: {
        change: "Switch to English",
        changeDone: "Jazyk: čeština"
    }
}
```

Ukázka kódu 3.6: Definice překladů pro i18n

[4]název je zkratka pro *internationalization* - číslovka 18 značí počet přeskočených znaků

```
<div
>    <h1>{{ $t("base.notFound") }}</h1>
    <router-link to="/">{{ $t("base.home") }}</router-link>
</div>
```

Ukázka kódu 3.7: Použití překladů v i18n

Bude-li v budoucnu někdy potřeba přidat další jazyky, stačí vzít pouze definice překladů dle klíčů a přeložit vše do nového jazyku. Zde je vhodné ještě zmínit, že pouhé překládání podle klíčů je sice rychlé, ale i tak by po nasazení nového jazyka měly být překlady zkontrolovány překladatelem přímo v aplikaci, a to navíc takovým, který rozumí cílové doméně. Pokud se tak neučiní, nastává často situace, kterou vídáme u zahraničních služeb, které expandují do Česka: překlady jsou dělané buďto strojově a nebo pouze dle klíčů, a v aplikaci se pak zobrazují slova, která bychom my jako Češi nikdy v daném kontextu nepoužili.

3.1.8 Material design

Nejrozšířenější grafickou knihovnou pro tvorbu webových a mobilních aplikací pro Android je bezesporu *Material Design* od Googlu.
První knihovna, na kterou jsem narazil, byla Vue Material [34]. Bohužel jsem záhy zjistil, že jelikož v době psaní práce (březen 2019) je teprve v beta verzi, a nemá implementované všechny komponenty, nakonec jsem se rozhodl pro její alternativu - Vuetify.
Jedná se o aktivně vyvíjenou knihovnu, která již podporuje veškeré základní prvky Material Designu a stále jsou přidávány i prvky nové [51].
Vuetify lze navíc jednoduše integrovat s již použitou knihovnou i18n.js popisovanou v předchozí sekci. TODO rozvést

3.1.9 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, a ty mezi sebou často potřebují komunikovat. Pěkný příklad je například *snackbar message*, někdy nazývaná také *toast message* či jednoduše *oznámení o provedení akce*. To je

komponenta, která musí být dostupná z jakékoliv jiné komponenty systému a vždy se musí zobrazovat na stejném místě a stejně se chovat. Je tedy žádoucí zpřístupnit její *stav* a to tak, aby se při změně stavu něco automaticky stalo, a aby změna stavu byla řízena jistými pravidly.

Vuex Vuex [53] je knihovna, která implementuje *State management pattern* pro Vue.js. Zařizuje jednotný přístup ke stavům komponent a umožňuje jejich řízené změny. Jeho použití v aplikaci skladového systému je vidět na ukázce kódu 3.8, 3.9 a 3.10, kde řeší zobrazování *snackbar message*.

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

Ukázka kódu 3.8: Vuex pro snackbar-message: definice

3.1.10 Hlídání konektivity

V moderních aplikacích, které veškerá data ukládají na API, je vhodné hlídat dostupnost tohoto API.
Prvním krokem k realizaci této funkce bylo zjistit stav připojení - tedy zda je aplikace online, nebo offline. Jako první jsem našel vlastnost prohlížeče `window.navigator.onLine` [35], která by přesně o tomto měla informovat a navíc poskytuje i možnosti poslouchat její změny pomocí běžných JS eventů. Po hlubším prozkoumání TODO TODO kontrola konektivity (proč nepoužívat browser properities, vlastní check, jeho náročnost na síť)

```
...
export default {
    name: "Snackbar",
    data: ...
    created: function () {
        this.$store.watch(state => state.snackbar.snack, () => {
            const msg = this.$store.state.snackbar.snack;
            if (msg !== '') {
                this.show = true;
                this.text = msg;
                this.$store.commit('setSnack', '');
            }
        })
    }
}
```

Ukázka kódu 3.9: Vuex pro snackbar-message: Snackbar komponenta

```
// Any Vue component
...
this.$store.commit('setSnack', '<message to display>');
...
```

Ukázka kódu 3.10: Vuex pro snackbar-message: použití z jiné komponenty

3.1.11 Renderování formulářů

Ještě před tím, než jsem začal tvořit formuláře ve své diplomové práci, jsem shodou okolností potřeboval upravit několik formulářů v jiné firemní 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.
Co se mi zejména 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()` [36] 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 atp. - tuto položku jsem tedy ponechal a smířil se s tím, že se bude u formulářů vždy kódovat ručně.
Stále ale ještě máme nutnost nastavit atribut v modelové třídě, v instanci formuláře a formulář nějak vykreslovat - tedy na třech různých místech: dvakrát v Typescriptu a jedenkrát v HTML. Má představa o jednoduše konfigurovatelném formuláři se ubírala směrem k vytvoření pouze jednoho konfiguračního souboru, odkud by se všechny tyto 3 věci generovaly, což se mi nakonec podařilo, a seznam jsem tedy stáhl na:

1. nastavit atribut v konfiguračním souboru,

2. nakódovat HTML, které atribut vypisuje.

Důležité je zde zdůraznit, že jsem odebral i nutnost vytvořit HTML kód formuláře: pokud po formulářovém prvku nejsou vyžadovány žádné nestandardní požadavky, jsou všechny atributy formuláře zpracovávány a vykresleny zcela automaticky pomocí komponenty, kterou jsem pro tento účel vytvořil.
V tuto chvíli je na místě projít ukázku kódu (3.11), která znázorňuje, jak může vypadat *definice formuláře* pro jednoduché skladové umístění:

```
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 3.11: Příklad definice formuláře: jednoduché skladové umístění

Rozdělení na `Form` a `FormRender` je zvoleno z toho důvodu, aby se oddělily logické celky: model (`Form`) a definice zobrazení (`FormRender`). Model se takto může celý při uložení poslat na API a při načtení se naopak celý přepíše daty z API. Definice zobrazení pak obsahuje informace, jak má samotný formulářový prvek vypadat a chovat se.
Takovýto konfigurační soubor se poté pouze načte v komponentě, která má formulář zobrazovat, a zde předá výše zmiňované komponentě pro standardizované vykreslování formulářů.
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 hlášky, která se bude zobrazovat pod formulářovým prvkem,

- *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 slušně upravovat a přizpůsobovat - pro mé potřeby vykreslování především formulářů na tvorbu či úpravu entit, které se v systému nacházejí: tj. skladů, dodavatelů, produktových karet, odběratelů, umístění apod., a dále formulářů zadávání a schvalování úloh, 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 dostupná v příloze TODO

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

TODO JS Flow

TODO Renderer formulářů

TODO text filtry

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

TODO kontextové menu

TODO exporty co CSV a JS

TODO renderer tabulek

TODO barvy accent a secondarz ale idempotence

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

3.2 Funkční specifikace

// TODO jednotlivé úlohy atp.

3.3 Perličky z vývoje

3.3.1 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 ssebou neslo poměrně hodně *breaking changes* [42], 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 narozdíl od oficiální dokumentace Vuetify [45] (kde checkboxy samozřejmě fungovaly), nerenderuje kus HTML, který je má na starost. Založil jsem si tedy nový lokální projekt s Vue.js + Vuetify, kde checkboxy samozřejmě také fungovaly. Postupně jsem tedy začal odebírat různé závislosti z npm, abych přišel na to, která knihovna tento problém

způsobuje. Při tomto procesu jsem rovnou zauditoval, zda opravdu potřebuji všechny dříve používané závislosti, a upravil i některé kusy kódu tak, aby závislosti již nebyly potřebné, a tedy jsem kód vlastně zefektivnil a zmenšil velikost výsledné aplikace. Stále jsem ale nemohl přijít na to, proč nefungují checkboxy.

Teprve asi po 6 hodinách a asi 40x přeinstalovanými všemi závislostmi, jsem se dostal k importu *Material Design Icons* [20]. Vuetify ve verzi 2.0 přidalo do možností své konfigurace klíč, který určuje, který ikonový font se má použit. Při migraci na novou verzi jsem použil ukázkové nastavení této hodnoty, tedy „mdi“. V žádném případě mě totiž nenapadlo, že *Material Design Icons (mdi)* a *Material Icons (md)* není to samé!

Po chvilce dalšího ladění s importem ikonek vyšlo najevo, že nastavení „mdi“ není kompatibilní s načítáním ikon z CDN Googlu, ale musí se použít balíček z npm. V případě, že chcete načítat ikonky z CDN, musí být hodnota *iconfont* nastavena pouze na „md“. Celý problém, na kterém jsem strávil tolik hodin a nechápavých výrazů šel tedy opravit diffem z ukázky kódu 3.12.

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

Ukázka kódu 3.12: Diff nastavení fontu ikonek ve Vuetify

Kapitola 4

Testování

4.1 Testování prototypu

TODO

4.2 Testování aplikace

4.2.1 První testování výsledné aplikace - během vývoje s odbornou osobou

Dne 19. 9. proběhlo první testování aplikace osobou, která mimo jiné činnosti pracuje také se starým skladovým systémem Sysel v roli vedoucího skladu.

Testování proběhlo při neformálním setkání v běžné kanceláři, testera jsem instruoval 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. 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. Této

detekci se podrobněji věnuje kapitola TODO.

Jediným výstupem z tohoto prvního testování vývojové verze aplikace je seznam postřehů, chyb a návrhů na zlepšení, které jsou k naleznutí v příloze TODO

Závěr

TODO

Zdroje

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

10. *Build amazing things* [online]. 2018 (cit. 2018-12-21). Dostupné z: https://www.npmjs.com/.

11. *Classes - JavaScript | MDN* [online]. 2019 (cit. 2019-03-02). Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes.

12. *Collection: Front-end JavaScript frameworks* [online]. 2018 (cit. 2018-12-17). Dostupné z: https://github.com/collections/front-end-javascript-frameworks.

13. *Comparison with Other Frameworks - Vue.js* [online]. 2018 (cit. 2018-12-14). Dostupné z: https://vuejs.org/v2/guide/comparison.html.

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

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

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

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

18. 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.

19. 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.

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

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

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

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

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

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

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

27. 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/.

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

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

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

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

32. 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.

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

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

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

36. *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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

54. 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/.

55. 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.

56. 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
BSD	Berkeley Software Distribution
CDN	Content Delivery Network
CI	Continuous Integration
CLI	Command Line Interface
CSS	Cascading Style Sheets
CORS	Cross-origin resource sharing
DI	Dependency Injection
E2E	End-to-end
ES	ECMAScript
FIT	Fakulta informačních technologií
GUI	Graphical User Interface
HTML	Hypertext Markup Language
JS	Javascript
MIT	Massachusetts Institute of Technology
MI	Magisterská Informatika
NPM	Node Package Manager
NUR	Návrh uživatelského rozhraní
OOP	Objektově orientované programování
OS	Operační Systém
SASS	Syntactically Awesome Style Sheets
SPA	Single Page Application
SVN	Subversion

TDD	Test-driven development
	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.
Hot Swapping	Jedná se o způsob zobrazení změn v aplikace 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.
Middleware	Software realizující integraci mezi dvěma jinými systémy, typicky pomocí API.
Sentry	Sentry je webová služba pro sledování chyb, které v aplikaci nastanou. Při použití v produkčním prostředí může vývojář díky Sentry o chybě vědět ještě dříve, než ji uživatel nahlásí, a to včetně všech detailů, jako například jaké kroky chybě předcházely, prostředí, ve kterém k chybě došlo a mnoho dalších.
Snackbar	Velmi podobný toast message, narozdíl od čistě informativního toast message umožňuje Snackbar uživateli spustit i programátorem definovanou akci. Zobrazuje se typicky ve spodní části aplikace a informuje o proběhlé akci.
Subversion	Systém pro správu zdrojových kódů a dalších assetů. Dnes se používá zejména pro ne-textové soubory, ty textové jsou většinou verzovány v gitu.
Toast message	Krátká informativní zpráva, který se objevuje ve spodní části aplikace a obsahuje typicky informaci o potvrzení provedení akce.

TypeScript	Nadmnožina JavaScriptu, která jej rozšiřuje především o statické typování proměnných a další atributy z OOP.
Use case	TODO
Vuetify	Knihovna pro Vue.js, která implementuje Material Design a poskytuje základní stavební komponenty, ale i pokročilé bloky jako například datové tabulky.
Vuex	Knihovna pro Vue.js, která se stará o state management.
Webpack	Webpack je software, který zpracovává součásti webových aplikací a tvoří z nich balíčky vhodné pro webové prohlížeče. Primárně je zaměřen na Javascript, ale dokáže zpracovávat i řadu dalších formátů, přes styly v css či sass, obrázky v png, jpeg, svg či konfigurace v json, yaml a dalších.
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

TODO Přílohy