Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Využití moderních client-side technologií – vývoj 3D návrháře nábytku

David Holada

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. David Holada

Systémové inženýrství a informatika
Informatika

Název práce

Využití moderních client-side technologií – vývoj 3D návrháře nábytku

Název anglicky

Usage of modern client-side technologies – development of 3D furniture designer

---

Cíle práce

Cílem práce je popsat možnosti využití moderních client-side technologií pro tvorbu interaktivních webových aplikací, v tomto kontextu dále podat přehled o výhodách současné generace klientských nástrojů proti generaci předchozí a jako výsledek předešlého demonstrovat využití těchto technologií na nově vytvořené webové aplikaci 3D návrháře nábytku.

Metodika

Práce sestává ze dvou částí – teoretických východisek a praktické části. Metodika práce zpracování teoretické část je založena na studiu odborných informačních zdrojů. Na základě syntézy zjištěných poznatků a praktických zkušeností budou popsány možnosti tvorby aplikací na straně klienta.

V praktické části bude proveden návrh a implementace aplikace v podobě 3D návrháře nábytku. Tato aplikace bude otestována a budou shrnuty zkušenosti z jejího vývoje a nastíněny možnosti případného dalšího rozvoje. Při zpracování této části práce budou využity standardní prostředky a metody softwarového inženýrství.

Doporučený rozsah práce

60-80 stran

Klíčová slova

HTML5, CSS3, JavaScript, JS, ES5, ES6, ReactJS, jQuery, EmotionJS, Styled Components, LESS, SASS, Node.js, MongoDB, PHP, MySQL, Canvas

---

Doporučené zdroje informací

Alex Banks, Eve Porcello, Learning React: Functional Web Development with React and Redux, O'Reilly Media, Inc., 2017. ISBN: 9781491954621

Michael Wanyoike, Build Your Own React Universal Blog App, SitePoint, 2018

Robin Wieruch, The Road to learn React: Your journey to master plain yet pragmatic React.js, 2017. ISBN: 172004399X

Sitepoint Team, Your First Week With React, SitePoint, 2017

Zac Gordon, React Explained: Your Step-by-Step Guide to React, 2019. ISBN: 1798752980

Předběžný termín obhajoby

2019/20 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 19. 2. 2020

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 2. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 02. 04. 2020

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Využití moderních client-side technologií – vývoj 3D návrháře nábytku" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 5. dubna 2020 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Jiřímu Brožkovi, Ph.D. za odborné vedení této diplomové práce.

Využití moderních client-side technologií – vývoj 3D návrháře nábytku

Abstrakt:

Tato práce pojednává o současných možnostech využití moderních client-side technologií při vývoji webových aplikací. V první části je krátce zmíněn historický vývoj těchto technologií. Větší zaměření je zde kladeno na porovnání dvou posledních přístupů k tvorbě těchto webových aplikací a sice staršího jQuery a novějších single-site application frameworků. Je také provedena komparace starší verze JavaScriptu ES5 a verzí ES6+, která s tímto problémem úzce souvisí. Ze single-site application frameworků je práce zaměřena hlavně na React, ve kterém je rovněž realizovaná praktická část práce – webová aplikace 3D návrhář nábytku. Užitečnost tohoto návrháře je demonstrována na průzkumu českého online nábytkářského trhu. Výsledkem tohoto průzkumu je zjištěn stav konkurenčních řešení. Tato konkurenční řešení jsou dále technologicky zanalyzována a následně je provedena formulace jejich nedostatků. Nově vzniknutá aplikace 3D návrháře nábytku, jejíž vývoj je v práci podrobně popsán, si klade za cíl tyto nedostatky eliminovat a vytvořit tak produkt, který technologicky převyšuje současná řešení.

Klíčová slova: HTML, CSS, JavaScript, ES5, ES6, React.js, jQuery, Emotion.js, Node.js, Express.js, SQL, MongoDB, MERN, WebGL, Three.js

Usage of modern client-side technologies - development of 3D furniture designer

Abstract:

This thesis talks about current client-side web development technologies usage options. In the first part of this thesis there is briefly mentioned the historic evolution of these technologies. Main focus is on the comparison of the last two client-side technological paradigms – an older jQuery and the new single-side application frameworks as well as on the comparison of ES5 and ES6+ versions of JavaScript which are tightly connected with this topic. Main focus in the terms of single-side application frameworks is on React which is used as a main library to develop the practical part of this thesis – 3D furniture designer web application. Usefulness of this application is proven by Czech furniture online market research. As a result of this research there is gathered base information about the rival application solutions on the market. These rival applications are technologically analyzed and there are formulated the cons of these solutions. Newly developed application 3D furniture designer which development will be described in detail targets on elimination of these cons and as a result of this there will be created technologically more advanced solution than any other application on the current market.

Keywords: HTML, CSS, JavaScript, ES5, ES6, React.js, jQuery, Emotion.js, Node.js, Express.js, SQL, MongoDB, MERN, WebGL, Three.js

Obsah

Seznam obrázků

Seznam tabulek

Seznam grafů

Seznam použitých zkratek

SPA: Single-page application(s)
JS: JavaScript
ES5: ECMAScript5
ES6: ECMAScript6
DB: databáze

ÚVOD

Ve vývoji softwaru, stejně jako v každém jiném oboru, dochází v průběhu času k vývoji technologií, pomocí níž lze dosáhnout stejných či dokonce lepších výsledků za stejný nebo dokonce lepší čas. Stejně tak se tomu děje i s JavaScriptem, jazykem pro tvorbu webových stránek na straně klienta, který prochází vývojem již od svého vzniku v roce 1995 (Aston, 2015) a který za poslední roky prošel jednou z největších transformací v jeho historii a to hlavně díky rozvoji tzv. single-page aplikací (SPA).

S rozvojem webu totiž v poslední době dochází k čím dál větší potřebě tvořit interaktivní aplikace. Jedním z prvních, kdo si této potřeby všiml, byl Google, který kromě dnes již historického Ajaxu z roku 2005, který kdysi využil pro Gmail a Google Maps, také přišel v roce 2010 s prvním masově využívaným frameworkem pro tvorbu single-page aplikací, s AngularJS (Gavigan, 2018). Netrvalo dlouho a se svým řešením přišel následně i Facebook, který v roce 2011 publikoval svoji vlastní knihovnu s názvem React. Dnes k těmto 2 gigantům můžeme započítat ještě třetí, v současnosti se hodně rozvíjející framework Vue.js, který byl vydán v roce 2014 (Hermans, 2018).

Byť se zdají být roky vydání těchto frameworků na IT poměry již historií, vždy přesto nějakou dobu trvá, než se technologie dostanou do širší praxe, a tak si vzpomínám, jak jsem ještě v roce 2013 pracoval na projektu, který by byl dnes typickým příkladem pro použití nějakého z těchto SPA JS framworků, tehdy však ještě bohužel byla pro vývoj vybrána o generaci starší technologie a sice jQuery.

Nyní o 6 let později v roce 2019 jsou již naštěstí tyto frameworky běžnou praxí, a tak když jsem dostal nedávno nápad na vytvoření podobné aplikace, na které jsem pracoval právě před 6 lety, ihned jsem věděl, po jaké technologii sáhnout.

Vybral jsem React.js a myslím, že pro účely této diplomové práce bude velice zajímavé porovnat to, jak je tento nástroj přínosný právě pro vývoj interaktivních webových aplikací, a to zejména v kontextu srovnání s přístupem starým. Aplikaci, která bude výsledkem tohoto snažení, jsem pojmenoval „3D návrhář nábytku“.

1 CÍL PRÁCE A METODIKA

1.1 CÍL PRÁCE

Cílem práce je popsat možnosti využití moderních client-side technologií pro tvorbu interaktivních webových aplikací, v tomto kontextu dále podat přehled o výhodách současné generace těchto nástrojů proti generaci předchozí a jako výsledek předešlého demonstrovat využití těchto technologií na nově vytvořené webové aplikaci 3D návrháře nábytku, jakožto technologicky vyspělejšího řešení proti současným řešení týchž návrhářů vyskytujících se na českém trhu.

1.2 METODIKA

Práce se sestává ze dvou částí – teoretických východisek a praktické části.

Metodika zpracování teoretické části je založena na studiu odborných informačních zdrojů. Na základě syntézy zjištěných poznatků a praktických zkušeností budou popsány současné možnosti tvorby aplikací na straně klienta, zejména pak v kontextu s historickým vývojem těchto technologií. Větší pozornost bude dále věnována porovnání rozdílů mezi posledními dvěma vývojovými stupni klientských technologií ES5/ES6 a jQuery/SPA. Ze SPA nástrojů bude věnován prostor hlavě knihovně React.js.

Výsledkem praktické části bude naprogramování React.js 3D návrháře nábytku. Za tímto účelem bude však nejprve proveden průzkum trhu českých nábytkářských společností, jehož výsledkem bude zjištění současné technologické situace konkurenčních řešení návrhářů nábytku. Průzkum trhu bude prováděn za pomoci standardních webových vyhledávačů Seznam a Google. Jednotlivá technologická řešení budou analyzována zejména za využití prohlížečových nástrojů pro vývojáře webového prohlížeče Firefox, zejména pak za využití modulů *„Průzkumník“*, *„Síť“* a *„Výkon“*. Pro potřeby následného zpracování dat získaných z této analýzy bude využito databázového dotazovacího jazyka SQL.

Tvorba 3D návrháře bude probíhat v operačním systému Ubuntu 18.04 za využití vývojového prostředí Atom. Podrobný postup tvorby 3D návrháře bude krok po kroku zaznamenáván do repositáře za použití verzovacího nástroje Git. Stěžejní části pak budou stručně popsány přímo v textu práce. Výčet jednotlivých commitů bude přiložen v příloze a výsledná aplikace bude umístěna na veřejně dostupnou webovou adresu. Finální verze

React návrháře bude nakonec řádně kvalifikována a kvantifikována vůči současným konkurenčním řešení. Na úplný závěr proběhne zamyšlení nad přínosem a zobecnitelností výsledků tohoto nově vytvořeného 3D návrháře nábytku.

2 TEORETICKÁ VÝCHODISKA

2.1 HISTORICKÉ POZADÍ JAVASCRIPTU

Historie JavaScriptu se začíná psát v roce 1995. V té době na trhu internetových prohlížečů dominuje prohlížeč Netscape s 80% podílem před konkurenčním Microsoft Internet Explorerem (Delaney, 2019) a to v internetovém světě, kde existují pouze statické HTML stránky bez jakékoliv dynamičnosti.

Vzniká proto požadavek na vytvoření jazyka, který by tuto dynamičnost obstaral. S požadavkem přichází právě tvůrci prohlížeče Netscape a vytvoření jazyka se ujímá Brendan Eich, který během 10 dní jazyk dokončí (Cassel, 2018) a položí tak základy dnešního JavaScriptu.

Tato první verze Javascriptu je ze začátku implementována pouze v domovském prohlížeči Netscape, čímž prohlížeč získává další výhodu nad tehdy konkurenčním Internetem Exporerem. Ten tak musí stav dohánět a v roce 1996 reversním inženýrstvím vydává kopii JavaScriptu pro svůj prohlížeč IE3 pojmenovanou JScript (Peyrott, 2017).

V roce 1997 je JavaScript standardizován jako ES1 (ECMAScript1). Tato standardizace pokládá základy JavaScriptu, jejíž velká část se používá dodnes.

V roce 1998 probíhá standardizace JavaScriptu verze ES2, ta však přináší pouze minoritní vylepšení.

V roce 1999 dochází ke standardizaci verze ES3, která nepřichází s ničím revolučním snad kromě implementace ošetření výjimek.

Na konci 90. let se však vývoj JavaScriptu jako takového dramaticky zpomaluje. V této době již totiž začíná ovládat trh s internetovými prohlížeči Internet Explorer (Anthony, 2011), který díky úspěšnosti svého operačního systému Windows 95 úspěšně předhání konkurenční Netscape. Microsoft totiž v tu dobu přichází s marketingově chytrým tahem, který zapříčiní zánik Netscapu. Přechází na nezpoplatněný model distribuce Internet Exploreru, který ve verzi 2.0 přidává zdarma ke všem kopiím svých OS (Naughton, 2015). Konkurenční Netscape je tak nucen přejít taktéž na nezpoplatněný model, což ve výsledku způsobí ztrátu jediných příjmů pro Netscape na rozdíl od

Microsoftu, který může nadále financovat svůj prohlížeč z prodejů operačního systému Windows.

Na trhu internetových prohlížečů tak zavládá na dlouho dobu prakticky monopol Internetu Exploreru, což stejně jako v jiných odvětvích, zapříčiňuje zpomalení technologického pokroku. Toto zpomalení neminulo ani vývoj JavaScriptu, jehož další stabilní verze tak vychází až v roce 2009 pod názvem ES5 (Flavio, 2018).

Přestože však na počátku nového tisíciletí vývoj JavaScriptu stojí na bodu mrazu, potřeba používat čím dál interaktivnější webové aplikace neustupuje. Google tak začíná v těchto letech pracovat na svých produktech Gmail (2004) a Google Maps (2005), pro které potřebuje dynamicky načítat obsah stránek spolu s daty ze serveru, a tak v roce 2005 vzniká termín AJAX (Asynchronous JavaScript and XML). Jedná se o JavaScript postavený okolo objektu *„XMLHttpRequest“* (Hoffmann, 2019). Oficiální specifikace AJAXu pak vzniká o rok déle v roce 2006.

Spolu s AJAXem však v roce 2006 vzniká ještě jedna dodnes hojně používaná technologie. Vzhledem k tehdejší špatné kompatibilitě JavaScriptu napříč prohlížeči vyvstává potřeba vytvořit knihovnu, která tuto kompatibilitu vyřeší, knihovna vzniká ještě v roce 2006 a je pojmenována jQuery. Nově vzniklá syntaxe nejenomže řeší nekompatibilitu napříč prohlížeči, ale zároveň její syntaxe ohýbá tehdejší JavaScript do mnohem použitelnější formy (Guo, 2019).

V roce 2009 nastává pro Javascript velice úspěšný rok. V květnu totiž Ryan Dahl vydává NodeJS, což poprvé v historii umožňuje běh Javascriptu nejen na clientu, ale zároveň také na serveru (Adam, 2017). Tímto je odstartováno tzv. Javascript only paradigma, což znamená, že celou část webové aplikace lze nyní již napsat pouze v JavaScriptu bez znalosti server-side programovacích jazyků (PHP, Python, .NET). V prosinci téhož roku navíc konečně vychází po dlouhých 10 letech nová verze JavaScriptu ES5. Tato verze přidává podporu funkcí pro lepší práci s poli spolu s funkcemi pro práci s formátem dat JSON a funkcemi pro práci s objekty (W3Schools, ECMAScript 5 - JavaScript 5).

V roce 2010 přichází Google s prvním SPA frameworkem AngularJS (Gudelli, 2019). To umožňuje začít psát velice interaktivní webové aplikace za pomoci mnohem pohodlnějšího a efektivnějšího zápisu než v jQuery či AJAXu.

V následujících letech nastává boom podobných frameworků. V roce 2012 tak vychází MeteorJS, v roce 2013 pak ReactJS (vyvinutý pod křídly Facebooku) a v neposlední řadě v roce 2014 Vue.js (Wanyoike, 2018).

Poslední větší revoluce na poli Javascriptu se odehrává v roce 2015, kdy vychází nová verze Javascriptu pod názvem ES6. Skok mezi touto verzí a verzí předchozí je proti předchozím změnám mezi verzemi naprosto diametrální. ES6 nepřichází jen s pár novými funkcemi, nýbrž s celou plejádou nových možností.

Od vydání ES6 vychází každým rokem nová verze Javascriptu, která však pokaždé přináší víceméně pouze minoritní změny (Ramanadham, 2019). Stále tak platí, že největší pokrok v Javascriptu se udál ve verzi ECMAScript 6. Dnes, 4 roky od vydání ES6, je při dodržování ročního vydávacího cyklu verzí posledním standardem Javascriptu ECMAScript 10 vydaný v červnu 2019.

2.2 SOUČASNÝ STAV NA POLI JAVASCRIPTU

Vývoj tedy dovedl Javascript do stavu, který je v současné době definován single-page aplikacemi. SPA, tvořenými v SPA frameworcích, nástupcích starších technologií jako ES5 a jQuery. Co však tento zatím poslední technologický skok přináší?

2.2.1 Rozdíl mezi ES5 a ES6

Mezi verzemi ES5 a ES6 došlo k několika důležitým změnám. Níže budou popsány hlavní z nich (Banks, 2017, s.10 - 28).

Zápis řetězců a proměnných

ES5 zápis:

```
const veta = 'Hodnota proměnné A je ' + promennaA + ' jednotek.';
```

ES6 zápis:

```
const veta = `Hodnota proměnné A je ${promennaA} jednotek`;
```

Zde je již na takto primitivním příkladu vidět, že nový způsob zápisu je mnohem přehlednější a kratší. Zápisu se říká *„template literals“* někdy také *„object literals“* a umožňuje výpis proměnné pomocí znaku *„$“* a složených závorek na rozdíl od předchozí „plusové“ notace. Nový zápis taktéž umožňuje zápis řetězce na několik řádků, což bylo v ES5 problémem. (Banks, 2017, s.20).

Zápis funkcí

ES5 zápis:

```
function sectiDveCisla(a,b) {
        return a + b;
}
```

ES6 zápis:

```
const sectiDveCisla = (a,b) => a + b;
```

Novému zápisu se říká *„arrow functions“*. (Banks, 2017, s.14). Tento zápis je oproti ES5 zápisu opět mnohem úspornější. Problémem sice může být na první pohled zvýšená míra abstrakce oproti staršímu ES5 zápisu. Naštěstí je tato abstrakce vykoupena získáním větší přehlednosti, a to hlavně v kontextu programování v single-page application frameworcích.

Přidání nových typů proměnných (let a const)

V ES5 existoval pouze jediný způsob, jak nadefinovat proměnnou, a to pomocí klíčového slova *„var“* (Banks, 2017, s.10):

```
var a=10;
```

Zde je potřeba zmínit, že každá nadefinovaná proměnná v Javascriptu má jakýsi obor platnosti. U definice pomocí klíčového slova *„var“* je tento obor platnosti (v angl. variable scope) určen funkcí, ve které je proměnná definována. Z toho vyplývá, že všechny proměnné definované mimo funkce jsou globální (přístupné odevšad).

ES6 proto přišla s 2 novými způsoby definice proměnných. Jedná se o *„let“* a *„const“* (Chima, 2017). Tyto nové 2 způsoby definice proměnných již nemají obor platnosti určen funkcí, nýbrž blokem, ve kterém se nachází. Blokem se nazývá jakýkoliv úsek kódu mezi složenými závorkami *„{}“*.

```
let a=10;
const a=10;
```

Druhou změnou definic proměnných v ES6 je, že *„let“* a *„const“* nemohou být proti „var“ znovu deklarovány, což je velice přínosné z hlediska hlídaní jedinečnosti názvů proměnných. Nestane se tak např., že je někde na začátku kódu deklarována jedna

proměnná a po tisících řádcích dále tato proměnná redefinovaná, protože se na použití jejího názvu na začátku souboru již zapomnělo.

Rozdíl mezi *„const“* a *„let“* pak spočívá v možnostech redeklarace. Zatímco u *„let“* lze měnit hodnoty proměnné v průběhu algoritmu u *„const“* měnit hodnoty nejde. Přes *„const“* by tedy měly být definovány vyloženě konstanty, tedy proměnné s hodnotou, u níž se nepočítá s její změnou hodnoty v průběhu běhu algoritmu. Jelikož tento typ proměnné nejde měnit, zvyšuje se tak také bezpečnost aplikace, protože proměnné nemohou být přepsány externími skripty (Salvet, 2017).

Importy

ES5 zápis:

```
var module = require(./module);
```

ES6 zápis:

```
import module from './module';
import { module1, module2 } from './modules';
```

Import došel v ES6 specifikaci kromě lepší syntaxe také k možnosti elegantního importu několika různých „modulů“ z jednoho souboru (Banks, 2017, s.27).

Podpora ES6 v prohlížečích

Je však potřeba upozornit na jeden důležitý fakt. ES6 bohužel není podporován všemi prohlížeči. Podporu ES6 v současných prohlížečích zachycuje následující tabulka (Tabulka 1):

Browser	Version	Date
Chrome	51	May 2016
Firefox	54	Jun 2017
Edge	14	Aug 2016
Safari	10	Sep 2016
Opera	38	Jun 2016

Tabulka 1 - Podpora ES6 mezi prohlížeči (W3Schools, Javascript Versions)

Tyto „ES6 ready“ prohlížeče pak dle metrik caniuse.com používá v současnosti 92.07% internetové populace (CanIUse, 2019). To znamená, že nativní ES6 nefunguje zhruba 8 % uživatelů.

Aby se tento problém vyřešil, používají se tzv. Javascript compilery, které dokážou ES6 kód převést na ES5 a umožnit tak nový kód interpretovat i ve straších prohlížečích. Mezi nejznámější z nich patří např. Babel (babeljs.io).

2.2.2 Single-page application frameworky vs JS/jQuery

Hlavní výhodou single-page application frameworků oproti čistě JS/jQuery přístupu je rychlost, konkrétně rychlost změn v DOMu. Aktualizace DOM stromu je totiž relativně pomalá záležitost. Při každé změně DOMu se totiž musí porovnávat starý DOM strom s tím novým, aby se zjistilo, co se má změnit a teprve poté se změny z jednoho stromu přenášejí na nový aktualizovaný strom. Problém je, že se takto musí aktualizovat celá stromová struktura DOMu a algoritmy, které toho jsou schopny mají výpočetní složitost zhruba n^3 (React Docs, Reconciliation). Se zvyšujícím se počtem prvků tak velice rychle narůstá počet operací a pokud se aktualizuje nějaká složitější HTML struktura, začíná mít běžný způsob problém.

Proto přišel React s něčím nazývaným jako VirtualDOM (React Docs, Virtual DOM and Internals). Tato struktura, jak již název napovídá, představuje také DOM, ale virtuálně, tzn. existuje pouze v paměti. Navíc je jeho struktura odlišná oproti standardnímu DOMu. VirtualDOM je totiž ve své podstatě Javascriptový objekt (JSON), který ukládá stejné informace jako DOM, ovšem aktualizace VirtualDOMu oproti standardnímu DOMu mohou probíhat několikanásobně rychleji. Této vlastnosti tak bylo využito při vymýšlení principu VirtualDOMu, kdy jsou všechny změny v DOM stromu prvně předpočítány ve VirtualDOM stromu a až poté je tento celý předpočítaný VirtualDOM nahrazen za standardní DOM. Vzhledem k tomu, že aktualizace VirtualDOMu má výpočetní složitost pouze n, oproti n^3 u standardního DOMu (React Docs, Reconciliation) je dosahováno tímto způsobem mnohem lepších výsledků.

Ale single-page aplikace nevynikají pouze svojí rychlostí. Jejich další neméně důležitou devizou je struktura a logika aplikací. Všechny části aplikace se totiž dělí na jednotlivé komponenty. Tento přístup přispívá k lepší přehlednosti, udržitelnosti i znovupoužitelnosti (React Docs, Component and Props).

2.2.3 React

Porovnávání jednotlivých single-page frameworků a následné rozhodování, který použít, by vydalo za samotnou práci, pro účely této diplomové práce tak byl bez hlubší

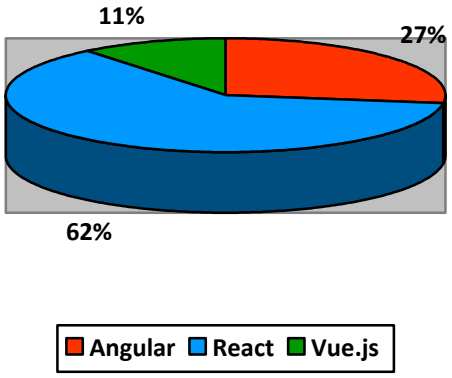
analýzy zvolen React. Níže bude popsáno alespoň základní rozhodovací kritérium pro volbu právě tohoto frameworku a v neposlední řadě bude také dodrženo schéma této kapitoly a zdůrazněny nové přístupy ke tvorbě aplikací, kterými nedávno prošel i samotný React.

Proč právě React?

Jednoduchou odpovědí na tuto otázku jsou dvě slova „trh práce“. React je zkrátka v současnosti nejžádanější znalostí na poli vývoje JS aplikací.

Dle nabídek práce na Startupjobs.cz, což je portál specializovaný na nabídky zaměstnání v IT, šlo v lednu 2020 nalézt 57 nabídek práce odpovídající hledání s tagem *„Angular“* (StartupJobs, AngularJS), 130 nabídek s tagem *„React“* (StartupJobs, React JS), na tag *„Vue.js“* pak bylo pozitivních pouze 23 výsledků (StartupJobs, Vue.js). Procentní vyjádření těchto výsledků zachycuje Graf 1 níže.

React je tedy i dle těchto dat nejžádanější znalostí na trhu práce, a to je již samo o sobě celkem silným argumentem na to programovat právě za pomocí tohoto frameworku.



Graf 1 - Procentuální zastoupení nabídek práce dle jednotlivých technologií dle portálu Startupjobs.cz (vlastní zpracování dle dat Startupjobs.cz)

Novinky na poli Reactu

Nejenom, že dochází ke skokům mezi používanými technologiemi, ale taktéž dochází k vývoji uvnitř těch stávajících. To je případ i Reactu, který ve verzi 16.8 z 16. února 2019 doslova naboural dosavadní paradigma tvoření jeho komponent (React Docs, Using the State Hook).

S příchodem React Hooks, totiž umožnil psát plně funkční komponenty bez nutnosti používání tříd. Nově tak lze taktéž plně využívat funkcionální komponenty. Tyto

funkcionálně psané komponenty navíc odstranily mnoho neduhů, kterými trpěly komponenty psané právě pomocí tříd.

Komponenty psané přes třídy totiž, pokud chtěly někde dále ve svém kódu volat vlastní funkce, tak je musely napřed nabindovat pomocí „*bind()*“. Stav komponent se pak měnil přes funkci „*setState()*“ a aby šel stav vůbec používat, musela se volat prvně funkce „*construstor()*“ a „*super()*“. Vzorová komponenta zapsaná pomocí tříd by tak mohla vypadat např. takto:

```
import React from 'react'

class News extends React.Component {
  constructor() {
    super()
    this.state = {
      state: 1
    }
    {/* this.exampleMethod = this.exampleMethod.bind(this); */}
  }

  render() {
    return (
      <>
        {this.state.state}
        <button onClick={() => this.setState({ state: 2 })}>
          Change state
        </button>
      </>
    )
  }
}

export default News
```

Naproti tomu ve funkcionálních komponentách pro definování funkcí odpadá nutnost bindování funkcí a pro použití stavu, který by ještě před verzí 16.8 ve funkcionální komponentě nebylo možné použít, se využívá hooku „*useState()*“. Vzorová komponenta zapsaná pomocí funkce by tak mohla vypadat např. takto:

```
import React, { useState } from 'react'

function News() {
  const [state, setState] = useState(1)

  return (
    <>
      {state}
      <button onClick={() => setState(2)}>Change state</button>
    </>
  )
}

export default News
```

Jak je již z takto primitivního příkladu vidět, nový funkcionální zápis je mnohem kratší, přehlednější a pohodlnější pro zápis.

2.3 CSS NA SOUČASNÉM FRONTENDU

Frontend však neprošel pouze vývojem Javascriptu, vedle toho také za poslední dobu došlo k výraznému pokroku na poli kaskádových stylů. Zejména díky nástupu CSS3 a v souvislosti se single-page aplikacemi pak díky vydání různých CSS in JS frameworků.

2.3.1 Flexbox

Bezpochyby jednou z největších revolucí ve specifikaci CSS3 je *„flexbox“*. Flexbox je v CSS definován za pomoci hodnoty *„flex“* vlastnosti *„display“ (display: flex)*, která otevírá nové široké možnosti, jak skládat a zarovnávat strukturu HTML dokumentu.

Jedná se vlastně o první účelově vytvořenou CSS vlastnosti pro vytváření struktury webu. Do té doby používané různé jiné vlastnosti jako např. *„display: inline-block“* či *„float“* se totiž pro tyto účely pouze všemožně „ohýbaly“, avšak nikdy k němu nebyly prvoplánově vytvořeny, proto také práce s nimi byla velice omezená a v některých případech až frustrující. To je dnes naštěstí již minulostí.

Flexbox je vlastně sada vlastností, které lze aplikovat na všechny potomky elementu s definovanou CSS vlastností *„display: flex“* (CSS-tricks, A Complete Guide to Flexbox). Tomuto elementu se říká *„flex container“* a všichni jeho přímí potomci se pak označují jako *„flex items“*.

Flexbox vlastnosti pak můžeme rozdělit do 3 základních kategorií podle toho, jakou oblast vzhledu upravují.

CSS vlastnosti ovlivňující řazení flex itemů

flex-direction

Tato vlastnost určuje, zda se elementy ve flexboxu budou zarovnávat do sloupce *(„column“)* nebo do řádku *(„row“).* Výchozí chování je do řádku (W3Schools, CSS flex-direction Property).

```
flex-direction: row;
```

order

Jednotlivým elementům jde navíc díky vlastnosti *„order“* stanovit pozici (pořadí), na které se mají zobrazit, tím lze docílit posouvání pořadí zobrazování elementů bez nutnosti upravovat pořadí v HTML (W3Schools, CSS order Property). Např.

```
order: 3;
```

by tak zobrazil prvek na 3. pozici.

flex-wrap

Ve výchozím nastavení flexboxu se všechny elementy snaží vejít do jednoho řádku, vlastností *„flex-wrap“* však lze vynutit, aby se zalomily na řádek další, pokud již na jednom řádku není dostatek prostoru (W3Schools, CSS flex-wrap Property).

```
flex-wrap: nowrap;
```

CSS vlastnosti ovlivňující zarovnání flex itemů:

justify-content

Tato vlastnost rozhoduje o rozložení jednotlivých flex itemů v rámci jednoho řádku. Výchozí hodnotou je *„flex-start“* (W3Schools, CSS justify-content Property).

align-items

„Align-items“ je vlastně takový protipól pro *„justify-content“,* také rozděluje prostor mezi jednotlivými flex-itemy stejně jako *„justify-content“,* nýbrž ne ten horizontální, ale vertikální. A aby to nebylo tak jednoduché tak pouze za předpokladu *„flex-direction: row“,* při *„flex-direction: column“* se kontext vertikality a horizontality otáčí (W3Schools, CSS align-items Property).

align-self

Je obdobou *„align-items“* s tím rozdílem, že oproti *„align-items“* se aplikuje pouze pro prvek, kterému je hodnota nastavena, tzn. nastavuje se flex itemu a ne flex containeru (W3Schools, CSS align-self Property).

CSS vlastnosti ovlivňující rozměry flex itemů:

flex-grow

Umožňuje nadefinovat, kolikrát se má flex item **zvětšit** relativně proti ostatním flex itemům ve stejném kontejneru (W3Schools, CSS flex-grow Property).

flex-shrink

Naopak „flex-shrink“ udává, o kolik se má flex item **zmenšit** relativně proti ostatním flex itemům ve stejném kontejneru (W3Schools, CSS flex-shrink Property).

flex-basis

A nakonec *„flex-basis“* taktéž udává, jak se má změnit rozměr flex-itemu, ovšem specifikuje absolutně výchozí velikost flex itemu (W3Schools, CSS flex-basis Property).

2.3.2 CSS in JS frameworky

CSS sice prošlo v zatím poslední iteraci své specifikace (CSS3) obrovskými změnami (Kyrnin, 2020), nicméně jeden problém, který vyvstal v souvislosti se single-page aplikacemi i tak neřeší a konec konců ani nemůže, jde o strukturu a přehlednost stylopisů.

Samotné CSS má s úsporným, přehledným a udržovatelným zápisem problémy již od svého vzniku, proto už tu přes 10 let existují CSS preprocesory jako SASS či LESS a jiné další, které do CSS přinesly spoustu skvělých nových funkcionalit (LESS, overview).

Mezi nejužitečnější z těchto vylepšení patří asi možnost definovaní vlastních proměnných, zanořování jednotlivých CSS bloků do sebe či používání *„mixins“* (proměnné obsahující několik CSS pravidel). Vyjmenovávat zde však všechny možnosti CSS preprocesorů či jednotlivé dokonce popisovat už v kontextu dnešní doby nemá smysl, přeci jenom tu jsou s námi tyto technologie již nějakou tu dobu.

Mnohem zajímavější technologií dneška z pohledu CSS jsou však CSS in JS frameworky, vždyť první známější z nich Styled Components byl vydán teprve v roce 2017 (Github, 2017). Tyto frameworky, jak již jejich název napovídá, přistupují k CSS úplně odlišnými přístupem. Nepoužívají totiž externí stylopisy v separátních souborech, jak bylo doposud v CSS zvykem, ale zapisují styly přímo do JS souborů.

Toto se může zdát na první pohled jako velký krok zpátky, vždyť psát interní styly bylo vždy v CSS považováno za tzv. bad practice (Kyrnin, 2020). Ovšem v kontextu single-page aplikací tento postup začíná dávat opět smysl.

Hlavním důvodem, proč tento přístup opravdu dává smysl, je přehlednost a udržitelnost kódu. Single-page aplikace totiž fungují na komponentovém přístupu, což ve zkratce znamená, že aplikace obsahuje spousta JS souborů s jednotlivými funkcionalitami a ty jsou následně znovupoužitelné (React Docs, Components and Props).

Výchozí situace bez použití CSS in JS frameworku je pak následující. V aplikaci existuje spousta JS souborů se spoustou CSS pravidel. Tyto CSS pravidla jsou pak definována v nějakém externím stylopisu. Časem však vývojář zjistí, že mít jeden soubor se stylopisem je nepřehledné, a tak přirozeně rozdělí stylopisy na několik souborů, což v té situaci vypadá jako rozumné řešení, konec konců lepší řešení bez CSS in JS frameworku pravděpodobně ani neexistuje. Bohužel ani tento způsob organizace stylů není nejvhodnější. Pro každou komponentu (každý .js soubor) totiž musí vývojář otevírat jiný soubor se styly, a to ve výsledku práci zpomaluje a způsobuje nepřehlednou.

CSS in JS framworky proto přišly s ideou zapisovat styly přímo do JS komponent, pokud pak chce vývojář upravit nějakou část vzhledu aplikace, má jak styly, tak HTML a JS na jednom místě, což je pro udržitelnost a přehlednost velký přínos. Nicméně tím výčet výhod nekončí.

Druhou důležitou výhodou tohoto přístupu je, že se v jednotlivých komponentách mohou podmínkovat rovnou CSS pravidla např. podle aktuálního stavu komponenty. To za předpokladu přístupu s externím stylopisem nelze, a tak se toto dosud obcházelo podmínkováním CSS tříd, které měly definice v externím stylopisu (Emotion Docs, The css Prop).

Z výše uvedeného pak vychází ještě další výhoda. Jelikož není potřeba používat CSS třídy, není ani potřeba třídy pojmenovávat a vývojář tak nemusí řešit problémy s namespaces.

Emotion a Styled components

Mezi nejpoužívanější CSS-in-JS frameworky dneška patří Emotion a Styled Components. Oba tyto frameworky, jak již bylo uvedeno výše, zapisují CSS vlastnosti přímo do JS souborů, čímž narušují dosavadní dogma externích stylopisů. Každý z těchto frameworků na to jde však trochu odlišným způsobem.

Styled Components prvně definují komponenty a to tak, že je prvně komponenta pojmenována, následně je určeno, jakým tagem se má komponenta vykreslovat, a nakonec jaké styly má tento tag přijímat. V kódu by pak Styled Components zápis vypadal asi nějak takto (Styled Components Docs, Getting Started):

```
const Title = styled.h1`
    font-size: 1,5em;
    text-align: center;
    color: palevioletred;
`;

const Wrapper = styled.section`
  padding: 4em;
  background: papayawhip;
`;

render(
  <Wrapper>
    <Title>
       Hello World!
     </Title>
  </Wrapper>
);
```

Emotion ke stejnému problému však přistupuje trochu jinak, stejný kód jako je výše, by šel v Emotion přepsat do této formy:

```
render(
   <section
      css{css`
         font-size: 1,5em;
         textalign: center;
         color: palevioletred;
      `}

   >
      <h1
         css={css`
            padding: 4em;
            background: papayawhip;
         `}
       >
          Hello World!
       </h1>
    </section>
);
```

Nyní přirozeně vyvstává otázka, který přístup k zápisu je lepší. Asi záleží na osobních preferencích. Pokud bych však měl mluvit za sebe, vybral bych si Emotion.

Emotion totiž proti Styled Components řeší 2 problémy mnohem lépe:

1) Lepší přehlednost. Okamžitě je totiž vidět, které styly se aplikují na jaký tag. Styled Components totiž trpí již jistou mírou abstrakce. V *„render()“* funkci jsou totiž vidět pouze komponenty vzniklé, v případě většího souboru, někde o X řádků výše a tak na první pohled není vidět, jaké styly a jaký tag daná komponenta představuje.

2) Není potřeba vymýšlet pro každou reprezentaci HTML tagu speciální název. Pro každou komponentu ve Styled Components se totiž musí vymýšlet speciální název, v Emotion stačí zapisovat pouze HTML tag a k němu příslušná CSS pravidla.

Kromě formy zápisu však Emotion proti Styled Components exceluje ještě v něčem jiném. Umí totiž generovat source mapy (Emotion Docs, Source Maps). V rozsáhlé aplikaci je tato možnost k nezaplacení. V prohlížečovém inspektoru stylů je pak díky source mapám totiž okamžitě jasné, z jakého souboru a jakého řádku tohoto souboru je inspektované pravidlo generováno.

3 PRAKTICKÁ ČÁST – 3D NÁVRHÁŘ NÁBYTKU

Nyní po formulaci teoretických východisek z pohledu současného vývoje front-endu by již nemělo být pochyb o tom, že příchod nových technologií přináší na pole web developmentu nové, a hlavně pokročilejší možnosti tvorby interaktivních aplikací. Proto bude těchto nových technologií využito v následující praktické části práce, a to při tvorbě aplikace „3D návrhář nábytku“.

3.1 PROČ 3D NÁVRHÁŘ NÁBYTKU?

Jakožto demonstrace využití nových front-end technologií by šla naprogramovat široká škála různých single-page aplikací. Inspirace by bylo konec konců dost, vždyť současný web se poslední dobou těmito aplikacemi začal zaplavovat. To je však paradoxně ten hlavní problém. Inspirace je sice dost, naneštěstí možná ale až příliš. Prakticky veškeré nápady na vývoj něčeho nového a zajímavého totiž dnes ztroskotávají na tom, že jsou již realizovány. V těchto případech pak nezbývá nic jiného než nanejvýš danou aplikaci lehce rozšířit či vylepšit. To však není tak zábavné a ani užitečné ve srovnání s vyvíjením aplikace, která na trhu ještě není.

„3D návrhář nábytku“ se nachází někde na pomezí těchto 2 kategorií. Podobné aplikace sice existují, nicméně všechny na tolik specifické, že ve formě, která bude navržena a naprogramována, nikde (přinejmenším na českém trhu) žádná neexistuje. Jeho existence by přitom mohla zlepšit průběh zadávání objednávek nejednomu truhláři, či truhlářské společnosti.

3.2 KDE SE VZAL NÁPAD?

Jako spousta užitečných nápadů přišel i tento nápad tzv. cestou „zezdola“, čili od potřeb koncového zákazníka, v tomto případě z potřeb mě samotného. Když jsem totiž v nedávné době potřeboval objednat skříňku na míru, narazil jsem na problém. Při prohledávání internetových stránek všemožných truhlářství a podobných subjektů jsem totiž nenašel jedinou webovou stránku, kde by šlo interaktivně nakonfigurovat obyčejnou skříňku. Její definice se přitom skládá pouze z několika málo parametrů. Existuje sice pár aplikací pro návrh nábytku, vždy však natolik specificky zaměřené, že je nelze použít pro objednávku tohoto typu.

Proces objednávky obyčejné skříňky je tak na českém trhu, jak bude konec konců ukázáno v následující kapitole „Průzkum trhu“, zredukován na vyplnění obyčejného kontaktního formuláře a to ještě „nejlépe“ doplněného o pole pro přiložení souboru s vlastním náčrtkem. Tato řešení jsou z hlediska UX v dnešní době již rozhodně nepřijatelná. O důvod navíc, proč se vývojem vlastního návrháře zabývat.

3.3 PRŮZKUM TRHU

Průzkum trhu bude prováděn za pomocí 2 nejvýznamnějších vyhledávačů českého internetu a sice Google.cz a Seznam.cz. Z posledního dostupného srovnání těchto 2 vyhledávačů za poslední kvartál 2018 bylo zjištěno, že na českém internetu se prohledává ze 75 % přes Google, 25% podíl trhu pak patří Seznamu (Kos, 2019).

Pro relevantní výsledky z průzkumu, je potřeba si zvolit správná klíčová slova. V tomto případě se jednoznačně pro objednávku skříně na míru nabízí fráze:

- „Skříň(e) na míru“
- „Nábytek na míru“
- „Skřín(ě) na zakázku“
- „Nábytek na zakázku“

3.3.1 Výsledky vyhledávání dle klíčových slov

Jako relevantní výsledky vyhledávání dle klíčových slov výše byly stanoveny první 3 strany výsledků z obou dvou vyhledávačů (Google, Seznam) očištěny o placenou reklamu čili o několik horních a dolních výsledků na každé straně označených jako „Reklama“.

Tabulky v příloze (Tabulka 7, Tabulka 8, Tabulka 9) zaznamenávají tyto výsledky vyhledávání z prosince 2019 odpovídající jednotlivým klíčovým slovům, a to od nejvýše umístěného po nejníže umístěný.

Pozn.: Některá nevyplněná pole na konci tabulek jsou způsobena těmito faktory.

- Vyřazení duplicitních výsledků, pokud se doména vyskytovala ve výsledcích vyhledávání vícekrát.

- Vyřazení nerelevantních výsledků vyhledávání, zejména výsledků z inzertních serverů (inzerce na prodej nábytku.

3.3.2 Redukce výsledků vyhledávání na relevantní data

Výsledky vyhledávání lze nyní použít pro potřeby průzkumu trhu. Analyzovat však manuálně stovky subjektů (302) by bylo neefektivní. Je tedy potřeba vyhledávací data nějakým způsobem zredukovat. Jelikož se ve výsledcích vyhledávání neustále opakují stejné subjekty na různé vyhledávací dotazy, je potřeba prvně data očistit o duplicity.

Manuální očištění o duplicity by bylo rovněž neefektivní, nabízí se tedy možnost převést data do databáze, aby s nimi šlo snáze pracovat pomocí SQL dotazů.

Jednotlivé domény lze tedy umístit do tabulky pojmenované např. „*searchResultsAddresses*“ obsahující sloupec s výsledky vyhledávání pojmenovaný např. „*webAddress*“.

Nad těmito daty již lze efektivně provádět SQL dotazy. Duplicity by šly nyní očistit za pomoci SQL příkazu „*DISTINCT*“. Ještě lepší je však zároveň získat informaci, kolikrát se daný výsledek vyhledávání opakuje. Tento výpis pak lze získat za pomocí kombinace příkazu „*SELECT*“ a funkce „*count()*“ následujícím SQL dotazem:

```
SELECT count(webAddress), webAddress
FROM searchResultsAddresses
GROUP BY webAddress
ORDER BY count(webAddress) DESC
```

Výsledek tohoto dotazu pak demonstruje následující tabulka (Tabulka 2):

12	jninterier.cz	2	truhlarstvi-balihar.cz
9	indeco.cz	2	truhlarstvi-zdara.cz
8	vestavenky.cz	2	siko.cz
8	a-typ.cz	1	truhlarstvi-bomi.cz
8	vestavene-skrine-ardekor.cz	1	feldman.cz
7	stako.cz	1	sedaci-soupravy-nabytek.cz
6	amonit.cz	1	truhlarstvi-futo.cz
6	sten-skrine.cz	1	creative-interior.cz
6	skrine-komandor.cz	1	truhlarstvimencak.cz
6	lj-kuchyne.cz	1	jirecek.cz
6	levne-skrine.cz	1	top-interiery.eu
6	nabytek-max.cz	1	qlinterier.cz

5	skrine-mladaboleslav.cz	1	truhlarstvi-matek.cz
5	pjatak.cz	1	variowall.cz
5	skrinenamiru.cz	1	klasikcz.eu
4	halbos.cz	1	simlinterier.cz
4	kuchyne-brno.eu	1	interiery-skrine.cz
4	design-nadto.cz	1	truhlarstvi.name
4	skrinekatrin.cz	1	akvamex.cz
4	skrin.cz	1	holik.cz
4	truhlarstvi-marek.cz	1	lermoplus.cz
4	hanak-nabytek.cz	1	hanak-skrine.cz
4	truhlarstvimicek.cz	1	kprostor.cz
3	skrine.cz	1	laminonabyteknamiru.cz
3	truhlarstvi-marecek.cz	1	bpparket.cz
3	gilikdesign.cz	1	aktivpisek.cz
3	cizek-skrine.cz	1	nabytekpiza.cz
3	nabytekriha.cz	1	kuchyne-fokos.cz
3	nabytek-matefi.cz	1	navrhisivestavku.cz
3	dverecag.cz	1	sapelky.cz
3	andalo-skrine.cz	1	pjatek.cz
3	ujirika.cz	1	prostor.cz
3	vestavene-skrine-petricek.cz	1	eurourban.cz
3	indoor.cz	1	truhlarstvibomi.cz
3	nabytek-na-miru-brno.cz	1	belterra.cz
3	nabyteknamiru.cz	1	migala.cz
3	astr.cz	1	atyp-nabytek.cz
2	jamall.cz	1	bdorg.cz
2	maryskonabyteknamiru.cz	1	nabytekkarel.cz
2	bevedo.cz	1	intery-skrine.cz
2	navrhnisivestavku.cz	1	inspirointeriery.cz
2	salu.cz	1	truhlarstvi-borohradek.cz
2	nabytek-na-zakazku.com	1	nabyteknamiru-hk.cz
2	kmtruhlarstvi.cz	1	art-style.cz
2	simek-interier.cz	1	nabytek-na-miru.eu
2	ciganik.cz	1	zakazkovy-interier.cz
2	dreamwood.cz	1	truhlarstvi-janzita.cz
2	hezkydomov.cz	1	sipkadesign.cz
2	mbyt.cz	1	nadop.cz
2	truhlarstviex.cz	1	nazakazku.eu
2	simcak.net	1	skrine-liberec.cz
2	nabytek-trend.cz	1	zlutahala.cz

2	novak-nabytek.cz	1	interierstudio.eu
2	truhlarstvi-luxra.cz	1	nabytek-lupinovka.cz
2	nabytekvyroba.cz	1	zakazkoveinteriery.cz
2	kapitan.cz	1	nabytekkinta.cz
2	nabytek-dan.cz	1	apokork.cz
2	heth.cz	1	adam-nabytek.cz
2	tninterier.cz	1	nabytek-kratochvil.cz
2	rrgroup.cz	1	triant.cz
2	ikea.com	1	pelckuchyne.cz
2	moebelix.cz	1	favi.cz
2	stylvenkova.cz	1	nabytekdelfi.cz
2	vestavene-skrine-max.cz	1	vyroba-nabytku-kudrna.cz
2	hezkakuchyn.cz	1	frosch.cz

Tabulka 2 - Počet opakování jednotlivých domén mezi výsledky vyhledávání v různých prohlížečích na různá klíčová slova (Google, Seznam)

Bohužel i s touto redukcí zůstává k průzkumu **130** subjektů, což je stále vysoké číslo.

```
SELECT count(DISTINCT webAddress)
FROM webAddresses

130
```

Dále lze tedy tato data zredukovat pouze na významné domény, tedy ty domény, které se ve výsledcích vyhledávání objevily, např. nejméně 3x.

Tímto způsobem zůstane **37** nejsilnějších domén čili subjektů, kteří nejvíce odpovídají na relevantní výsledky vyhledávání na českém internetu. Těchto 37 domén je již dostatečně rozumný počet pro manuální analýzu a budou proto použity k samotnému průzkumu trhu. V tabulce výše (Tabulka 2) je tento finální výběr domén pro průzkum trhu označen zelenou barvou.

3.3.3 Průzkum nejvýznamnějších prodejců nábytku na míru

Webové stránky všech 37 subjektů zvolených jako soubor dat pro průzkum byly jedna po druhé podrobně zkoumány. Hlavním cílem tohoto průzkumu bylo identifikovat na webové stránce subjektu způsob prodeje skříní na míru. Způsob prodeje byl následně rozčleněn do 3 kategorií:

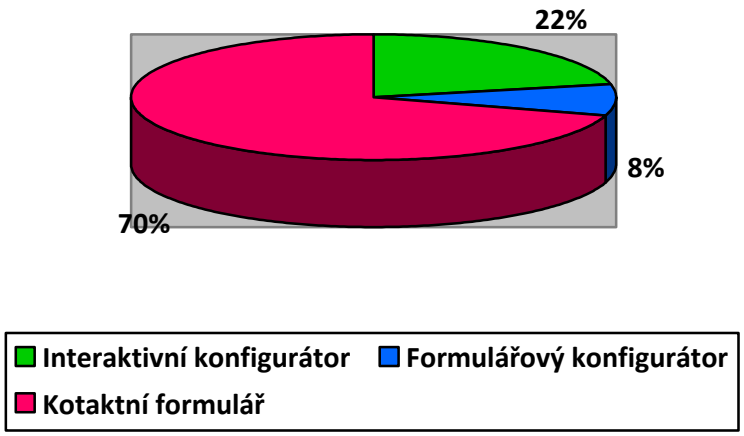
1) Existence interaktivního konfigurátoru nábytku
2) Jednoduchý konfigurátor pomocí formulářových polí
3) Objednávka pouze pomocí kontaktního formuláře

Výsledky průzkumu zachycuje následující Tabulka 3 a Graf 2.

Interaktivní konfigurátor	8
Indeco.cz, Stako.cz, Amonit.cz, Skrine-komandor.cz, Levne-skrine.cz, Sten-skrine.cz, Pjatak.cz, Nabyteknamiru.cz	
Formulářový konfigurátor	3
Vestavene-skrine-ardekor.cz, Skrin.cz, Skrine.cz	
Kontaktní formulář	26
Ostatní	

Tabulka 3 - Zastoupení jednotlivých typů prodejů mezi zkoumanými subjekty

Z procentního zastoupení jednotlivých typů prodejů mezi zkoumanými subjekty bylo dojito k naprosto jasnému závěru, že většina prodejců (70 %) zpracovává objednávky pouze na základě vyplnění kontaktního formuláře, 8 % prodejců používá alespoň nějaký zjednodušený typ formulářového konfigurátoru a pouze 22 % prodejců pak využívá nějakého sofistikovanějšího interaktivnějšího konfigurátoru.



Graf 2 - Procentní zastoupení jednotlivých typů prodejů mezi zkoumanými subjekty

3.3.4 Průzkum technických řešení interaktivních konfigurátorů

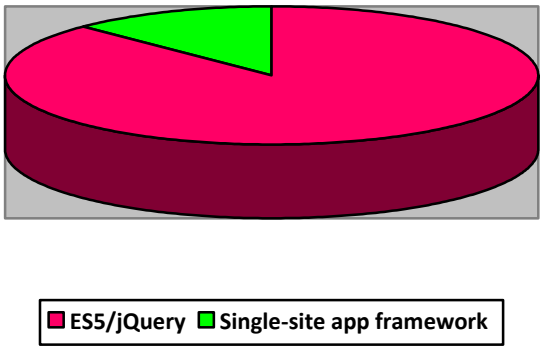
Nyní po separaci subjektů s interaktivními konfigurátory konečně lze prověřit premisu této diplomové práce a položit si tak znovu otázku „Pokud existují nějaké interaktivní návrháře nábytku, jak zastaralé je jejich technické zpracování?“.

Pro zjištění odpovědi na tuto otázku byly jednotlivé návrháře podrobeny průzkumu použitých technologií, výsledkem čehož bylo dojito k závěru, jenž shrnuje následující tabulka (Tabulka 4).

ES5/jQuery	7
Indeco.cz, Stako.cz, Amonit.cz, Skrine-komandor.cz, Levne-skrine.cz, Sten-skrine.cz, Pjatak.cz	
Single-page app framework	1
Nabyteknamiru.cz	

Tabulka 4 - Zastoupení použitých technologií mezi jednotlivými konfigurátory

Z těchto údajů jasně vyplývá, že téměř všechny analyzované konfigurátory fungují ještě na starých technologiích jQuery/ES5. Při celé analýze byla nalezena pouze jediná aplikace, která využívá moderního single-page application frameworku, v tomto případě se jednalo o AngularJS u prodejce nábytku Nabyteknamiru.cz.



Graf 3 - Procentní zastoupení použitých technologií mezi jednotlivými konfigurátory

Zajímavostí také je, že u některých subjektů, které se však již nedostaly do relevantního testovacího souboru, byly nalezeny dokonce konfigurátory vytvořené v dnes již prehistorickém Flashi. Jedním z těchto subjektů byl dokonce jeden z největších prodejců nábytku celosvětově a sice švédská Ikea (IKEA, PAX).

3.3.5 Průzkum funkcionality interaktivních návrhářů

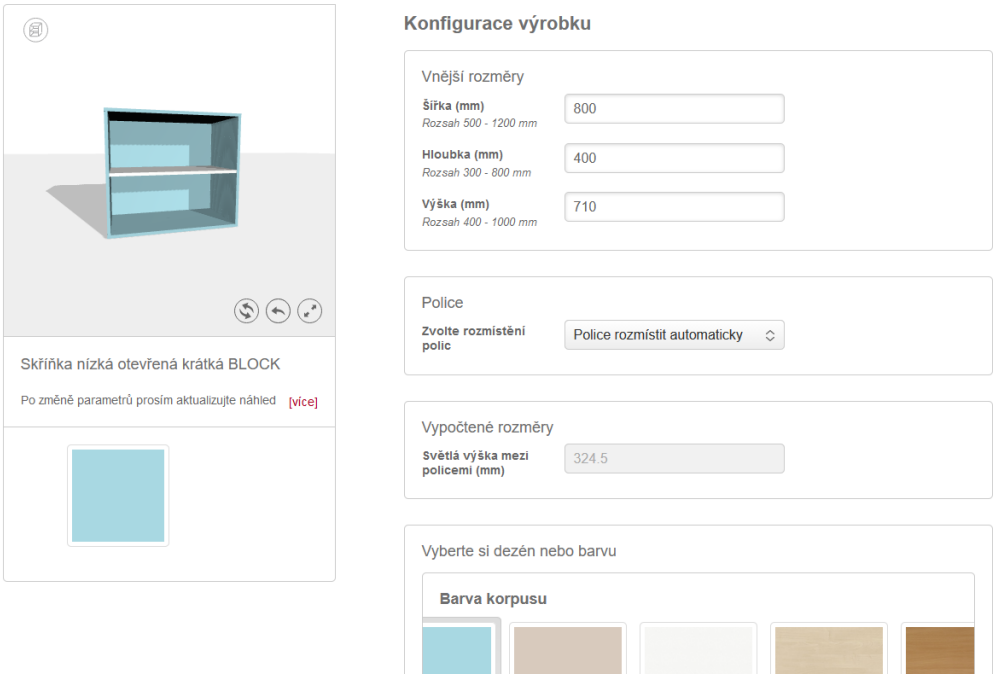
Nyní by již nemělo být pochyb o tom, že z hlediska použitých technologií je na poli konfigurátorů nábytků rozhodně prostor pro zlepšení. Ale jak je to s jejich funkcionalitou? Co jednotlivé konfigurátory umí? Na jaké typy objednávek jsou zaměřené? Jaké mají nedostatky? Přesně na tyto otázky bude odpovězeno v následujícím rozboru.

Ať už to vypadá jakkoliv nepravděpodobně, většina testovaných konfigurátorů se totiž specializuje na jeden specifický typ objednávek a sice na návrh vestavěné skříně, konkrétně 7 z 8 analyzovaných konfigurátorů spadá do této kategorie „*Návrhář vestavěných skříní*“. Ještě větší náhodou pak je, že všechny tyto návrháře pochází z kategorie *„jQuery/ES5*“ návrhářů. Zbylý konfigurátor nabyteknamiru.cz by se pak dal zařadit do kategorie „*Konfigurátor předdefinovaných typů nábytku“*.

Konfigurátor předdefinovaných typů nábytku (Angular návrhář)

Tento konfigurátor dopadl jak z hlediska funkčního tak z hlediska technologického nejlépe ze všech řešení.

Technologické řešení tohoto produktu využívá moderního single application frameworku Angular v součinnosti s rozhraním pro webový render návrhů nábytku z návrhářského softwaru společnosti Imos3D (www.imos3d.com). Toto řešení tak umožňuje konfigurovat prostřednictvím webového formuláře všechny firmou již namodelované produkty podobným způsobem jako by se dělo v desktopové aplikaci. Přesto má toto řešení pár nedostatků. Při změně většiny parametrů nábytku (snad kromě barvy) se totiž musí uživatel přesunout z formuláře pro definici nábytku (vpravo) zpět do náhledové části (vlevo) a zde ručně kliknout na tlačítko pro aktualizaci modelu, a to je z hlediska UX velice špatné řešení. Pokud má návrhář využívat všech možností single-page application frameworku, měl by se stav aplikace nadefinovaný ve formulářích okamžitě promítat do stavu aplikace v náhledu skříně. To se zde bohužel neděje, a dokonce i po kliknutí na aktualizaci modelu skříně se změna neprovede automaticky, ale uživatel musí čekat průměrně kolem 3 sekund, než se dotočí načítací ikona a zobrazí nová skříň. Dalším problémem pak je, že otáčení skříně ve 3D prostoru se rozhodně nedá považovat za plynulé. Vše ohledně plynulost bude více probráno v závěrečné kapitole „4. 6 Kvantifikace výsledků práce“.



Obrázek 1 - Návrhář nábytku nabyteknamiru.cz (https://www.nabyteknamiru.cz/kancelarske-skrine/192/skrinka-nizka-otevrena-kratka-block?c=33)

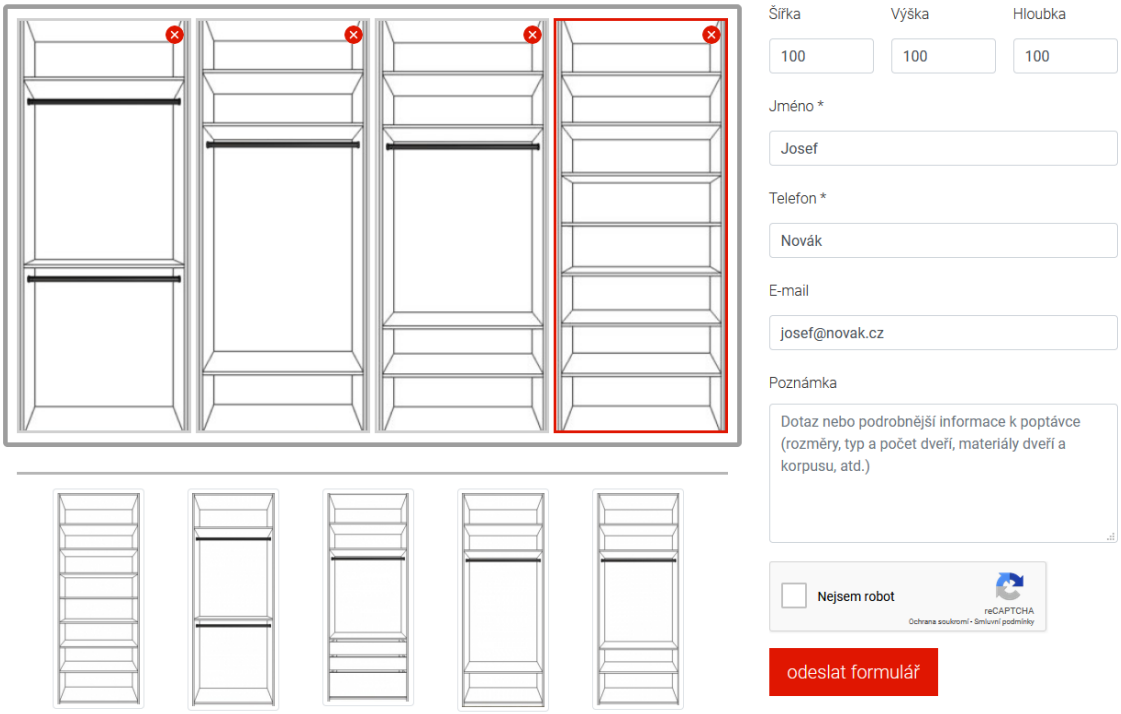
Návrháře vestavěných skříní

Těchto 7 návrhářů lze rozdělit na 2 podskupiny podle jejich technické kvality.

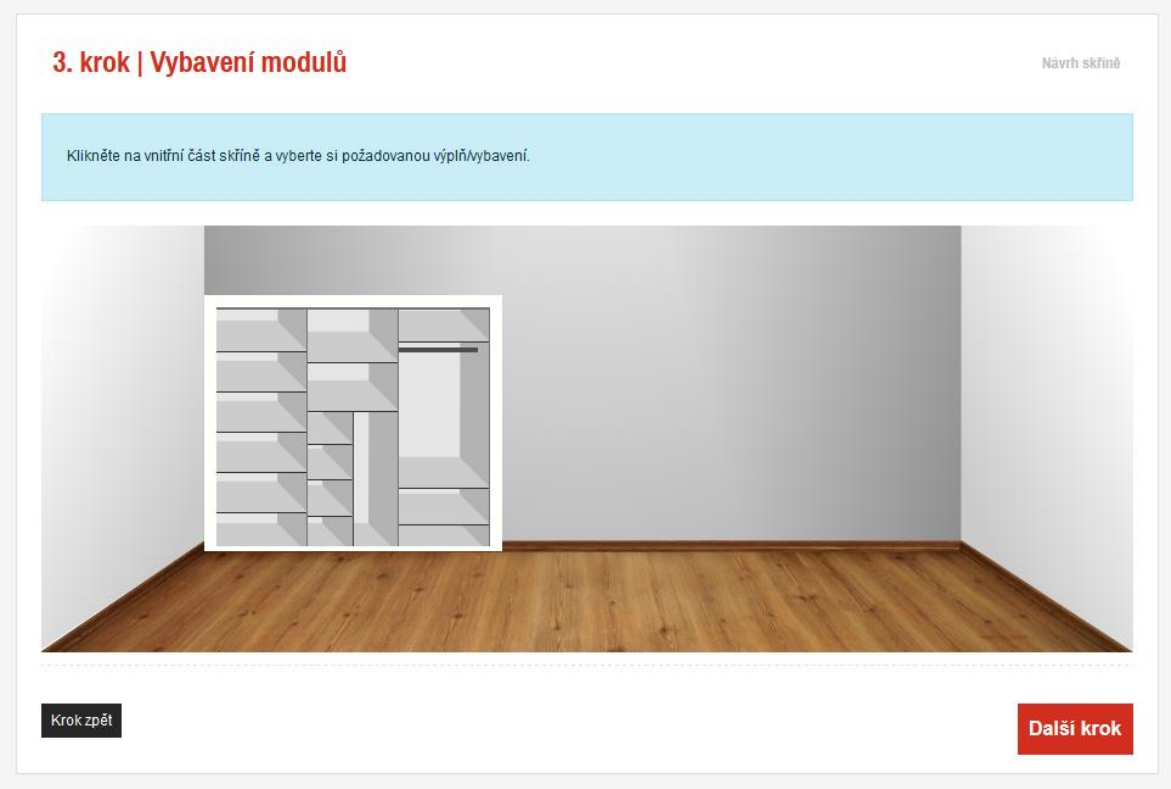
Primitivní obrázkové návrháře	Levne-skrine.cz, Sten-skrine.cz, Pjatak.cz
Sofistikovanější Canvas návrháře	Indeco.cz, Stako.cz, Amonit.cz, Skrine-komandor.cz

Tabulka 5 - Návrháře vestavěných skříní dle technického řešení

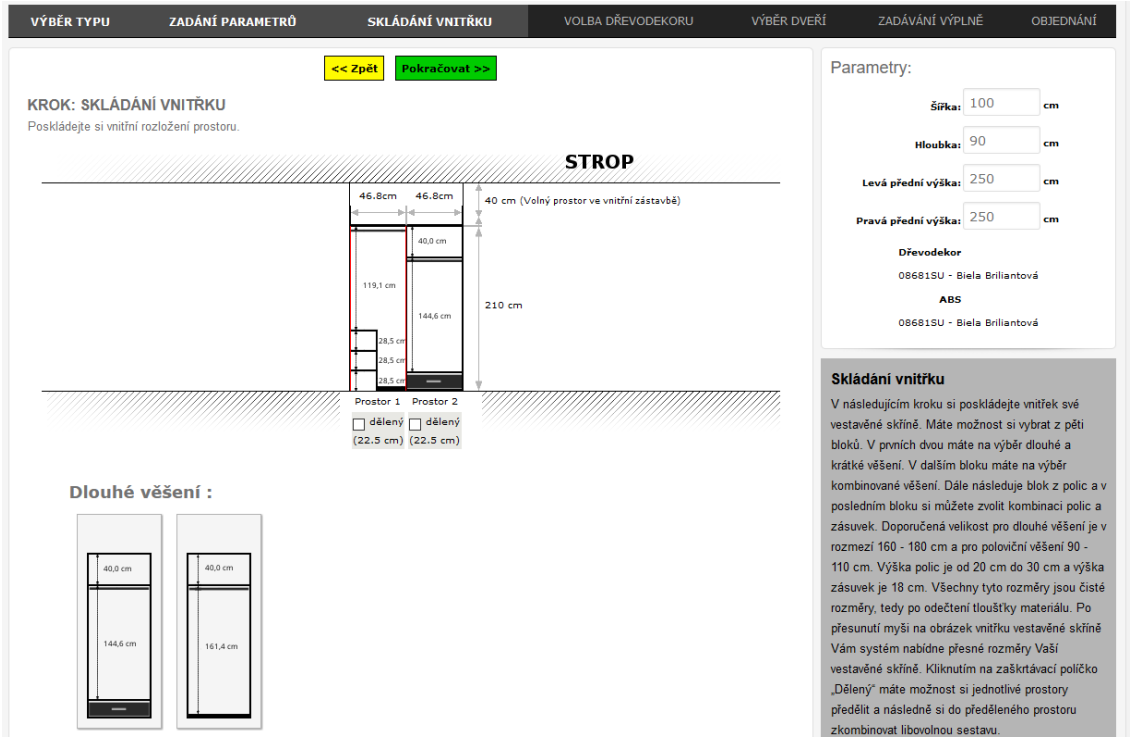
První skupina primitivních návrhářů je z technologického hlediska velice zastaralá, veškerá vizualizace skříně zde probíhá za pomoci skládání předdefinovaných obrázků jednotlivých prvků skříně. Náhled skříně tedy vůbec nereaguje na změnu rozměrů skříně, o barvě skříně ani nemluvě. Tyto návrháře tak lze považovat prakticky pouze za lehce vylepšené statické formuláře (Obrázek 2, Obrázek 3, Obrázek 4).



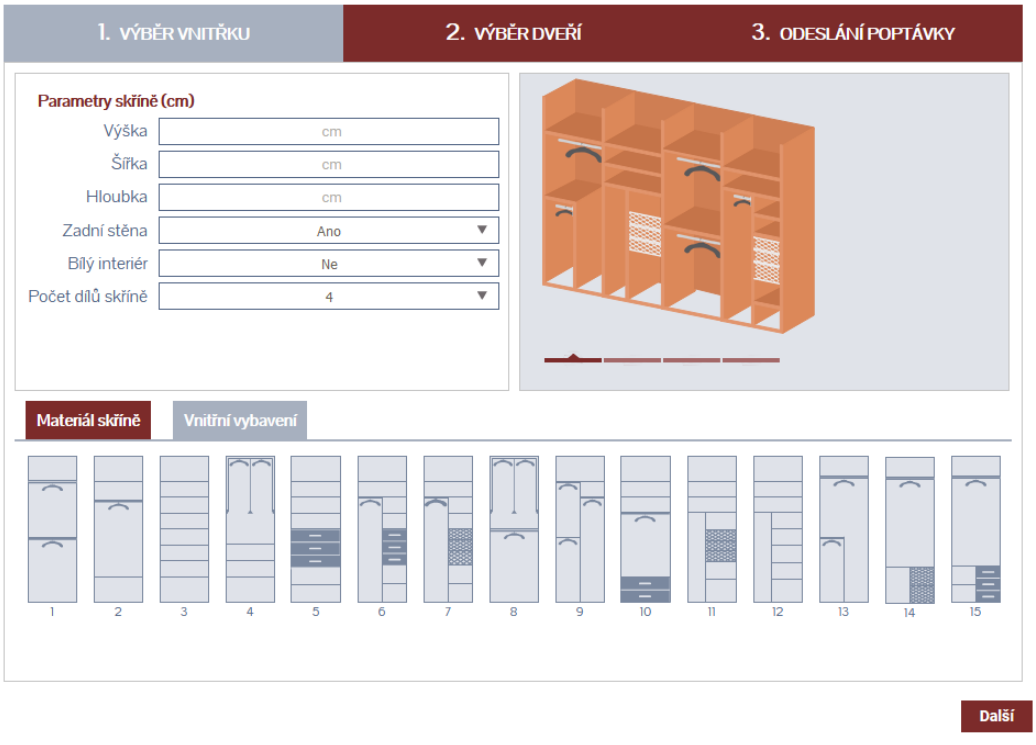
Obrázek 2 - Návrhář sten-skrine.cz (https://sten-skrine.cz/3d-navrhar)



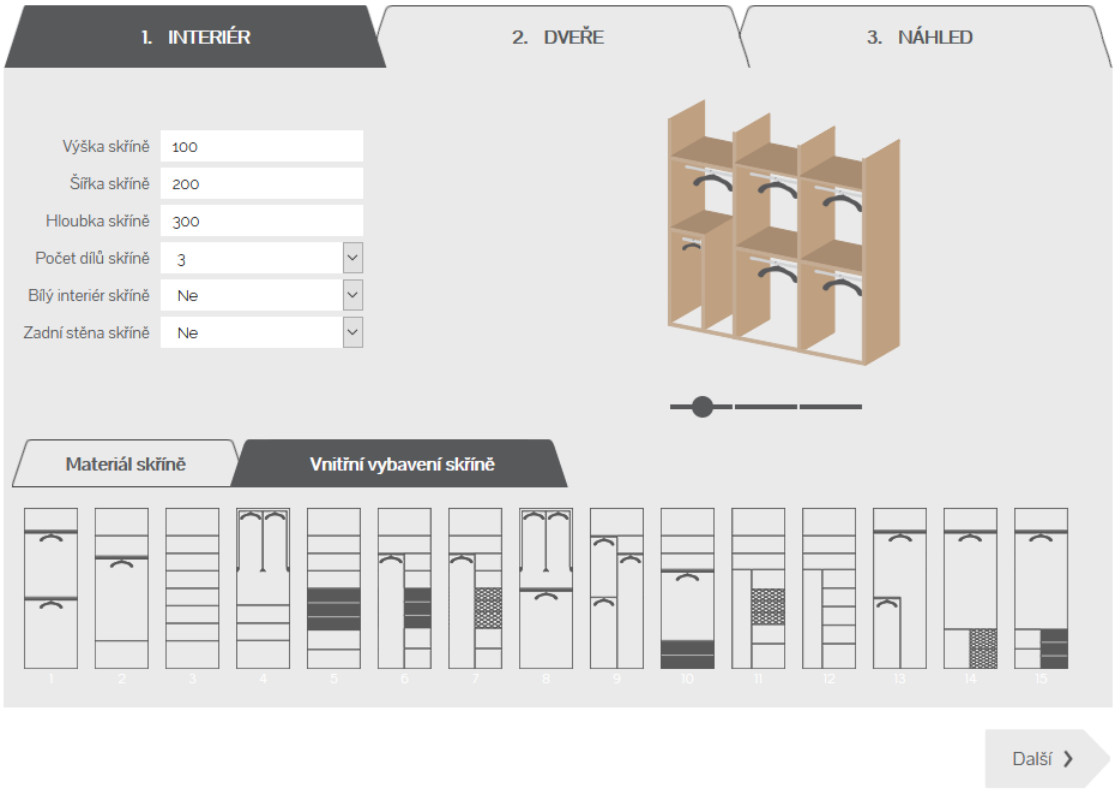
Obrázek 3 - Návrhář pjatak.cz (http://www.pjatak.cz/vestavene-skrine-na-miru/online-navrhar-skrine/)



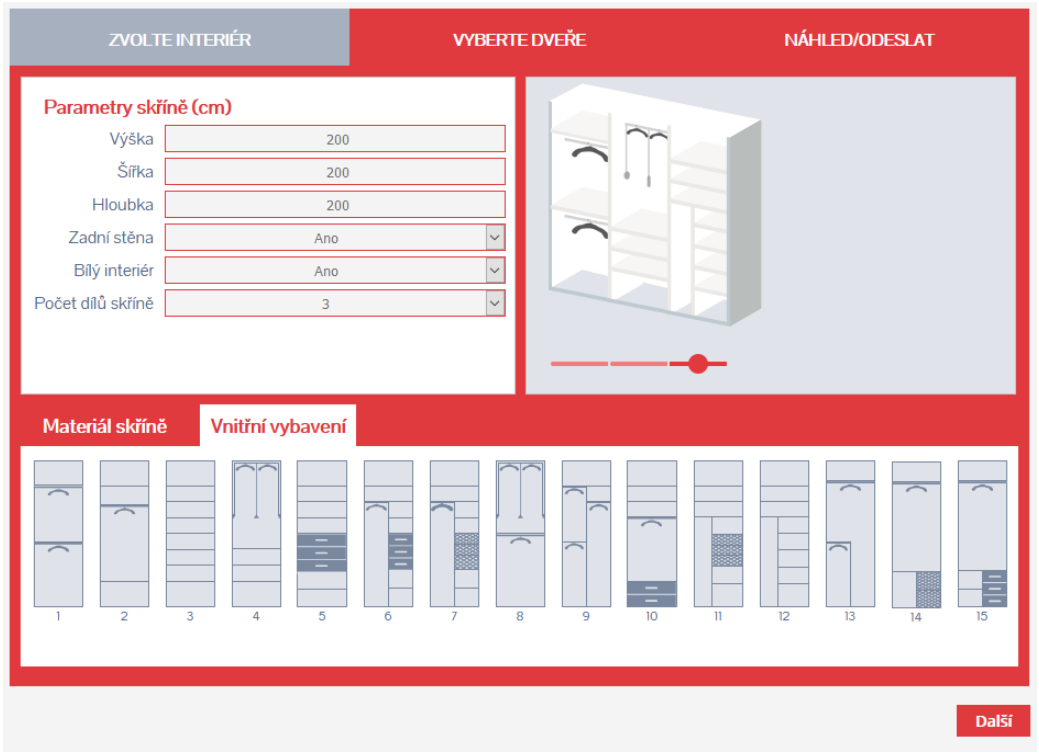
Obrázek 4 - Návrhář levne-skrine.cz (http://rozpocet.levne-skrine.cz/)



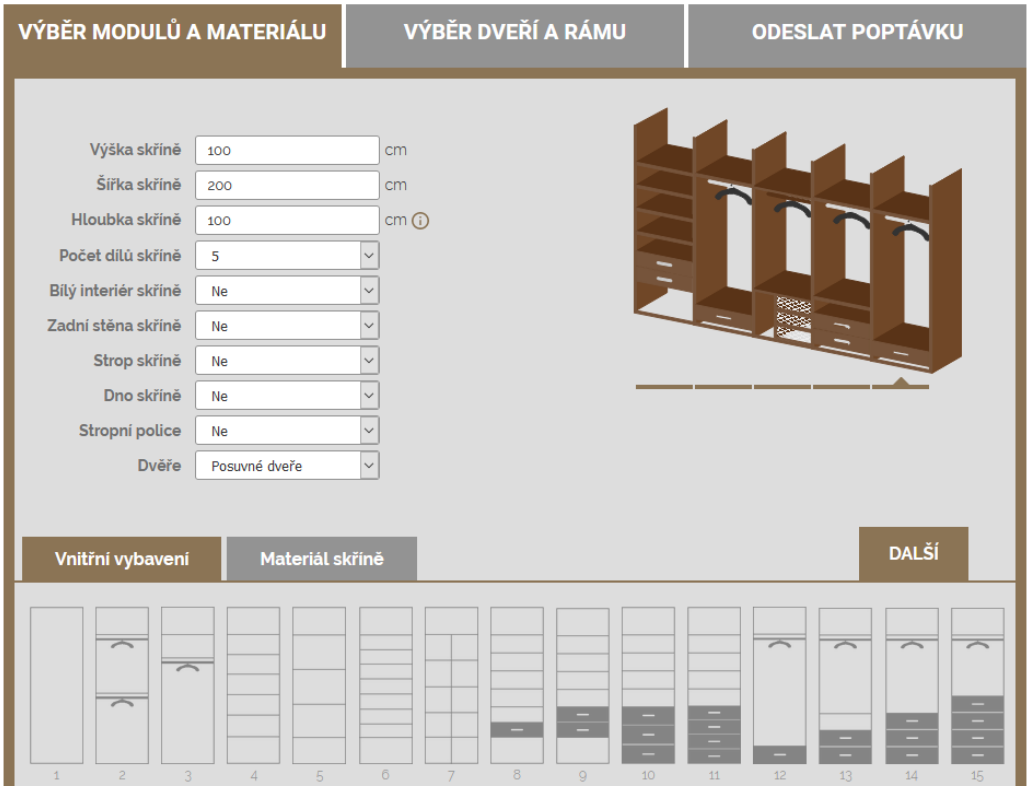
Obrázek 5 – Návrhář stako.cz (https://www.stako.cz/poptavka/3d-navrhar/)



Obrázek 6 - Návrhář indeco.cz (https://www.indeco.cz/navrhar-vestavenych-skrini/)



Obrázek 7 - Návrhář skrine-komandor.cz (https://www.skrine-komandor.cz/3d-navrhar/)



Obrázek 8 - Návrhář amonit.cz (https://www.amonit.cz/poptavky/navrhar-vestavenych-skrini/)

Zajímavější, přinejmenším z technického hlediska, je pak druhá skupina návrhářů. Už na první pohled je vidět, že tyto návrháře mají stejný základ. Pravděpodobně se jedná o produkt jedné společnosti, která návrhář v lehce diverzifikovaných formách nasadila několika subjektům naráz (Obrázek 5, Obrázek 6, Obrázek 7, Obrázek 8).

Tyto návrháře využívají z technologického hlediska vykreslování modelů skříní na HTML5 Canvas. To otevírá široké možnosti, jako např. obarvování modelů skříní různými barvami. Bohužel zde však již, jako u všech testovaných návrhářů, model skříně absolutně nereaguje na změnu velikosti, a tak při různě nastavených šířkách, hloubkách a výškách skříň zachovává pouze jeden a ten stejný poměr stran, což model výsledné skříně dosti znehodnocuje.

Modely skříní jsou pak takovým pseudo-3D. Technologie Canvasu jako takového totiž neumožňuje modelovat ve 3D, a tak se zde skříně zobrazují v pevném úhlu z boku, který vytváří dojem 3D, avšak jakékoliv otáčení v prostoru zde nepřipadá v úvahu.

I přestože tento návrhář nedisponuje technologickou dokonalostí, tak velikou devizou mu je hlavně implementace velkého množství předdesignovaných modelů skříní, ze kterých lze výslednou skříň nakonfigurovat.

3.4 TVORBA 3D NÁVRHÁŘE SKŘÍNĚ

Na základě výše uvedené analýzy trhu lze tedy nyní přistoupit k naprogramování technologicky lepšího řešení 3D návrháře skříně, které si bude klást za cíl hlavně odstranění nedostatků současných řešení.

3.4.1 Shrnutí nedostatků současných řešení

Následující tabulka (Tabulka 6) zachycuje nedostatky zkoumaných řešení:

	Primitivní obrázkové návrháře	**Sofistikovanější Canvas návrháře**	**Angular návrhář**
Dynamické propisování parametrů skříně (AJAX like, bez step-by-step reloadů)			
Propracovanější náhled skříně než obrázek			
Propracovanější náhled skříně než Canvas			
Promítání změn do modelu bez latence			
Otáčení, přibližování modelu bez latence			
Náhled skříně reflektující změnu barvy skříně			
Náhled skříně reflektující změnu rozměrů skříně			
Použitá technologie lepší než základní JS			
Použitá technologie SPA framework			
Vysoká variabilita typů skříní			

Tabulka 6 - Nedostatky současných řešení (zeleně – není nedostatek, červeně – je nedostatek, modře – nerelevantní parametr)

3.4.2 Definování cílů nového řešení

Nové řešení si klade za cíl vytvoření návrháře skříně bez nedostatků stávajících řešení. Tzn. vytvoření aplikace s následujícími charakteristikami:

a) využití SPA frameworku React.js
b) využití 3D web rendering technologie WebGL pro vykreslování 3D modelu skříně
c) dynamické aplikování všech definovaných vlastností skříně do 3D náhledu skříně bez zaznamenání latence
d) možnost otáčet kamerou a volně se pohybovat ve 3D zobrazení modelu skříně
e) možnost definice barvy skříně
f) možnost definice uspořádání polic skříně
g) možnost změny rozměrů skříně

3.4.3 Popis řešení

Jako základ aplikace byl zvolen balíček create-react-app (Facebook). Tento balíček, který na pozadí využívá technologií jako Babel nebo Webpack, vytváří front-endovou šablonu aplikace.

Aby aplikace fungovala nejenom na front-endu, ale také na back-endu, čili aby byla schopna ukládat informace i do databáze, byla tato front-endová šablona rozšířena na tzv. MERN stack (Jasraj), čili kombinaci následujících technologií: databáze **M**ongoDB, serveru **E**xpress, již použitého front-end frameworku **R**eactu a serverového jazyka **N**ode.js.

Pro hostování databáze bylo využito služby MongoDB Atlas (https://www.mongodb.com), pro verzování kódu služby GitLab (https://gitlab.com) a pro buildování aplikace služby Heroku.com (https://www.heroku.com/).

Pro účely CSS stylování byl využit balíček Emotion.js (https://emotion.sh).

Na vytvoření samotného 3D modelu skříně byla zvolena technologie WebGL, respektive knihovna Three.js (https://threejs.org/), respektive React wrapper knihovna React Three Fiber (https://github.com/react-spring/react-three-fiber).

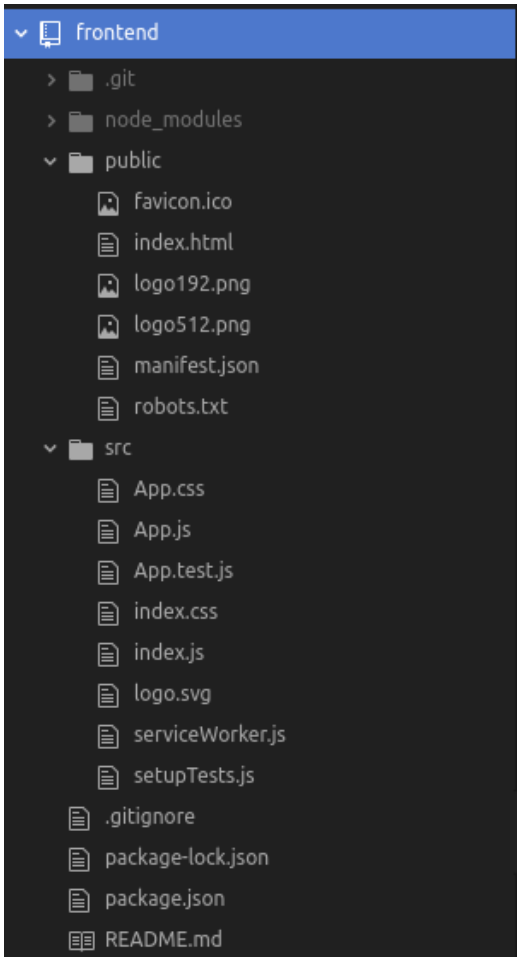
3.4.4 Postup řešení

Create-react-app

Jako základ aplikace byl zvolený npm balíček od vývojářů z Facebooku s názvem „create-react-app“ (https://create-react-app.dev/). Tento balíček lze nainstalovat např. pomocí spouštěče npm balíčků npx:

```
npx create-react-app navrhar/frontend --use-npm
```

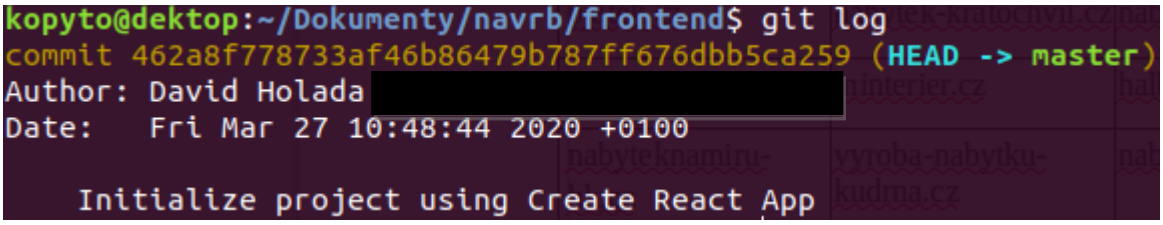
Instalací tohoto balíčku je vytvořen celý frontendový základ Reactové aplikace, tzn. je vytvořena základní souborová struktura aplikace, ale hlavně všechny nástroje, které se starají o build a které by bylo velice časově náročné nastavovat samostatně. Mezi tyto nástroje patří hlavně Webpack, Babel, ESLint, Prettier apod. Výslednou strukturu aplikace zachycuje následující obrázek:



Obrázek 9 - Souborová struktura aplikace (vlastní zpracování)

Git

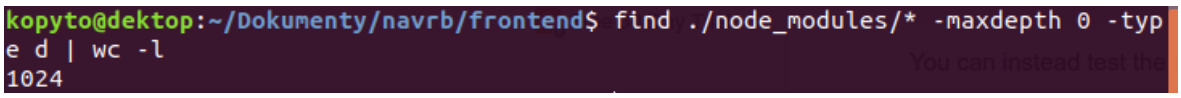
Z této struktury lze odvodit několik dalších věcí, které balíček *„create-react-app“* provedl. Např. hned z názvu první složky *„.git“* je vidět, že balíček rovnou inicializoval projekt jako git repositář, což je jedno z dalších drobných ulehčení práce, které tento balíček poskytuje. Pomocí příkazu *„git log“* pak jde rovnou zjistit, že instalací tohoto balíčku byla provedena nejenom inicializace repositáře, ale také, že byl rovnou proveden první inicializační commit celé aplikace.



Obrázek 10 - Inicializační commit aplikace (vlastní zpracování)

Node modules

V další části struktury lze vidět složku *„/node_modules“* se všemi nainstalovanými balíčky, které create-react-app potřebuje pro svůj běh. Jak je vidět z následujícího výpisu příkazu *„find“* pro spočtení všech balíčků (podsložek s hloubkou 0), tak balíčků je dohromady 1024.



Obrázek 11 - Výpis příkazu „find“ pro sečtení počtu node balíčků (vlastní zpracování)

To dává hrubou představu o tom, kolik konfigurační práce instalace tohoto balíčku usnadňuje.

Public folder

Další částí struktury je pak *„/public“* složka. Ta obsahuje pár základních souborů jako *„index.html“*, *„favicon.ico“* či *„robots.txt“*, pro další vývoj je však tato složka, až na několik výjimek nepoužívaná (Llobera, 2020).

Složka „src“

Nejdůležitější složkou celé frontendové části je ale rozhodně složka *„/src“,* kde se nachází drtivá většina celého kódu aplikace. To se nemusí sice na první pohled zdát, pravdou však je, že všechny nové části aplikace jsou přidávány právě do této složky.

Výčet souborů create-react-app pak uzavírají soubory „*.gitignore“*, *„package.json“*, *„package-lock.json“* a *„READ.me“*.

Soubor .gitignore

Soubor „*.gitignore“* standardně vylučuje některé adresáře, které nemá cenu verzovat, typicky se jedná o složku *„/node_modules“* či složku vytvořenou po vybuildování aplikace „/build“.

Soubor package.json

Zajímavější je však soubor „*package*.json“, který obsahuje např. všechny základní závislosti aplikace tzv. *„dependencies“*, jedná se však pouze o tzv. high-lvl závislosti. Pokud by se procházel kód těchto high-lvl závislostí, zjistili bychom, že těchto (v tomto případě 6) základních závislostí opravdu využívá všech 1024 npm balíčků zmiňovaných v kapitole *„4. 4.4. 3. Node modules“*. Dále zde existuje objekt *„scripts“,* který definuje základní skripty, kterými lze např. spouštět aplikaci *(„react-scripts start“)* či ji buildovat *(„react-scripts build“).* Dále se zde nachází ještě objekt *„eslintConfig“*, který nastavuje základní pravidla pro lintování kódu, čili pro kontrolu dodržování různých pravidel zápisu JavaScriptu. To je velice důležité hlavně pokud spolu pracuje více vývojářů a každý používá jiné konvence zápisu (např. jednoduché vs dvojité uvozovky). ESLint (https://eslint.org/) je schopen tyto změny detekovat a pokud nejsou změny dodržovány, tak nahlašovat errory, případně je i automaticky opravovat. Nejlépe pak pracuje tento nástroj ve spolupráci s nějakým eslint pluginem do IDE, který dokáže chyby ESLint z konzole vypisovat taktéž přímo do vývojového prostředí a chyby po uložení souboru případně automaticky upravovat. Např. do vývojového prostředí *„Atom“*, se tento balíček jmenuje *„linter-eslint“* (https://atom.io/packages/linter-eslint).

Posledním objektem v *„package.json“* je pak *„browsersList“*. Díky informacím v něm se přizpůsobují *„react-scripts“* tak, aby jejich výstupy podporovali určitou podmnožinu webových prohlížečů.

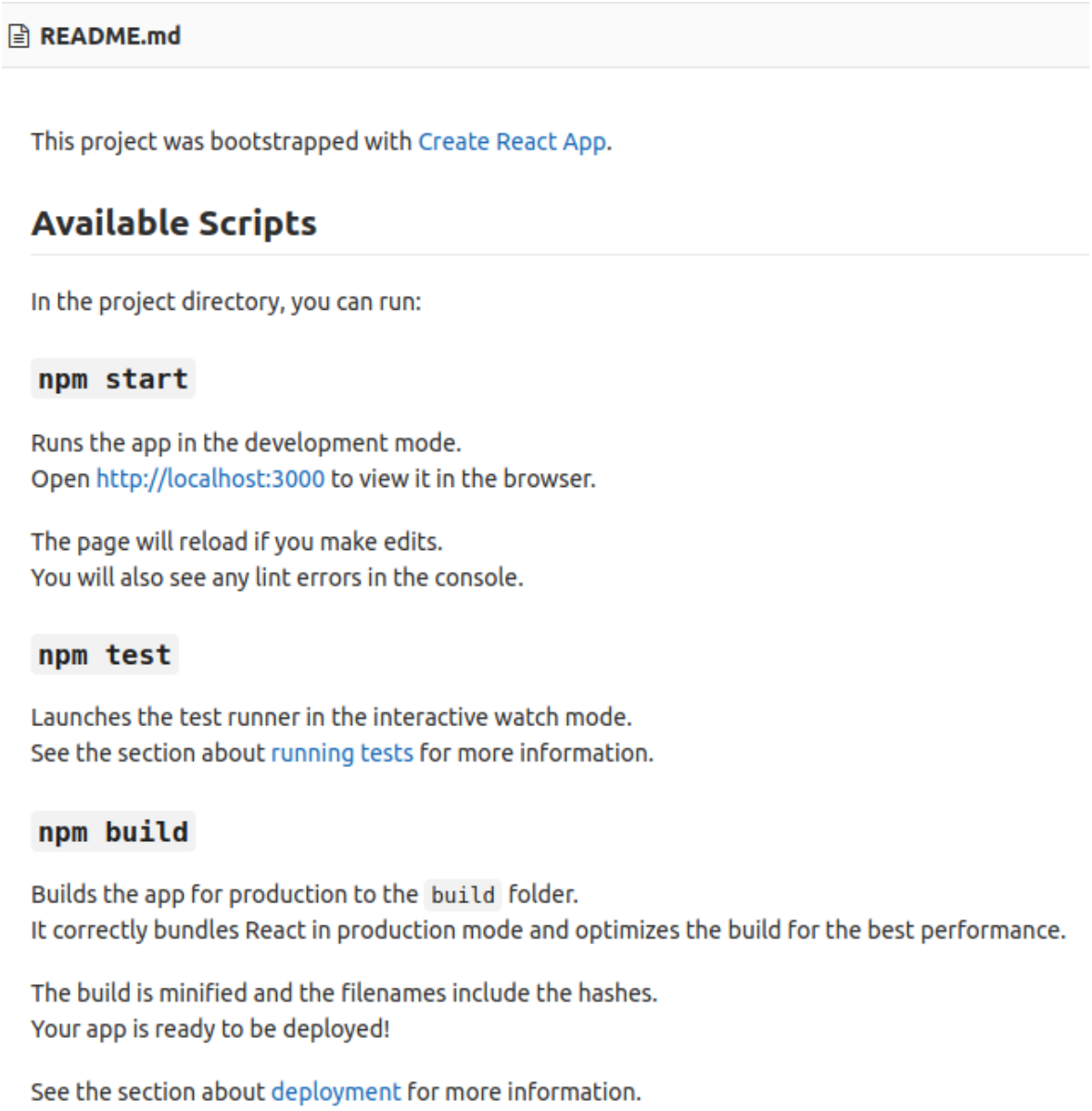
```
{
  "name": "frontend",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^4.2.4",
    "@testing-library/react": "^9.5.0",
    "@testing-library/user-event": "^7.2.1",
    "react": "^16.13.1",
    "react-dom": "^16.13.1",
    "react-scripts": "3.4.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": "react-app"
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

Obrázek 12 - Obsah souboru package.json (vlastní zpracování)

README.md

Posledním souborem celé aplikace je pak README.md, které slouží k popisu aplikace. V případě *„create-react-app“* popisuje základní skripty a odkaz na dokumentaci.

Zobrazení tohoto souboru na GitLabu zachycuje obrázek níže (Obrázek 13).



Obrázek 13 - Reprezentace souboru README.md na GitLabu (vlastní zpracování)

GitLab

Dalších krokem v tvorbě aplikace je využití plného potenciálu předpřipraveného, zatím pouze lokálního Git repositáře. Aby byl tento plný potenciál využit je nutné pro aplikaci také zvolit nějaké vhodné místo pro hostování vzdáleného repositáře. V případě 3D návrháře jsem se rozhodl pro službu GitLab.

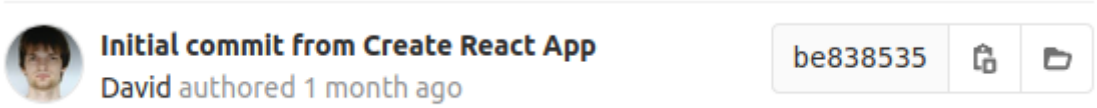
Po založení účtu na GitLabu a základním nastavení Gitu (Git Documentation, Getting Started – First-Time Git Setup) lze jednoduše pomocí několika příkazů provést *„push“* lokálního repositáře do repositáře vzdáleného (Obrázek 14).

Push an existing Git repository

```
cd existing_repo
git remote rename origin old-origin
git remote add origin git@gitlab.com:Holada/test313.git
git push -u origin --all
git push -u origin --tags
```

Obrázek 14 - Push existujícího Git repositáře (vlastní zpracování)

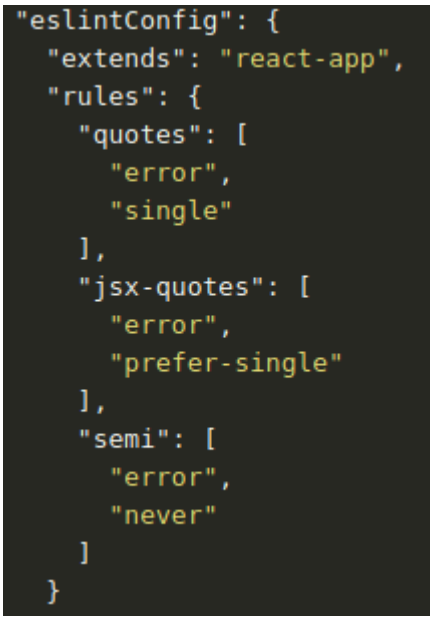
Tento krok umožní hlavně práci několika vývojářů najednou, kteří si mohou vzájemně posílat kód ze svých lokálních repositářů do repositářů vzdálených a vyměňovat si tak svou práci. Kromě toho však GitLab nabízí robustní UI pro správu repositáře mnohočetně přesahující možnosti obyčejného zobrazení výstupů git příkazů v konzoli. Už jen např. zobrazení commitu (Obrázek 15) zobrazuje GitLab mnohem více uživatelsky přívětivým způsobem, navíc bez uživatelovi nutnosti znát gitovské příkazy, což se může hodit hlavně nevývojářům, kteří chtějí mít přehled o postupu prací na daném projektu.



Obrázek 15 - Zobrazení commitu ve službe GitLab (vlastní zpracování)

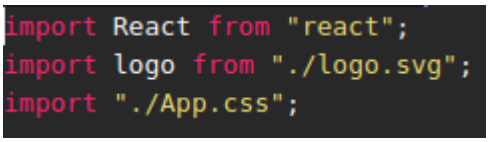
Úprava ESLint configu

Jak již bylo zmíněno výše, defaultní ESLint config v souboru *„package.json“* lze modifikovat dle vlastních potřeb. Proto jednou z prvních úprav na aplikaci 3D návrháře byla editace toho configu (Obrázek 16) tak, aby šlo používat jednoduché uvozovky místo dvojitých a dále, aby nebylo nutné na konci příkazů používat středníky.



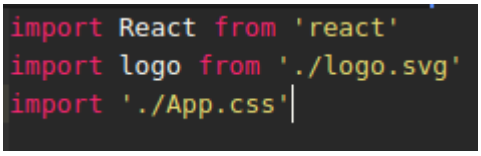
Obrázek 16 - Vlastní modifikace ESLint configu (vlastní zpracování)

Ukázka kódu využívající původní config by tak používala zápis z obrázku níže (Obrázek 17).



Obrázek 17 - Zápis kódu s původním configem (vlastní zpracování)

Zatímco po aplikaci nového configu by stejný kód vypadal takto (Obrázek 18).



Obrázek 18 - Zápis kódu s novým configem (vlastní zpracování)

Oba zápisy jsou z hlediska JS validní, záleží zde opravdu na preferencích jednotlivce, mně osobně však připadá druhý způsob zápisu čistší.

Samozřejmě, že ESLint configů existuje velká spousta, velice populární je např. confug, který používá AirBnb (https://github.com/airbnb/javascript).

Hot reload

Dalším užitečným vylepšení *„create-react-app“* je přidání tzv. *„hot realoadu“*, čili technologie, která sleduje změny v souborech a po jakékoliv jejich změně automaticky

změní obsah vyvíjeného webu. Stejné funkcionality umí docílit sice i defaultně implementovaný *„live reload“*, ovšem ten dělá tzv. *„F5 reload“*, zatímco *„hot reload“* vymění jen změněné části aplikace bez nutnosti kompletního refreshe aplikace. Vynucení *„hot reloaderu“* lze vynutit za pomocí drobné úpravy souboru *„src/index.js“*, do kterého stačí přidat následující kód (Obrázek 19).

```
if (module.hot) {
  module.hot.accept()
}
```

Obrázek 19 - Úprava src/index.js pro vynucení hot reloadu (vlastní zpracování)

Emotion.js

Ke stylování aplikace byl vybrán CSS in JS framework Emotion, ten umožňuje psát CSS přímo do jednotlivých React component a není tak potřeba udržovat styly v separátní CSS souborové struktuře. Odpadá tedy nutnost pro každou komponentu hledat zdrojové CSS v samostatném souboru, jelikož jsou všechny styly spojené s danou komponentou definované přímo v .js souboru dané komponenty. Tento způsob stylování tak pomáhá k udržitelnosti a přehlednosti kódu.

Emotion (https://emotion.sh/) se distribuuje jako npm balíček a lze jej nainstalovat pomocí příkazu:

```
npm i emotion
```

pro práci s Reactem je však lepší použít specifickou verzi vhodnější pro použití s Reactem a sice *„@emotion/core“*:

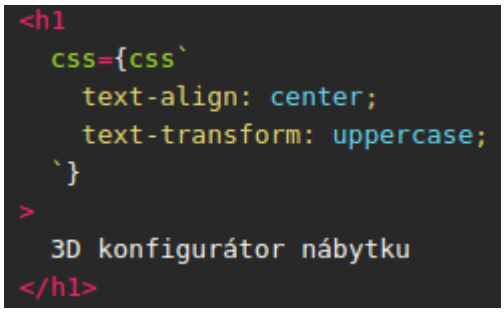
```
npm i @emotion/core
```

Po této instalaci lze již naimportovat do jednotlivých react component hlavní součást tohoto baláčku a sice *„css“* property (Obrázek 20).

```
import { css} from '@emotion/core'
```

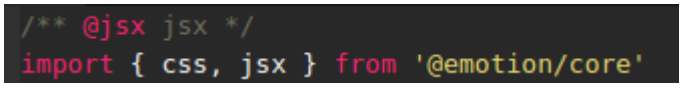
Obrázek 20 - Import Emotion.js (vlastní zpracování)

Stylopis výsledné komponenty s použitím *„css“* property by tak mohl vypadat např. takto (Obrázek 21):



Obrázek 21 - Ukázka užití Emotion.js css property (vlastní zpracování)

Daný kód by však zatím ještě nefungoval. Emotion *„css“* property potřebuje totiž ke svému správnému chodu používat k parsování kódu ještě vlastní *„jsx“* funkci namísto *„React.createElement“* funkce. Toho lze dosáhnout v zásadě 2 způsoby. Buď úpravou Babel presetu nebo pomocí přidání tzv. *„JSX pragmy“*. Jelikož však *„create-react-app“* neumožňuje změny Babel konfigurace, připadá v úvahu pouze 2. varianta (Obrázek 22).



Obrázek 22 - Import Emotion.js spolu s použitím @jsx (vlastní zpracování)

Kód z obrázku výše na začátku jednotlivých Reactích komponent pak umožňuje již bezproblémové používání Emotionu uvnitř React aplikace.

Heroku

Jelikož aplikace potřebuje mít kromě možnosti zveřejnění do repositáře také možnost vybuildování a deploye na nějakou veřejně dostupnou adresu, přichází zde na řadu služba Heroku. Tato služba totiž toto umožňuje. Není však jediná, podobnou funkcionalitu nabízí samozřejmě i mnohá konkurenční řešení např. AWS Amplify (https://aws-amplify.github.io/), Microsoft Azure (https://azure.microsoft.com/cs-cz/), Firebase (https://firebase.google.com/) či Netlify (https://www.netlify.com/). Heroku zde bylo zvoleno víceméně na základě dobré osobní zkušenosti s touto službou z minulých realizovaných projektů.

Samotný *„create-react-app“* frontend lze na Heroku velice lehce vybuildovat za pomoci připraveného Heroku Buildpacku pro *„create-react-app“* (https://github.com/mars/create-react-app-buildpack). Použití tohoto buildpacku se vynutí flagem *„–buildpack mars/create-react-app“* příkazu *„heroku create“*, který vytvoří novou Heroku aplikaci pod přihlášeným Heroku účtem.

Je tedy prvně potřeba vytvořit účet, nainstalovat Heroku, přihlásit se a pak posupovat dle příkazů níže (Obrázek 23):

```
npx create-react-app@3.x $APP_NAME
cd $APP_NAME
heroku create $APP_NAME --buildpack mars/create-react-app
git push heroku master
heroku open
```

Obrázek 23 - Příkazy pro deploy aplikace na Heroku (vlastní zpracování)

Toto by však byl pouze scénář vybuildování frontendové části aplikace. Jelikož je však pro plnohodnotné fungování aplikace 3D návrháře potřeba i backendová část je toto pouze část řešení. Pro vybuildování i backendové části lze použít rozšíření předchozího řešení jako např. *„heroku-cra-node“* (https://github.com/mars/heroku-cra-node), což je předpřipravené řešení, které vytváří základní strukturu aplikace včetně Node.js/Express backendu. I toto řešení však neřeší všechny problémy. Toto řešení sice již počítá s použitím serverové části, neřeší však některé další problémy jako např. spouštění backendu a frontendu vedle sebe jedním příkazem či nastavení proxy pro přesměrování API požadavků na jiný port (na port, na kterém běží server).

Nastavení serveru

Většina výše popisovaných úprav se zatím týkala víceméně frontendové části aplikace. Pro plné využití potenciálu aplikace 3D návrháře je však nutné počítat také s backendovou částí aplikace. Bez ní by např. nebylo možné vytvořit vlastní API pro ukládání objednávek z návrháře do databáze.

Prvním krokem pro zprovoznění backendu je pak vytvoření nové složky pro backendovou část aplikace a nainstalování Express serveru, toho lze docílit buď ručně pomocí:

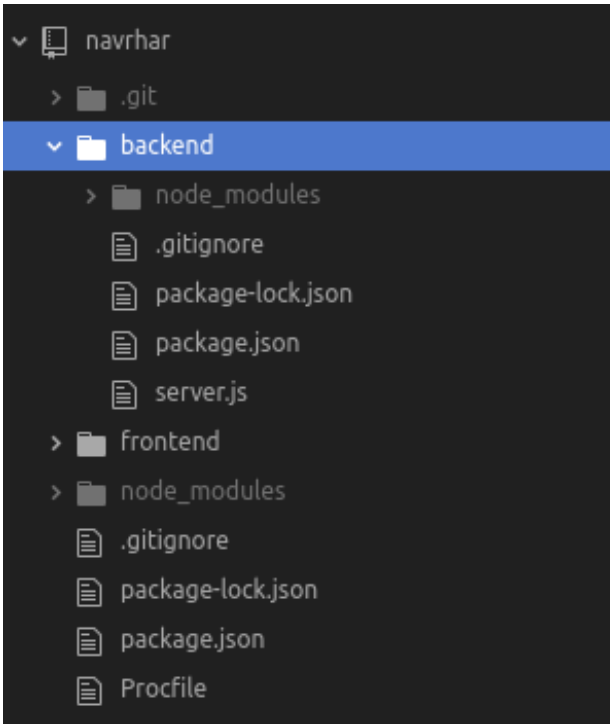
```
mkdir backend
cd backend
npm i express
```

V případě využití *„heroku-cra-node“* pak stačí v server složce aplikace provést instalaci npm balíčků předpřipravených v *„package.json“*, mezi kterými je již i Express a to pomocí:

```
cd server
npm i
```

V aplikaci 3D návrháře byla frontendová a serverová část aplikace zanořena do složek *„frontend“* a *„backend“*. Nová strukturu projektu od této chvíle zachycuje Obrázek 24.

Po instalaci express serveru je již možné začít se základní konfigurací tohoto serveru. V případě 3D návrháře se všechno toto nastavení odehrává v souboru *„backend/server.js“*.



Obrázek 24 - Struktura adresářů projektu (vlastní zpracování)

Nodemon

Pro pohodlnější práci při úpravách *„server.js“* byla přidána podpora *„nodemonu“*. Jelikož je potřeba při každé změně souboru *„server.js“* restartovat server, aby se výsledné změny projevily, je dobré tuto práci automatizovat. To umožňuje právě balíček *„nodemon“*, který lze nakonfigurovat tak, aby po uložení změn do souboru *„server.js“*, se server automaticky restartoval. Vývojář tak nemusí po každé změně server manuálně zastavovat a znovu spouštět. *„Nodemon“* lze nastavit jako skript v *„package.json“* (Obrázek 25).

```
"start": "nodemon -w server.js server.js"
```

Obrázek 25 - Server start script za použití nodem watcheru (vlastní zpracování)

Tento zápis pak umožňuje po spuštění *„npm start“* v *„/backend“* složce projektu sledovat změny v souboru *„server.js“* a pokud nějaké změny zaznamená spustit znovu server dle nastavení v souboru *„server.js“*.

Základní nastavení Express serveru vypadá asi takto (Obrázek 26).

```
const express = require('express')
const app = express()
const port = process.env.PORT || 5000

// console.log that your server is up and running
app.listen(port, () => console.log(`Listening on port ${port}`))
```

Obrázek 26 - Základní nastevení Express serveru (vlastní zpracování)

Nastavení proxy

V souboru *„server.js“* je vidět, že je pro server využíván odlišný port, než pro frontend. Pro frontend je využíván port 3000 (defaultně nastavený port od *„create-react-app“*), backend však musí fungovat na separátním portu, zde např. na portu 5000.

S porty však souvisí ale ještě jedna úprava. Aby aplikace uměla přeposílat frontendové požadavky na backendové API je potřeba do frontendového *„package.json“* přidat adresu proxy serveru (Obrázek 27).

```
"proxy": "http://localhost:5000"
```

Obrázek 27 - Nastavení proxy serveru (vlastní zpracování)

Nastavení concurrently

Další užitečnou úpravou pro práci na projektu je umožnit spouštět backend a frontend současně, doposud se totiž musel spouštět backend přes

```
cd /backend npm start
```

a frontend přes

```
cd /frontend npm start
```

Npm balíček *„concurrently“* však tyto potíže odstraňuje a umožňuje nadefinovat pouze jeden globální proces jak pro backend, tak pro frontend, kdy stačí poté pouze v rootu projektu zavolat *„npm start“* a spustí se jak frontend tak backend.

Výše popsaného lze docílit pomocí níže popsané úpravy „*scripts*“ uvnitř „*package.json*“ umístěného v rootu projektu (Obrázek 28).

```
"client": "cd frontend && npm start",
"server": "cd backend && npm start",
"start": "concurrently \"npm run server\" \"npm run client\""
```

Obrázek 28 - Scripts definované v package.json (vlastní zpracování)

MongoDB

Pro ukládání dat do databáze byla zvolena databáze MongoDB. MongoDB spadá do kategorie tzv. noSQL databází, tzn. že nad touto databází není možné provádět klasické SQL dotazy, jako je tomu u relačních databází. Tyto databáze totiž neukládají data standardně v tabulkách (ve sloupcích a řádcích), ale v JSON, respektive BSON objektech (MongoDB Documentation, BSON Types) a vzhledem k tomu, že je ukládání dat v tomto formátu v JavaScriptu hojně rozšířené, je výhodné tento typ struktury používat.

Implementace MongoDB

MongoDB databáze je opět distribuována jako npm balíček a jelikož se jedná o backendovou závislost, provede se její instalace ve složce backend a to pomocí:

```
npm i mongodb
```

Po této instalaci lze již vykonávat základní MongoDB příkazy pro manipulaci s MongoDB databází. Zatím ovšem pouze lokálně. Pro hostování na MongoDB nelokálně existuje model DBAS (database as a service). Mezi poskytovatele tohoto modelu hostování MongoDB databáze v současné době spadají víceméně 2 hlavní poskytovatelé. MongoDB Atlas (www.mongodb.com) a mLab (https://mlab.com/). Pro účely hostování databáze 3D návrháře byla zvolena služba MongoDB Atlas.

Pro zprovoznění této služby je potřeba prvně vytvořit si účet na mongodb.com a zde pak pojmenovat svoji databázi a případně v ní vytvořit collection (obdoba SQL tabulky), do které mají být následně data ukládána.

Pro spojení s DB na úrovni aplikace probíhá opět v souboru *„backend/server.js“* (Obrázek 29). Zde je potřeba prvně nadefinovat spojení s databází (řádky 9-26) a následně vytvořit API, přes které bude backend obdržovat data od frontendu (ř. 28-34). V případě

3D návrháře je toto API dostupné na adrese *„api/post“*. Obdobným způsobem lze dále definovat i API pro získávání informací z databáze na *„api/get“*.

Následně je také potřeba implementovat připojení odesílání dat na toto API na frontendu. Pro odeslání všech parametrů objednávky z 3D návrháře vypadá implemetance na straně frontendu (Obrázek 30).

Po těchto krocích lze úspěšně odesílat objednávky do databáze. Ukázka uloženého objektu s informacemi o objednávce ukazuje obrázek níže (Obrázek 31). Zde je vidět ještě jedno specifikum noSQL databází a sice automatické vytvoření primárního klíče *„_id“*, který by musel být v SQL databázi vytvořen ručně.

```
const express = require('express')
const MongoClient = require('mongodb').MongoClient
const bodyParser = require('body-parser')
const app = express()
const port = process.env.PORT || 5000

app.use(bodyParser.json())

const CONNECTION_URL =
  'mongodb+srv://user:pass@clustername-rzyff.mongodb.net/dbname?retryWrites=true&w=majority'
const DATABASE_NAME = 'test'

app.listen(port, () => {
  MongoClient.connect(
    CONNECTION_URL,
    { useNewUrlParser: true },
    (error, client) => {
      if (error) {
        throw error
      }
      database = client.db(DATABASE_NAME)
      collection = database.collection('colectionname')
      console.log('Connected to `' + DATABASE_NAME + '`!')
    }
  )
})

app.post('/api/post', (req, res) => {
  collection.insertOne(req.body, (error, result) => {
    if (error) {
      return response.status(500).send(error)
    }
  })
})
```

Obrázek 29 - Implementace připojení k MongoDB v backend/server.js (vlastní zpracování)

```
const parameters = props.parameters

const createIssue = newIssue => {
  fetch('/api/post', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(newIssue)
  })
    .then(response => response.json())
    .then(data => {
      console.log('Success:', data)
    })
    .catch(error => {
      console.error('Error:', error)
    })
}

function handleSubmit(e) {
  createIssue(parameters)
  e.preventDefault()
}
```

Obrázek 30 - připojení na API na straně frontendu (vlastní zpracování)


```
_id: ObjectId("5e820695133cf2736feab5a8")
width: 0.6
height: 2
depth: 0.55
color: "#966239"
backDesk: "0"
bottomDesk: "1"
topDesk: "1"
deskWidth: 0.02
parts: "3"
currentPart: 2
step: 4
showCurrent: true
currentColor: "#6ba92c"
bgColor: "#4d4b48"
configuration: Array
  0: 3
  1: 5
  2: 3
  3: 3
  4: 3
name: "František"
surName: "Novák"
street: "Lhota"
cp: "101"
zip: "23476"
phone: "777888666"
note: "Poznámka..."
```


Obrázek 31 - Ukázka výsledného objektu objednávky uloženého v MongoDB databázi (vlastní zpracování)

Tvorba 3D modelu skříně

K vizualizaci 3D objektů v prostředí webových aplikací lze použít knihovnu WebGL. Tato knihovna by však sama o sobě byla jen stěží použitelná. Definice 3D objektů a práce s nimi je zde totiž na velice nízko-úrovňové rovině, bylo proto potřeba vybrat nějakou z JS knihoven, která jednotlivé funkce této knihovny abstrahuje na použitelnější (pochopitelnější), úroveň. Jednou z těchto knihoven je např. Three.js (https://threejs.org/). Použití této knihovny spolu s Reactem však bohužel tzv. „hází klacky pod nohy". Three.js je určeno pro aplikace s imperativním stylem programování, ve kterém se programuje právě třeba v čistém JavaScriptu. Problémem však je, že React funguje na úplně jiném paradigmatu a sice s deklarativním programováním. Bylo tedy třeba vybrat knihovnu, která tuto Three.js knihovnu ohýbá tak, aby šla používat komponentově. Jednou z těch knihoven, někdy také zvaných wrapperů (Vorontsova, 2019), je např. *„react-three-fiber"* a právě této knihovny bylo použito pro tvorbu aplikace 3D návrháře.

Tato knihovna se opět distribuuje jako npm modul:

```
npm i react-three-fiber
```

Základní komponentou z tohoto balíčku je pak Canvas:

```
import { Canvas } from 'react-three-fiber
```

Canvas stejně jako HTML5 Canvas vytváří základní plátno, do kterého jsou následně umisťovány jednotlivé objekty. Tato komponenta dále také přijímá několik properties. Nejdůležitější z nich je property *„camera"*, ta určuje pod jakým úhlem, respektive z jakého místa (z jakých x,y,z souřadnic), má být nahlíženo na objekty renderované uvnitř Canvasu. Krátkým experimentováním bylo dojito k nastavení *„-1.2, 0, 2.6"* (Obrázek 32).

```
<Canvas camera={{ position: [-1.2, 0, 2.6] }}>
  <Scene
    parameters={props.parameters}
    setParameters={props.setParameters}
  />
</Canvas>
```

Obrázek 32 - Ukázka použití Three.js Canvas komponenty (vlastní zpracování)

Zde je nejzajímavější asi *„x“* souřadnice, která svoji zápornou hodnotou umožňuje zobrazit skříň lehce z boku, což vzbuzuje plastičtější dojem, než by vzbuzoval pohled přímo zepředu, kde by byla např. špatně pozorovatelná hloubka skříně.

Aby se však měla kamera vůbec na co dívat, je potřeba tyto objekty definovat. Většinu této logiky obstarává komponenta „<*Scene*>“. Uvnitř této komponenty jsou pak definovány všechny základní objekty definující skříň. Jelikož se celá skříň skládá víceméně z několika nějak do sebe zakomponovaných desek, lze většinu skříně sestavit pomocí různě definovaných kvádrů. Pro tento účel byla za pomoci *„react-three-fiber“* vytvořena komponenta „<*Cube*>“ (Obrázek 33).

```
const Cube = props => {
  let color = parseInt(props.color.replace('#', '0x'), 16)
  return (
    <mesh position={props.position} onClick={props.onClick}>
      <boxBufferGeometry attach='geometry' args={props.size} />
      <meshStandardMaterial attach='material' color={color} />
    </mesh>
  )
}
```

Obrázek 33 - Ukázka implementace Cube komponenty na vykreslení desek skříně (vlastní zpracování)

Tato komponenta byla upravena tak, aby mohla být v co největší míře znovupoužitelná, může tak přijímat různé properties jako např. barvu, pozici, velikost, či onClick event.

Na obrázku níže (Obrázek 34) je pak vidět např. její použití pro render horní stěny skříně.

1. parametr *„size“* je ze všech nejjednodušší, určuje pouze rozměry desky. V tomto případě je horní deska stejně široká jako šířka celé skříně (*width*), vysoká jako výška desky (*deskWidth*) a hluboká jako hloubka skříně (*depth*). Problematičtější fáze však nastává hlavně u výpočtu pozice jednotlivých desek. Skříň se totiž musí rovnoměrně rozšiřovat či zužovat doleva a doprava, aby vždy zůstala z uživatelova pohledu ve středu <*Canvas*>*u*.

Pro tento účel zde existuje proměnná *„halfToLeft“*, která stanovuje začátek vykreslování skříně přesně jako polovinu šířky skříně doleva od středu soustavy souřadnic. Dále je potřeba zmínit, že skříň může mít několik dílů, jednotlivé díly se tak renderují cyklem a každá skříň má posunutý začátek podle pozice, na které se v rámci celé skříně

nachází. K tomuto účelu zde existuje ve výpočtu pozice skříně iterační proměnná *„i“*. Výpočet také dále musí počítat např. s šířkou desky.

```
{props.parameters.topDesk === true && (
  <Cube
    size={[
      props.parameters.width,
      props.parameters.deskWidth,
      props.parameters.depth
    ]}
    position={[
      halfToLeft +
        (i + 1) * props.parameters.width -
        props.parameters.width / 2, //horni stena
      0.5 -
        props.parameters.deskWidth / 2 +
        (props.parameters.height - 1) * 0.5,
      0
    ]}
    color={
      props.parameters.currentPart === i + 1 &&
      props.parameters.step === 3 &&
      props.parameters.showCurrent === true
        ? props.parameters.currentColor
        : props.parameters.color
    }
    onClick={e => {
      props.setParameters({
        ...props.parameters,
        currentPart: i + 1
      })
      e.stopPropagation()
    }}
  />
)}
```

Obrázek 34 - Implementace renderu horní stěny v 3D modelu návrháře (vlastní zpracování)

Barva jednotlivých desek se řídí podle definice barvy zvolené uživatelem, ovšem tato barva se přebíjí barvou aktuálně zvolené skříně, pokud se uživatel nachází v části aplikace, kde se nějakou barvou musí odlišit právě upravovaná skříň.

Nakonec pak *„onClick“* event zajišťuje výběr konkrétní části skříně k editaci. Zde je zajímavé použití funkce *„stopPropagation()“*, která řeší zajímavý problém, který při vývoji vyvstal. Pokud totiž uživatel kliká na nějakou část skříně z takového úhlu, pod kterým se pod sebou zobrazuje více částí skříní, tak *„onClick“* event probublá až na

poslední (nejvzdálenější) objekt a spustí *„onClick“* event na objektu, na který uživatel vůbec netuší, že klikl. Funkce *„stopPropagation()“* toto probublávání *„onClick“* eventu zastavuje a zapříčiňuje, že se *„onClick“* event spustí opravdu pouze na první viditelné vrstvě z uživatelova pohledu.

Další zajímavou částí modelu nejsou však pouze objekty definující skříň. *„React-three-fiber“* umožňuje navíc i taktové vychytávky jako např. osvícení scény. Nakonfigurovat správné osvětlení scény je sice celkem „husarský kousek“, ale nakonec se jako celkem účinná kombinace ukázala následující konfigurace (Obrázek 35).

```
<ambientLight intensity={0.9} />
<pointLight intensity={0.9} position={[0.2, 0.2, -3]} />
```

Obrázek 35 - Konfigurace osvětlení scény návrháře (vlastní zpracování)

<*ambientLight* /> zde representuje ambientní osvícení, tzn. rovnoměrné osvícení celé scény, <*pointLight*/> pak bodové světlo zaměřující svůj jas pouze na určené místo.

Routování

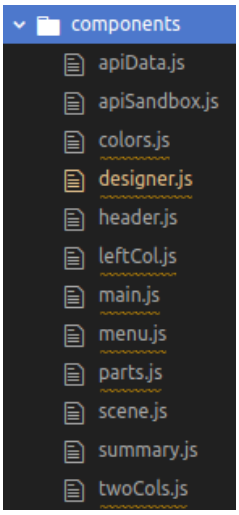
Asi poslední zajímavou částí aplikace je vyřešení problému změny uživatelského rozhraní odvozeného od části aplikace (adresy), na které se uživatel právě nachází. K tomuto účelu existuje balíček *„react-router-dom“* (https://reacttraining.com/react-router), který dokáže měnit URL adresy bez reloadu aplikace a dle různých URL načítat relevantní obsah (komponenty). Ukázka načítání relevantních částí aplikace dle kroku, ve kterém se uživatel právě nachází, pak vypadá v 3D návrháři takto (Obrázek 36).

```
<Switch>
  <Route exact path='/'>
    <TwoCols>
      <LeftCol>
        <Menu parameters={parameters} setParameters={setParameters} />
        <Main parameters={parameters} setParameters={setParameters} />
      </LeftCol>
      <Designer
        parameters={parameters}
        setParameters={setParameters}
      ></Designer>
    </TwoCols>
  </Route>
  <Route path='/colors/'>
    <TwoCols>
      <LeftCol>
        <Menu parameters={parameters} setParameters={setParameters} />
        <Colors parameters={parameters} setParameters={setParameters} />
      </LeftCol>
      <Designer
        parameters={parameters}
        setParameters={setParameters}
      ></Designer>
    </TwoCols>
  </Route>
  <Route path='/parts/'>
    <TwoCols>
      <LeftCol>
        <Menu parameters={parameters} setParameters={setParameters} />
        <Parts parameters={parameters} setParameters={setParameters} />
      </LeftCol>
    </TwoCols>
  </Route>
```

Obrázek 36 - Ukázka nastavení routování aplikace (vlastní zpracování)

Komponenty

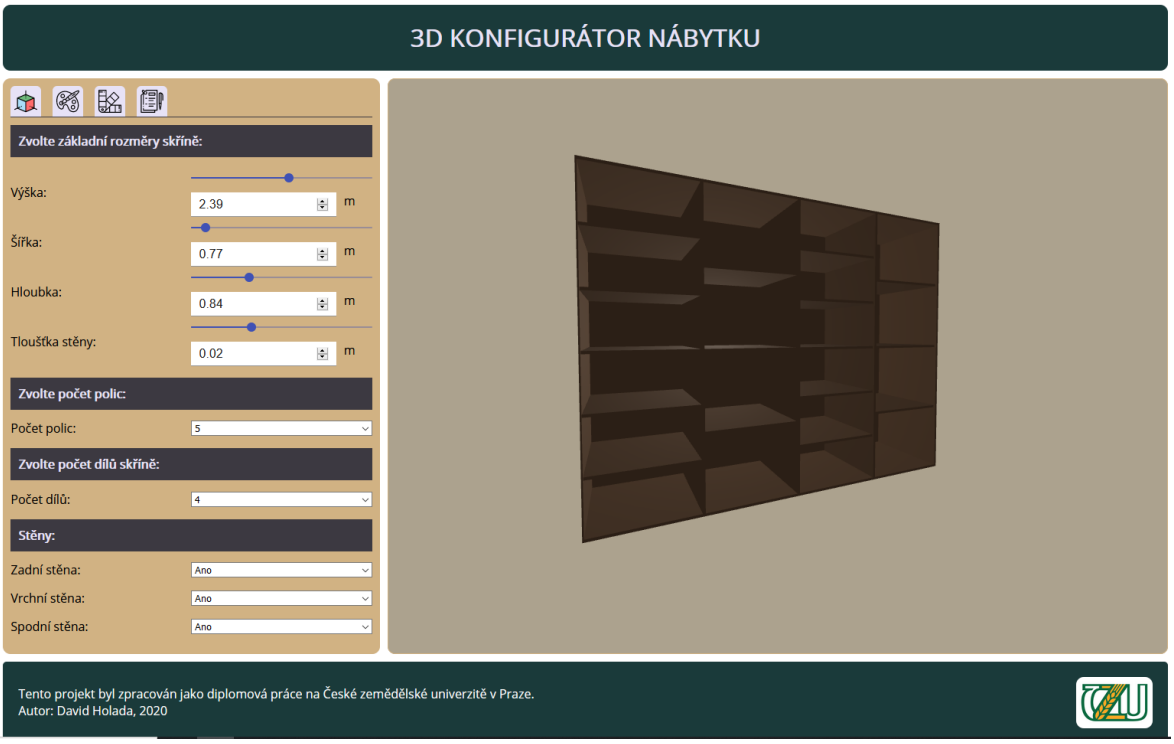
Závěrem by bylo vhodné ukázat atomičnost, jíž bylo využito při tvorbě aplikace 3D návrháře. Obrázek níže (Obrázek 37) znázorňuje, že výsledný kód Reacových částí aplikace byl rozdělen do 12 logicky oddělených celků.



Obrázek 37 - Výsledné React komponenty aplikace 3D návrháře (vlastní zpracování)

Hotová aplikace

Díky nastudování potřebných technologií a následné aplikaci nabytých poznatků mohla být vytvořena finální podoba aplikace 3D návrháře, který nejen díky možnostem Reactu a WebGL splnil všechny vytyčené cíle z kapitoly 4. 4. 2.



Obrázek 38- Výsledná podoba 3D návrháře nábytku (vlastní zpracování)

Podrobný postup programování celé aplikace lze najít v přílohách na obrázcích č. 42-46.

Samotná aplikace je vybuildována na adrese https://navrhar.herokuapp.com/. Její neinteraktivní verzi pak zachycuje Obrázek 38 výše.

3.5 TECHNOLOGICKÁ KOMPARACE

Využití technologie React.js spolu s WebGL se ukázalo jako správné řešení. Toto řešení technologicky překračuje starší technologie jako čistý JS či jQuery a HTML5 Canvas.

Hlavní rozdíl nového řešení lze spatřit hlavně v rychlosti aplikace, struktuře a přehlednosti kódu.

Rychlost aplikace je zajištěna díky React.js Virtual DOMu, který bez jakékoliv latence dokáže reflektovat všechny změny z formulářových prvků do modelu skříně.

Strukturu kódu pak zajišťuje React spolu s React Three Fiber možností definovat WebGL objekty pomoci deklarativního zápisu. Tyto deklarativně psané WebGL objekty pak mohou pružně měnit svoje charakteristiky dle aktuálního stavu React aplikace. Této výhodné struktury by v klasickém JS imperativním programování nešlo dosáhnout.

Přehlednost pak zajišťuje React díky svému komponentovému přístupu, kdy se kód přirozeně dělí do pokud možno co funkčně nejatomičtějších částí.

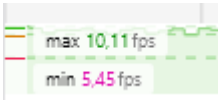
3.6 KVANTIFIKACE VÝSLEDKŮ PRÁCE

Kvantifikovaná komparace jednotlivých návrhářů by byla bohužel asi jen těžko proveditelná vzhledem k různorodosti jednotlivých řešení, proto bylo pro toto porovnání vybráno nejpodobnější a zároveň technologicky nejvyspělejší řešení a sice návrhář nábytku nabyteknamiru.cz.

3.6.1 Plynulost pohybu modelu skříně

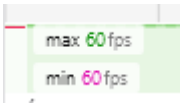
Díky nástroji Frame rate (https://developer.mozilla.org/en-US/docs/Tools/Performance/Frame_rate) zabudovaném ve vývojářských nástrojích prohlížeče Firefox lze kvantifikovat plynulost pohybu 3D modelu skříně.

Toto měření potvrdilo už na první pohled znatelnou domněnku, že pohyb 3D modelu nábytku na webu nabyteknamiru.cz není plynulý. Názory na to, co je pro oko již plynulé se různí. Model by však měl dosahovat alespoň snímkové frekvence, ve které se natáčejí videa čili 24 snímků za sekundu, nejlépe pak 60 snímků za sekundu, což je maximální možný počet, který dokáže zobrazit běžný monitor. Bohužel model skříně na nabyteknamiru.cz dosáhl na snímkovou frekvenci pouze někde mezi **5.45** a **10.11** snímky za sekundu.



Obrázek 39 - Snímková frekvence 3D modelu návrháře nábytku nabyteknamiru.cz (vlastní zpracování)

Pokud bylo stejné měření provedeno na návrháři, který je výsledkem této práce, bylo dosáhnuto bez problému plynulých **60** snímků za sekundu.

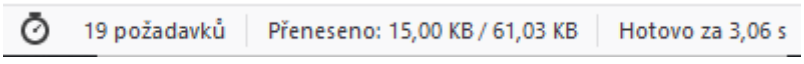


Obrázek 40 - Snímková frekvence vlastního řešení návrháře (vlastní zpracování)

Tento rozdíl pravděpodobně způsobuje nativní vs. nenativní řešení 3D modelu skříně. Vzhledem k tomu, že nabyteknamiru.cz konvertuje modely z desktopové aplikace Imos do webového zobrazení naproti nativnímu modelování přímo ve WebGL, kterého využívá návrhář vytvářený v této práci.

3.6.2 Rychlost propisování změn do modelu skříně

Zde nabyteknamiru.cz naráží na podobný problém. Jelikož modely nejsou modelovány přímo na webu, ale jsou načítány modely vytvořené v desktopové aplikaci, musí se pro každou změnu rozměrů načíst nový model skříně, a to trvá průměrně kolem 3 sekund (Obrázek 41).



Obrázek 41 - Měření rychlosti načtení nového modelu skříně na nabyteknamiru.cz

Během této doby se točí načítací kolečko a veškerý interaktivní dojem se v tu chvíli vytrácí. Kdyby tento návrhář používal modely matematicky definované přímo ve WebGL, nemusel by se pro každý rozměr načítat zbytečně nový model, protože by se jen

upravily parametry toho stávajícího, který by se podle nových parametrů automaticky aktualizoval.

Návrhář, který je výsledkem této práce, se vydal právě tímto směrem a jeho latence při změně modelu je tak vlastně nulová (respektive na tolik nízká, že je prakticky neměřitelná).

3.7 ZOBECNITELNOST VÝSLEDKŮ

Nově vytvořenou aplikaci 3D návrháře skříní lze již nyní považovat za solidní základ, který je možno dále nabídnout a znovupoužít pro potřeby dalších jednotlivých výrobců ať už skříní nebo nábytku obecně. Ve spolupráci s těmito výrobci, kteří již mají připravené modely velkého množství nábytku, by šlo tyto modely následně použít a integrovat do nového návrháře či jakkoliv jinak pozměnit funkcionalitu návrháře dle přání zadavatele.

4 ZÁVĚR

Tato práce přinesla vhled do současného vývoje na poli webového front-endu. Tento vývoj byl demonstrován komparací dvou posledních stupňů technologií pro tvorbu interaktivních webových aplikací na straně klienta: jQuery a SPA frameworků, zejména pak Reactu.

Současný stav využití těchto technologií byl ověřen na celkem rozsáhlém českém nábytkářském trhu, konkrétně pak na všemožných variantách aplikací webových návrhářů nábytku na míru. Bylo zjištěno, že současný stav na tomto trhu trpí velkým technologickým deficitem a že téměř všechna současná řešení využívají již zastaralých technologií.

Na základě toho došlo ke stanovení cíle vytvoření vlastního řešení 3D návrháře nábytku na míru, který by tímto technologickým deficitem již netrpěl. Pro dostání tomuto cíli byl zvolen moderní SPA framework React spolu s 3D renderovací webovou technologií WebGL. Za pomoci těchto nástrojů bylo možné naprogramovat vlastní řešení 3D návrháře nábytku, které eliminovalo technologické a s tím i nepřímo související funkční nedostatky současných řešení.

Výsledné řešení bylo umístěno na adresu http://navrhar.herokuapp.com. Toto řešení může nyní sloužit přinejmenším jako technologické demo, a hlavně být dále nabídnuto a přizpůsobeno potřebám jednotlivých subjektů působícím na českém nábytkářském trhu, které by o toto řešení mohli projevit zájem. Velké množství těchto subjektů bylo konec konců i definováno v kapitole „Průzkum trhu“.

5 SEZNAM POUŽITÝCH ZDROJŮ

Bibliografie

Adam, Marc F. 2017. The History and Impact of Node.js. *Nixa.ca.* [Online] 10 2017. https://nixa.ca/blog/the-history-and-impact-of-nodejs/.

AngularJS. *StartupJobs.* [Online] https://www.startupjobs.cz/nabidky?technologie=angularjs.

Anthony, Sebastian. 2011. The Browser Cold War. *Extremetech.com.* [Online] 15. 12 2011. https://www.extremetech.com/computing/109127-the-browser-cold-war.

Aston, Ben. 2015. A brief history of JavaScript. *Medium.com.* [Online] 1. 4 2015. https://medium.com/@benastontweet/lesson-1a-the-history-of-javascript-8c1ce3bffb17.

Banks, Alex. 2017. *Learning React: Functional Web Development with React and Redux.* místo neznámé : O'Reilly Media, Inc., 2017. ISBN: 9781491954621.

BSON Types. *MongoDB documentation.* [Online] https://docs.mongodb.com/manual/reference/bson-types/.

CanIUse. 2019. ECMAScript 2015 (ES6). *CanIUse.com.* [Online] 2019. https://caniuse.com/#feat=es6.

Cassel, David. 2018. Brendan Eich on Creating JavaScript in 10 Days, and What He'd Do Differently Today. *TheNewStack.* [Online] 26. 8 2018. https://thenewstack.io/brendan-eich-on-creating-javascript-in-10-days-and-what-hed-do-differently-today/.

Components and Props. *React Docs.* [Online] https://reactjs.org/docs/components-and-props.html.

CSS align-items Property. *W3Schools.* [Online] https://www.w3schools.com/cssref/css3_pr_align-items.asp.

CSS align-self Property. *W3Schools.* [Online] https://www.w3schools.com/cssref/css3_pr_align-self.asp.

CSS flex-basis Property. *W3Schools.* [Online] https://www.w3schools.com/cssref/css3_pr_flex-basis.asp.

CSS flex-direction Property. *W3Schools.* [Online] https://www.w3schools.com/cssref/css3_pr_flex-direction.asp.

CSS flex-grow Property. *W3Schools.* [Online] https://www.w3schools.com/cssref/css3_pr_flex-grow.asp.

CSS flex-shrink Property. *W3Schools.* [Online]
https://www.w3schools.com/cssref/css3_pr_flex-shrink.asp.
CSS flex-wrap Property. *W3Schools.* [Online]
https://www.w3schools.com/cssref/css3_pr_flex-wrap.asp.
CSS justify-content Property. *W3Schools.* [Online]
https://www.w3schools.com/cssref/css3_pr_justify-content.asp.
CSS order Property. *W3Schools.* [Online]
https://www.w3schools.com/cssref/css3_pr_order.asp.
CSS-tricks. A Complete Guide to Flexbox. *CSS-tricks.com.* [Online] https://css-tricks.com/snippets/css/a-guide-to-flexbox/.
Delaney, Jeff. 2019. The Weird History of JavaScript . *Dev.to.* [Online] 20. 6 2019.
https://dev.to/codediodeio/the-weird-history-of-javascript-2bnb.
Docs, React. Reconciliation. *Reactjs.org.* [Online]
https://reactjs.org/docs/reconciliation.html.
Facebook. Create a New React App. *React documentation.* [Online]
https://reactjs.org/docs/create-a-new-react-app.html.
Flavio. 2018. The Complete ECMAScript 2015-2019 Guide. *FlavioCopes.* [Online] 1. 2 2018. https://flaviocopes.com/ecmascript/.
Gavigan, Dave. 2018. The History of Angular. *Medium.com.* [Online] 3. 4 2018.
https://medium.com/the-startup-lab-blog/the-history-of-angular-3e36f7e828c7.
Getting Started. *Styled Components Docs.* [Online] https://styled-components.com/docs/basics#getting-started.
Getting Started - First-Time Git Setup. *Git Documentation.* [Online] https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup.
Github. 2017. Styled Components. *Github.com.* [Online] 2017. https://github.com/styled-components/styled-components/releases?after=v2.0.0-6.
Gudelli, Arunkumar. 2019. History of AngularJs. *Angular Wiki.* [Online] 19. 10 2019.
https://www.angularjswiki.com/angular/history-of-angularjs/.
Guo, Danny. 2019. The history and legacy of jQuery. *LogRocket blog.* [Online] 13. 8 2019. https://blog.logrocket.com/the-history-and-legacy-of-jquery/.
Hermans, Liesbeth. 2018. Vue on 2018 — Interview with Evan You, author of the Vue.js framework. *blog.hackages.io.* [Online] 6. 2 2018. https://blog.hackages.io/https-blog-hackages-io-evanyoubhack2017-cc5559806157.

Hoffmann, Jay. 2019. What Does AJAX Even Stand For? *Thehistoryoftheweb.com.* [Online] 4. 3 2019. https://thehistoryoftheweb.com/what-does-ajax-even-stand-for/.

Chima, Sarah. 2017. Var, let and const- what's the difference? *Dev.to.* [Online] 25. 10 2017. https://dev.to/sarah_chima/var-let-and-const--whats-the-difference-69e.

Jasraj. MERN stack. *Geeks for geeks.* [Online] https://www.geeksforgeeks.org/mern-stack/.

Kos, Ladislav. 2019. Infografika: Podíl vyhledávačů Google a Seznam na českém internetu #2019. *Evisions.cz.* [Online] 24. 1 2019. https://www.evisions.cz/blog-2019-01-24-infografika-podil-vyhledavacu-google-a-seznam-na-ceskem-internetu-2019/.

Kyrnin, Jennifer. 2020. Avoiding Inline Styles for CSS Design . *Lifewire.* [Online] 8. 3 2020. https://www.lifewire.com/avoid-inline-styles-for-css-3466846.

Kyrnin, Jennifer. 2020. The Difference Between CSS2 and CSS3 . *Lifewire.* [Online] 12. 3 2020. https://www.lifewire.com/css2-vs-css3-3466978.

Llobera, Lewis. 2020. Using the Public Folder. *Create React App.* [Online] 13. 2 2020. https://create-react-app.dev/docs/using-the-public-folder/.

Naughton, John. 2015. Netscape: the web browser that came back to haunt Microsoft . *The Guardian.* [Online] 22. 3 2015. https://www.theguardian.com/global/2015/mar/22/web-browser-came-back-haunt-microsoft.

Overview. *LESS.* [Online] http://lesscss.org/.

PAX. *IKEA.* [Online] https://www.ikea.com/cz/cs/planners/pax-pubbc841fc0.

Peyrott, Sebastian. 2017. A Brief History of JavaScript. *Auth0.* [Online] 16. 1 2017. https://auth0.com/blog/a-brief-history-of-javascript/.

Ramanadham, Kiran. 2019. ES5, ES6, ES7, ES8, ES9: What’s new in each Version of JavaScript. *GreyCampus.* [Online] 29. 5 2019. https://www.greycampus.com/blog/programming/java-script-versions.

React JS. *StartupJobs.* [Online] https://www.startupjobs.cz/nabidky?technologie=react-js.

Salvet, Pavel. 2017. Proměnné v JavaScriptu. *Interval.cz.* [Online] 6. 4 2017. https://www.interval.cz/clanky/promenne-v-javascriptu/.

Source Maps. *Emotion Docs.* [Online] https://emotion.sh/docs/source-maps.

The css Prop. *Emotion Docs.* [Online] https://emotion.sh/docs/css-prop.

Using the State Hook. *React Docs.* [Online] https://reactjs.org/docs/hooks-state.html.

Virtual DOM and Internals. *React Docs.* [Online] https://reactjs.org/docs/faq-internals.html.

Vorontsova, Marina. 2019. Starting with React 16 and Three.js in 5 minutes. *Bits and Pieces.* [Online] 8. 5 2019. https://blog.bitsrc.io/starting-with-react-16-and-three-js-in-5-minutes-3079b8829817.

Vue.js. *StartupJobs.* [Online] https://www.startupjobs.cz/nabidky?technologie=vue-js.

W3Schools. ECMAScript 5 - JavaScript 5. *W3Schools.* [Online] https://www.w3schools.com/js/js_es5.asp.

W3Schools. JavaScript Versions. *W3Schools.* [Online] https://www.w3schools.com/js/js_versions.asp.

Wanyoike, Michael. 2018. History of front-end frameworks. *LogRocket blog.* [Online] 16. 10 2018. https://blog.logrocket.com/history-of-frontend-frameworks/.

Přílohy

5.1 PRŮZKUM TRHU – VÝSLEDKY VYHLEDÁVÁNÍ

Google.cz		**Seznam.cz**	
Skříň na míru	**Skříně na míru**	**Skříň na míru**	**Skříně na míru**
stako.cz	skrinenamiru.cz	skrin.cz	skrine-komandor.cz
indeco.cz	stako.cz	amonit.cz	amonit.cz
hanak-nabytek.cz	indeco.cz	skrine-komandor.cz	skrin.cz
ikea.com	hanak-nabytek.cz	sten-skrine.cz	sten-skrine.cz
vestavenky.cz	ikea.com	vestavenky.cz	jninterier.cz
skrinenamiru.cz	jninterier.cz	skrine-mladaboleslav.cz	vestavene-skrine-max.cz
jninterier.cz	vestavenky.cz	jninterier.cz	vestavene-skrine-ardekor.cz
amonit.cz	amonit.cz	dverecag.cz	dverecag.cz
sten-skrine.cz	skrine-komandor.cz	stako.cz	stako.cz
skrine-komandor.cz	sten-skrine.cz	vestavene-skrine-ardekor.cz	vestavenky.cz
navrhisivestavku.cz	navrhnisivestavku.cz	vestavene-skrine-max.cz	skrine-mladaboleslav.cz
andalo-skrine.cz	vestavene-skrine-ardekor.cz	a-typ.cz	a-typ.cz
levne-skrine.cz	pjatak.cz	truhlarstvi-bomi.cz	maryskonabyteknamiru.cz
pjatak.cz	levne-skrine.cz	skrine.cz	indoor.cz
moebelix.cz	moebelix.cz	interiery-skrine.cz	simcak.net
truhlarstvimicek.	ujirika.cz	design-nadto.cz	truhlarstvibomi.cz

cz			
vestavene-skrine-ardekor.cz	skrinekatrin.cz	indeco.cz	intery-skrine.cz
skrine-mladaboleslav.cz	truhlarstvimicek.cz	indoor.cz	indeco.cz
halbos.cz	hanak-skrine.cz	lj-kuchyne.cz	skrinenamiru.cz
cizek-skrine.cz	skrine-mladaboleslav.cz	simcak.net	skrine.cz
skrinekatrin.cz	lj-kuchyne.cz	vestavene-skrine-petricek.cz	design-nadto.cz
dverecag.cz	skrine.cz	kapitan.cz	lj-kuchyne.cz
skrin.cz	nabytekpiza.cz	zlutahala.cz	levne-skrine.cz
indoor.cz	nabytek-dan.cz	levne-skrine.cz	nabyteknamiru.cz
		nabytek-na-miru-brno.cz	kapitan.cz
		nabyteknamiru.cz	zakazkovy-interier.cz
		creative-interior.cz	nabytek-na-miru-brno.cz

Tabulka 7 - Výsledky vyhledávání pro klíčová slova "Skříň na míru" a "Skříně na míru"

Google.cz		**Seznam.cz**	
Skříň na zakázku	**Skříně na zakázku**	**Skříň na zakázku**	**Skříně na zakázku**
stako.cz	stako.cz	truhlarstvi-marek.cz	vestavenky.cz
hanak-nabytek.cz	hanak-nabytek.cz	vestavenky.cz	truhlarstvi-matek.cz
indeco.cz	indeco.cz	vestavene-skrine-ardekor.cz	vestavene-skrine-ardekor.cz
jninterier.cz	skrinenamiru.cz	jninterier.cz	salu.cz

vestavenky.cz	jninterier.cz	a-typ.cz	nabytekriha.cz
skrinenamiru.cz	vestavenky.cz	heth.cz	a-typ.cz
skrine-komandor.cz	amonit.cz	nabytekriha.cz	heth.cz
sten-skrine.cz	sten-skrine.cz	kuchyne-brno.eu	jninterier.cz
amonit.cz	skrine-komandor.cz	stylvenkova.cz	truhlarstvi-balihar.cz
andalo-skrine.cz	andalo-skrine.cz	truhlarstvi-balihar.cz	kuchyne-brno.eu
navrhnisivestavku.cz	pjatak.cz	astr.cz	stylvenkova.cz
skrine-mladaboleslav.cz	vestavene-skrine-ardekor.cz	bevedo.cz	truhlarstviex.cz
pjatak.cz	skrine-liberec.cz	salu.cz	akvamex.cz
vestavene-skrine-ardekor.cz	levne-skrine.cz	simek-interier.cz	nabytekvyroba.cz
kuchyne-brno.eu	skrinekatrin.cz	dreamwood.cz	novak-nabytek.cz
skrinekatrin.cz	lj-kuchyne.cz	truhlarstviex.cz	astr.cz
lj-kuchyne.cz	vestavene-skrine-petricek.cz	novak-nabytek.cz	nabytek-max.cz
levne-skrine.cz	a-typ.cz	nabytekvyroba.cz	bevedo.cz
a-typ.cz	cizek-skrine.cz	nabytek-max.cz	bpparket.cz
halbos.cz	nabytek-matefi.cz	hezkakuchyn.cz	simek-interier.cz
nabytekriha.cz	halbos.cz	favi.cz	dreamwood.cz
skrin.cz	ujirika.cz		
vestavene-skrine-petricek.cz	apokork.cz		

cizek-skrine.cz	truhlarstvi-zdara.cz		
truhlarstvimicek.cz	truhlarstvimicek.cz		
sapelky.cz	truhlarstvi-futo.cz		
ujirika.cz	hezkydomov.cz		

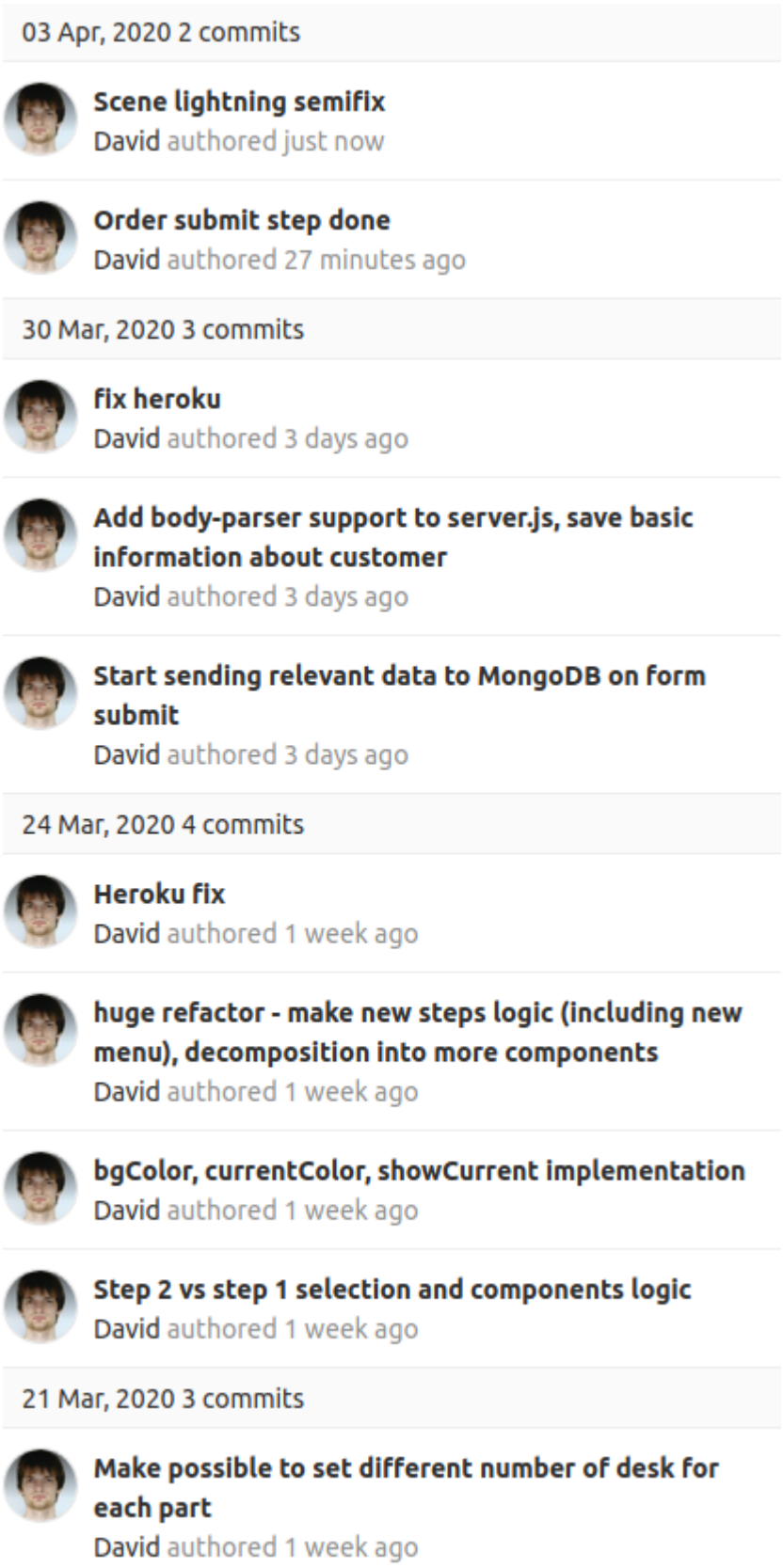
Tabulka 8 - Výsledky vyhledávání pro klíčová slova "Skříň na zakázku" a "Skříně na zakázku"

Google.cz		**Seznam.cz**	
Nábytek na míru	**Nábytek na zakázku**	**Nábytek na míru**	**Nábytek na zakázku**
nabytek-max.cz	jninterier.cz	nabytek-max.cz	truhlarstvi-marek.cz
jninterier.cz	truhlarstvi-luxra.cz	indeco.cz	nabytek-max.cz
jamall.cz	nabytek-na-zakazku.com	nadop.cz	variowall.cz
indeco.cz	nabytek-trend.cz	nabytek-lupinovka.cz	holik.cz
nabyteknamiru.cz	truhlarstvi-marecek.cz	triant.cz	jninterier.cz
truhlarstvi-marecek.cz	truhlarstvi-marek.cz	frosch.cz	truhlarstvi-marecek.cz
stako.cz	mbyt.cz	lj-kuchyne.cz	a-typ.cz
nabytek-na-zakazku.com	migala.cz	hezkakuchyn.cz	mbyt.cz
laminonabyteknamiru.cz	gilikdesign.cz	jirecek.cz	aktivpisek.cz
truhlarstvi-marek.cz	nabytek-max.cz	siko.cz	prostor.cz
nabytek-matefi.cz	truhlarstvi-borohradek.cz	a-typ.cz	design-nadto.cz
nabytek-trend.cz	jamall.cz	klasikcz.eu	bdorg.cz

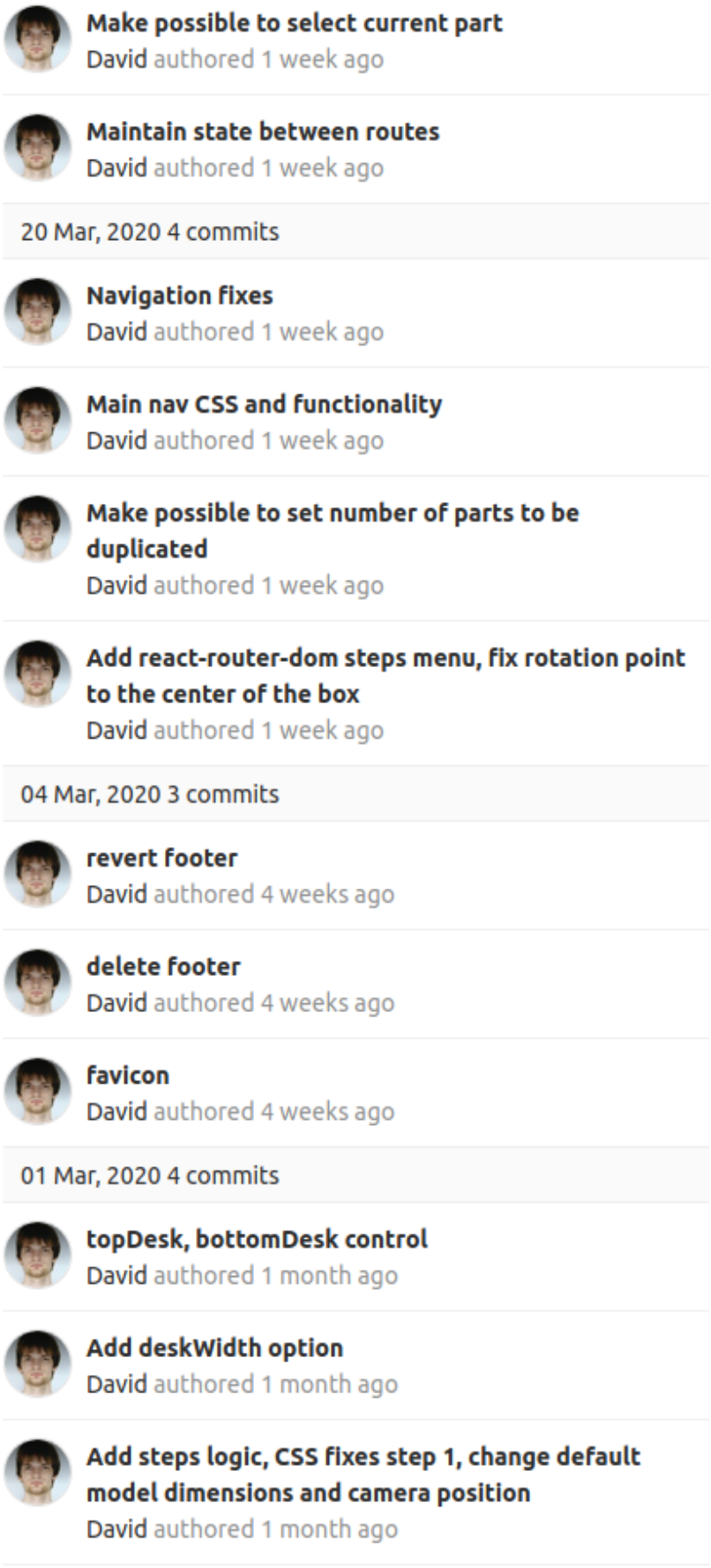
truhlarstvi-luxra.cz	rrgroup.cz	astr.cz	art-style.cz
nabytek-dan.cz	pjatak.cz	lermoplus.cz	nazakazku.eu
rrgroup.cz	indeco.cz	nabytek-na-miru-brno.cz	kuchyne-brno.eu
adam-nabytek.cz	ciganik.cz	hezkydomov.cz	zakazkoveinteriery.cz
siko.cz	kmtruhlarstvi.cz	eurourban.cz	truhlarstvi-zdara.cz
maryskonabytek namiru.cz	sipkadesign.cz	jninterier.cz	pelckuchyne.cz
gilikdesign.cz	interierstudio.eu	nabytekkarel.cz	feldman.cz
kmtruhlarstvi.cz	design-nadto.cz	nabytek-na-miru.eu	qlinterier.cz
ciganik.cz	nabytek-matefi.cz	tninterier.cz	gilikdesign.cz
pjatak.cz	nabytek-kratochvil.cz	nabytekkinta.cz	truhlarstvi.name
atyp-nabytek.cz	tninterier.cz	halbos.cz	kprostor.cz
nabyteknamiru-hk.cz	vyroba-nabytku-kudrna.cz	nabytekdelfi.cz	kuchyne-fokos.cz
		sedaci-soupravy-nabytek.cz	belterra.cz
		simlinterier.cz	truhlarstvi-janzita.cz

Tabulka 9 - Výsledky vyhledávání pro klíčová slova "Nábytek na míru" a "Nábytek na zakázku"

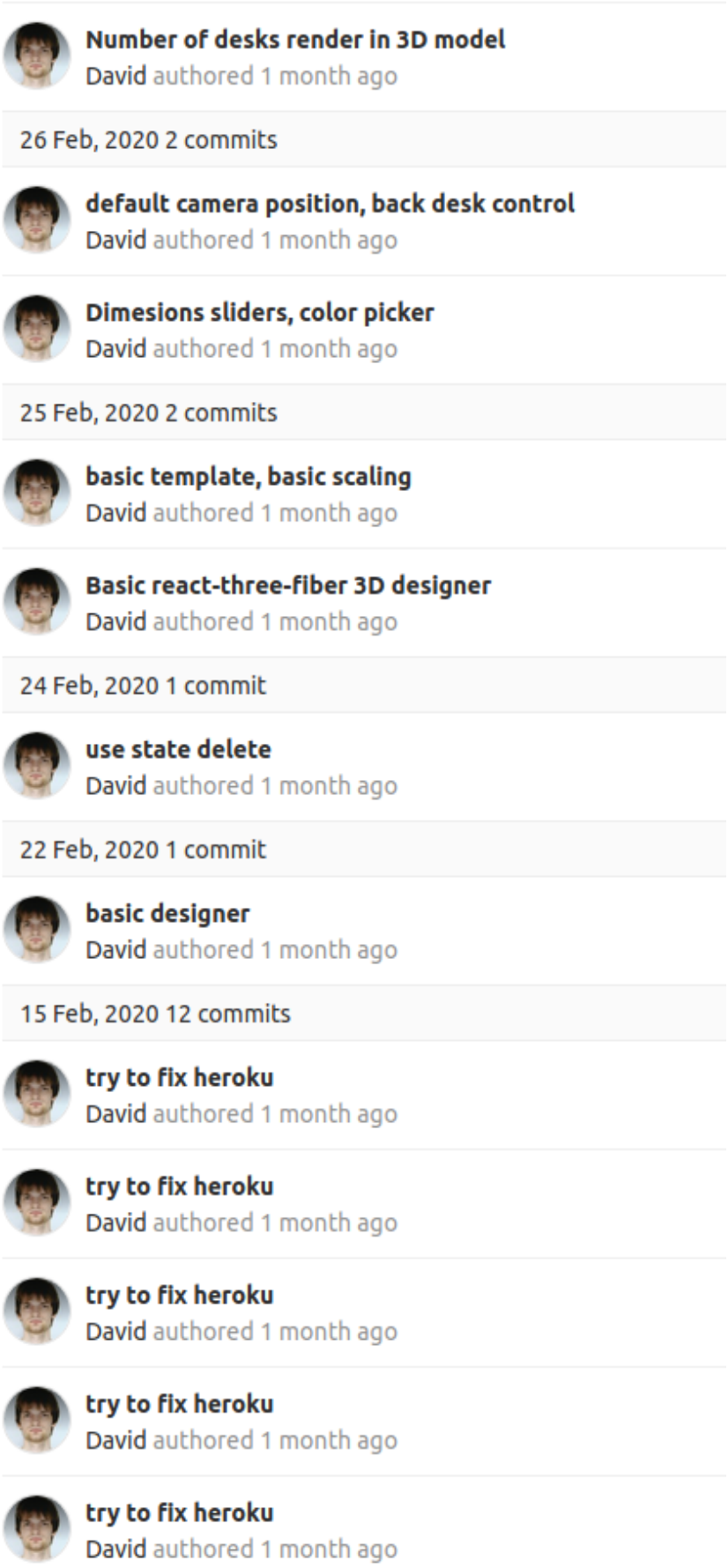
5.2 POPIS POSTUPU PRACÍ – VÝPIS COMMITŮ



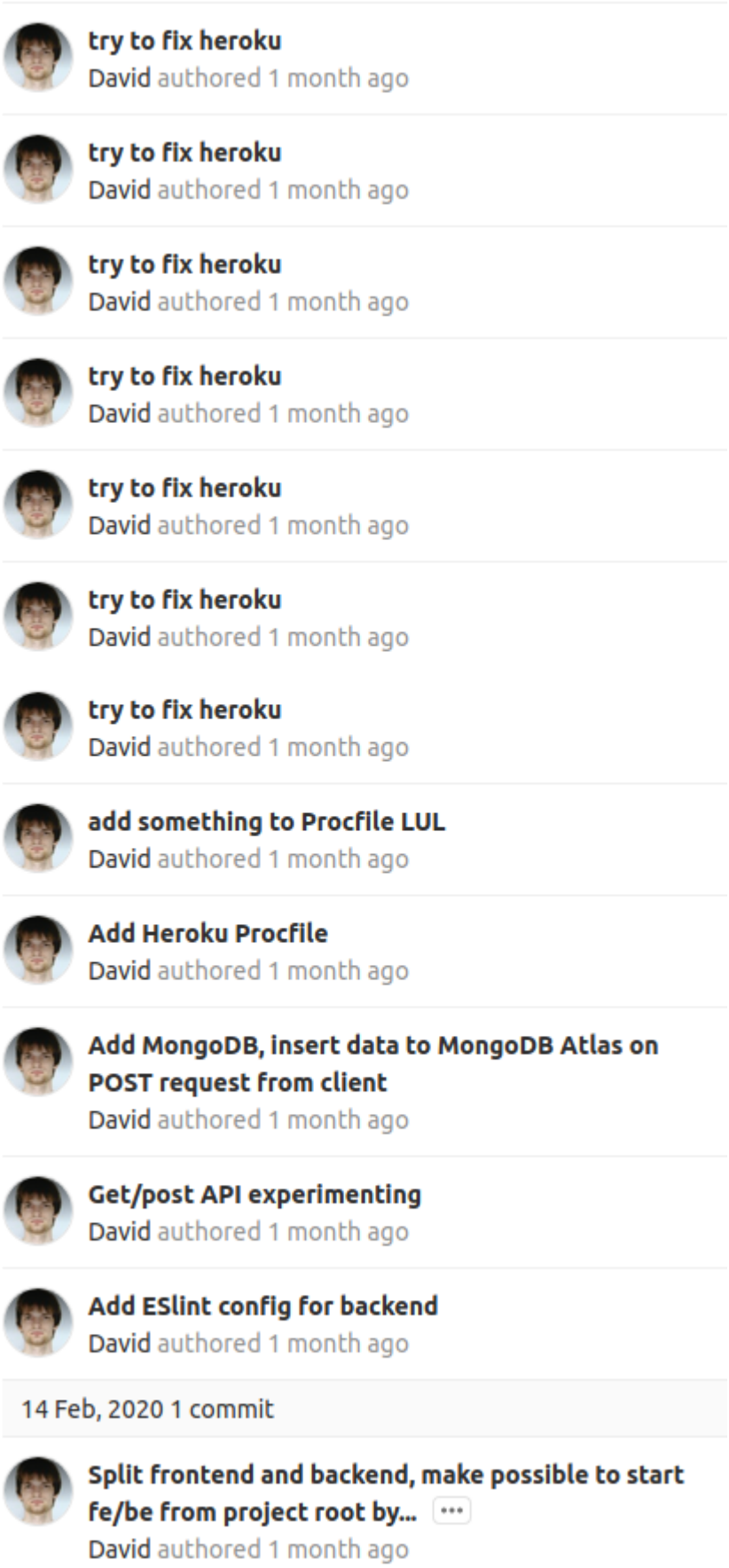
Obrázek 42 - Znázornění postupu prací (commitů) na aplikaci 5/5 (vlastní zpracování)



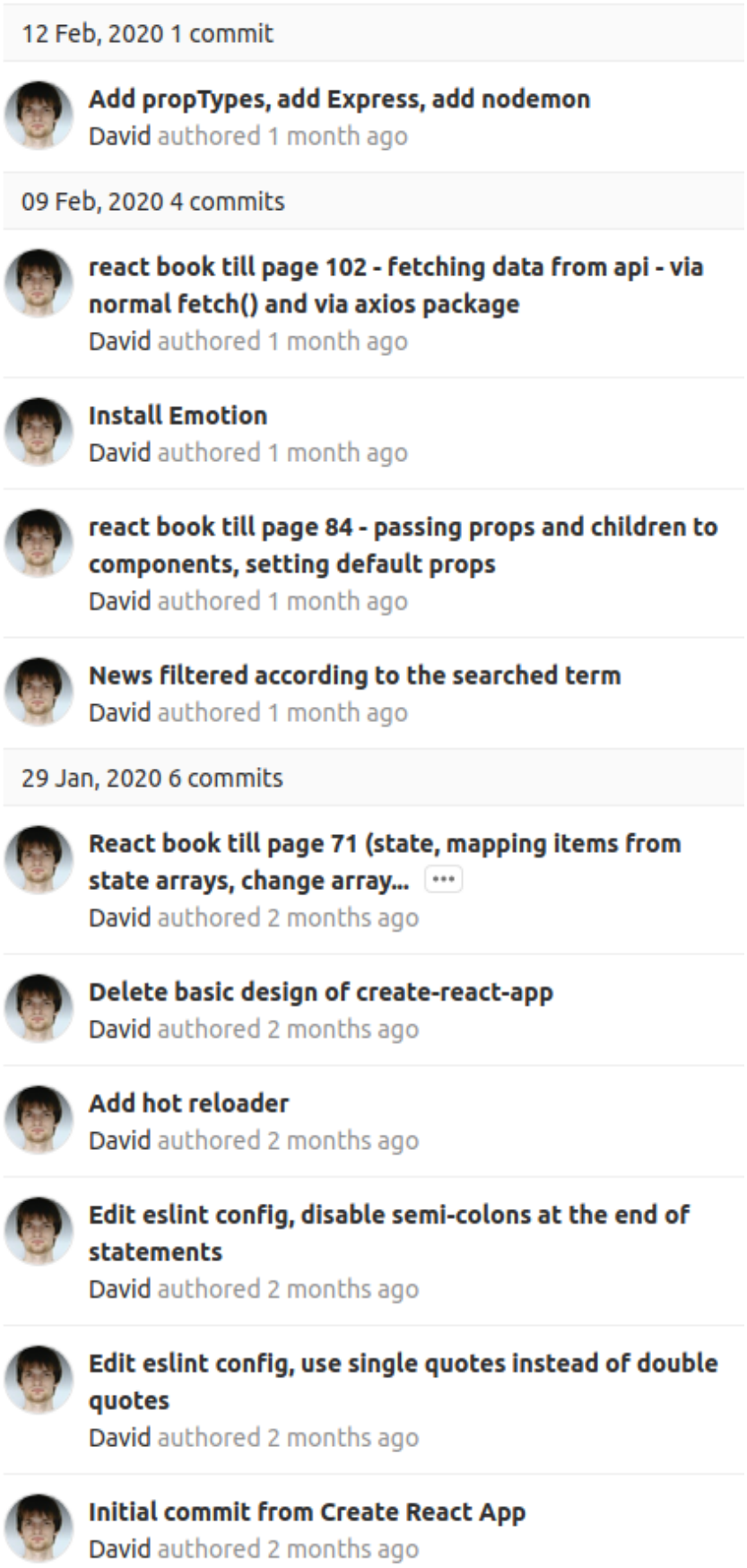
Obrázek 43 - Znázornění postupu prací (commitů) na aplikaci 4/5 (vlastní zpracování)



Obrázek 44 - Znázornění postupu prací (commitů) na aplikaci 3/5 (vlastní zpracování)



Obrázek 45 - Znázornění postupu prací (commitů) na aplikaci 2/5 (vlastní zpracování)



Obrázek 46 - Znázornění postupu prací (commitů) na aplikaci 1/5 (vlastní zpracování)