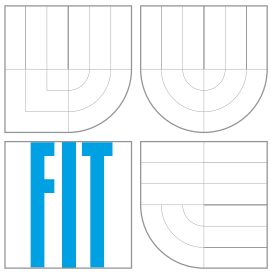
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

WEBOVÝ NÁSTROJ PRO TVORBU GRAFICKÝCH PREZENTACÍ

WEB TOOL FOR GRAPHICAL PRESENTATION DESIGN

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

BORIS KOŠINA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VÍTĚZSLAV BERAN, Ph.D.

BRNO 2016

Abstrakt

Bakalárska práca popisuje proces tvorby jednoduchého grafického editora s podporou produktových katalógov v prostredí webového klienta. Editor umožní pomocou grafických primitív, textov a obrázkov vytvárať grafické prezentácie produktov určené do reklamných kampaní na sociálnych sieťach. V práci je tento proces popísaný od prieskumu vhodných technologií a požiadaviek zadávateľa, až po návrh rozhrania a jeho testovanie. Ďalšie časti sa venujú grafickým primitívam a obrazovým filtrom. Popisujú ich návrh, implementáciu a následné integrovanie do finálneho riešenia editora.

Abstract

This bachelor thesis describes process of creating of simple graphic editor with support of product feeds in web client environment. The purpose of editor is to create graphical presentations of products intended for social media campaigns through the graphical primitives, texts and images . Final thesis gives a detailed description of aforementioned process, where research of appropriate technologies, customer requirements, and last but not least interface design and its testing was studied. Further sections within this thesis deal with graphical primitives and image filters. They describe characterization of design, implementation and further integration was carried out to final solution of editor.

Kľúčové slová

Webová aplikácia, Grafický editor, Užívateľské rozhranie, React, Redux, Canvas

Keywords

Web aplication, Graphic editor, User interface, React, Redux, Canvas

Citácia

KOŠINA, Boris. *Webový nástroj pro tvorbu grafických prezentací.* Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Beran Vítězslav.

Webový nástroj pro tvorbu grafických prezentací

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Vítězslava Berana, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Boris Košina
18. mája 2016

Poďakovanie

Chcel by som poďakovať vedúcemu mojej bakalárskej práce Ing. Vítězslavu Beranovi, Ph.D. za odborné vedenie a rady pri písaní tejto práce.

Obsah

Kapitola 1

Úvod

V súčasnej dobe sa pre veľa ľudí stávajú sociálne siete každodennou realitou. Medzi najrozšírenejšie patrí Facebook s najväčším počtom aktívnych užívateľov každý deň. To predstavuje množstvo potencionálnych zákazníkov pre marketingové oddelenia spoločností, a preto je inzerovanie na sociálnych sieťach stále populárnejšie.

Úspešnej reklame predchádza riešenie dvoch problémov. Prvým je zacielenie kampane na vhodnú skupinu ľudí podľa ich záujmov, veku, jazyka a množstvo iných charakteristík, ktoré si o svojich užívateľoch sociálne siete ukladajú. Druhým problémom je oslovenie tejto skupiny atraktívnou reklamou. Tieto problémy sa snaží riešiť inzertná platforma zadávateľa témy – spoločnosť Roi Hunter, ktorá prepojuje Facebook Ads, Google Analytics a produktové katalógy vo formáte XML, v ktorých sú uchované informácie o produktoch internetových obchodov. Umožňuje tak svojím klientom spravovať množstvo reklám na ich produkty a cieliť ich na zákazníkov.

Cieľom bakalárskej práce je podieľať sa na návrhu a následnej implementácií webovej aplikácie – jednoduchého grafického editora, ktorý doplní funkcionalitu už existujúceho systému o vytváranie graficky prívetivých reklám, určených pre sociálne média Facebook a Instagram. Účelom tejto aplikácie je, aby mohli marketingoví pracovníci z radov klientov vytvoriť šablóny, z ktorých sa budú následne automaticky vytvárať reklamné prezentácie pre tisícky produktov z produktových katalógov, a to bez pokročilých grafických znalostí a bez nutnosti učiť sa používať alebo vlastniť iný grafický editor.

Teoretickej stránke problému sa venuje Kapitola 1, v ktorej sú popísané jednotlivé technológie, potrebné pre tvorbu moderných webových aplikácií a užívateľského rozhrania. Kapitola 2 obsahuje podrobný popis požiadaviek zadávateľa na funkcionalitu aplikácie a prvotný návrh kostry editora, spojený s jej implementáciou pre testovacie účely. Kapitola 3 rozoberá návrh a implementáciu modulov pre obrazové filtre a grafické primitíva použitých vo finálnej verzií editora.

Kapitola 2

Teória

Kapitola obsahuje popis použitia navrhovanej aplikácie a ďalej prehľad teoretických znalostí a technológií, ktoré bolo potrebné naštudovať pre tvorbu moderných webových aplikácií a riešenia zadania práce.

2.1 Štruktúra aplikácie

Nasledujúca časť obsahuje **popis aplikácie** ako celku a **špecifikovanie častí, ktorým sa bude táto práca venovať**.

Systém zadávatela spracúva informácie o produktoch svojich klientov z radov internetových obchodov a prepojuje ich s dátami Google Analytics a Facebook Ads. Toto prepojenie umožňuje presnejšie zacielenie jednotlivých reklamných kampaní a ich automatické vytváranie. S automatickým generovaním reklám však vzniká problém ako klientom umožniť ich personalizáciu podľa svojej firemnej identity, cieľovej skupiny alebo iných parametrov.

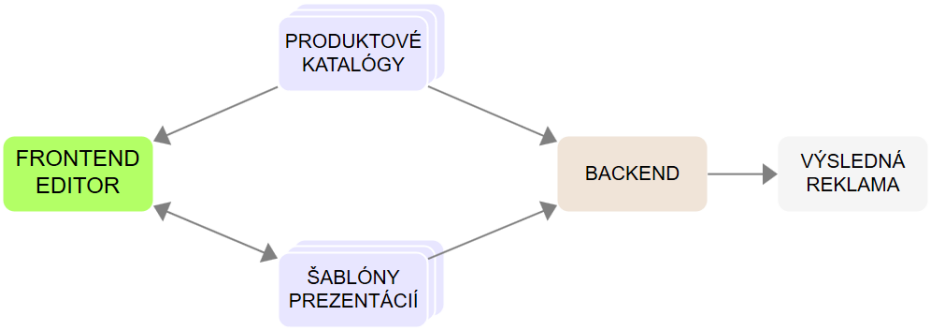
V súčasnosti sa grafické návhry klientov vytváraju v bežne dostupných grafických editoroch (Photoshop, Gimp) a do systému sa vkladajú ručne. Tento spôsob je neefektívny a užívateľský neprívetivý.

Riešením je doplniť existujúci systém o webovú aplikaciu - grafický editor, ktorý umožni vytvárať znovu použiteľné šablóny reklám. Do týchto šablón budú následne v backende systému vložené informácie o jednotlivých produktoch (názov, cena, obrázky, ...).

Obr. 2.1: Spájanie šablón s dátami.

Samotná aplikácia sa teda bude skladať z dvoch častí:

- **Backend** generujúci hotové reklamy (rastrové obrázky) spájaním šablón vytvorených v editore, s dátami produktov (fotky a informácie) z produktových katalógov. Produktové katalógy môžu obsahovať tisícky produktov – výstupom potom môže byť rovnaký počet reklám. Navyše, pred samotným generovaním reklám, je potrebné obrazové dáta analyzovať po stránke technickej kvality (ostrosť, kompresia, rozlíšenie). Tieto operácie bude preto spracovávať backend aplikácie, kde sa využije paralelného spracovania dát.
- **Frontend** grafický editor slúžiaci pre návrh a tvorbu šablón reklamných prezentácií. Podrobne sa mu venuje Kapitola 3.



Obr. 2.2: Štruktúra celej aplikácie.

Predmetom tejto bakalárskej práce je frontend editor. Konkrétne **návrh, pilotnú implementáciu prototypu a testovanie** grafického rozhrania v Kapitole 3. Ďalej sa práca venuje implementácií modulov **obrazových filtrov a grafických primitív** do finálnej aplikácie dodanej zadávatelom v Kapitolách 4 a 5. Keďže budú tieto časti integrované do existujúceho systému, sú kladené určité požiadavky a obmedzenia na použité technológie.

2.2 Produktové katalógy

Pre uchovanie informácií o produktoch sa používajú zoznamy položiek - produktové katalógy (product feeds) vo formáte XML. Jedná sa o zoznamy produktov a ich atribútov, ktoré ho jedinečne identifikujú v rámci jedného súboru. [1] Atribútmi je možné špecifikovať informácie o názve produktu, jeho kategórii, cene, odkazy na fotky a množstvo iných. Jedná sa o formát podporovaný širokou škálou služieb ako Facebook Ads, Google Analytics, služieb porovnávajúcich ceny alebo vyhľadávačov.

```
<?xml version="1.0"?>
<rss xmlns:g="http://base.google.com/ns/1.0" version="2.0">
    <channel>
        <title>Test Store</title>
        <link>http://www.example.com</link>
        <description>An example feed item</description>

        <item>
            <g:id>DB_1</g:id>
            <g:title>Dog Bowl In Blue</g:title>
            <g:description>Plastic Bowl in blue color</g:description>
            <g:link>http://www.example.com/bowls.html</g:link>
            <g:image_link>http://images.example.com/b.png</g:image_link>
            <g:brand>Example</g:brand>
            <g:condition>new</g:condition>
            <g:availability>in stock</g:availability>
            <g:price> 9.99GBP</g:price>
            <g:shipping>
                <g:country>UK</g:country>
                <g:service>Standard</g:service>
                <g:price> 4.95GBP</g:price>
            </g:shipping>

            <g:google_product_category>Pet Supplies</g:google_product_category>
        </item>
    </channel>
</rss>
```

Kód 2.1: Príklad katalógu s jedným produktom.

2.3 Návrhové a architektonické vzory

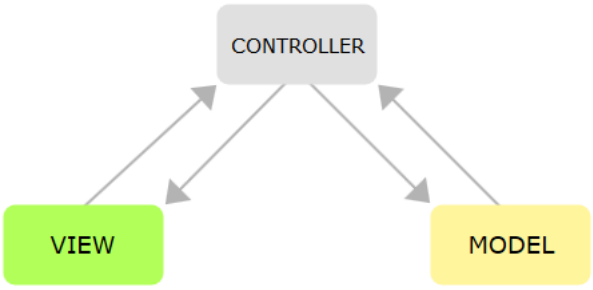
Táto časť rozoberá návrhové vzory a princípy vhodné pre návrh aplikácie v rámci bakalárskej práce.

S rastúcimi požiadavkami na webové aplikácie a ich komplexnosť a s potrebou písať udržiavateľný a znovu použiteľný kód, je dôležité venovať pozornosť návrhovým vzorom. Pod návrhovým vzorom sa rozumie riešenie často opakovaného problému pri návrhu aplikácií.[2]

2.3.1 MVC (Model-View-Controller)

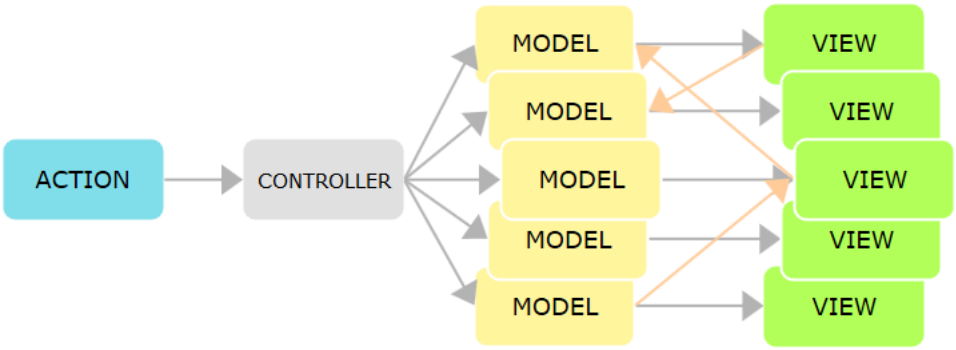
V posledných rokoch patrí architekturálny návrhovy vzor MVC medzi najrozšírenejšie spôsoby návrhu webových aplikácií. MVC vynucuje rozdelenie aplikácie na tri logické časti: Model, View a Controller. [3]

- **Model**
  Spravuje aplikačné dáta a ich logiku, pričom nevie nič o častiach View a Controller. Keď sa Model zmení, notifikuje Controller, ktorý aktualizuje View podľa tejto zmeny.
- **View**
  Je vizuálnou reprezentáciou modelu, ktorá predstavuje súčasný stav a ďalšie prvky užívateľského rozhrania. Vďaka oddeleniu vizuálnej reprezentácie od logiky, môžeme mať viacero View častí pre rovnaké dáta. Tvorí ho HTML, CSS a JavaScript.
- **Controler**
  Predstavuje sprostredkovateľa medzi Model a View. Má za úlohu aktualizáciu View pri zmene dát (Modelu) a tiež pridáva poslúchač udalostí (event listener) pre View, na základe ktorého aktualizuje Model.



Obr. 2.3: Architektúra MVC.

Pri komplexných aplikáciách s veľkým počtom modelov a views naráža táto architektúra na problém s obojstranným tokom dát, kde môže mať jedna zmena kaskádový efekt a šíriť sa celou aplikáciou, čo vedie k nepredvídateľným výsledkom. [4]

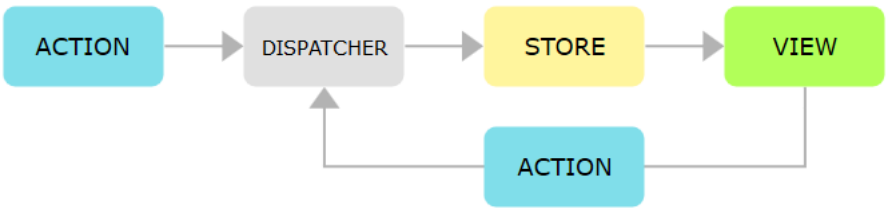


Obr. 2.4: Problémy MVC pri komplexných aplikáciach.

2.3.2 Flux

Tento problém sa snaží riešiť architektúra Flux, ktorá vynucuje použitie jednosmerného toku dát medzi komponentami aplikácie. Flux aplikácie sa skladajú zo štyroch hlavných a nezávislých častí s odlišnými vstupmi a výstupmi: Dispatcher, Store, View a Actions. [5]

- **Dispatcher:** Je to centrálny bod, ktorý spravuje všetky dátové toky aplikácie. Neobsahuje žiadnu logiku aplikácie - ide len o jednoduchý mechanizmus výpočtu nového stavu aplikácie z vyvolaných akcií a predošlého stavu. Pre každú prijatú akciu zavolá príslušné zaregistrované spätné volania (callbacks).
- **Store**
  Obsahuje dáta a logiku aplikácie a jeho úloha by sa dala prirovnať Modelu z architektúry MVC, na rozdiel od ktorého však spravujú stav viacerých objektov danej logickej domény.
- **View**
  Naslúchajú na zmeny zo Stores, na základe ktorých sa vykresľujú. Slúži teda na zobrazovanie dát zo Store a na sprostredkovanie interakcie s užívateľom, po ktorých vyvoláva príslušné akcie. Mohol by sa teda prirovnať ku kombinácií View a Controller z architektúry MVC. Líši sa iba v nemožnosti priamej komunikácie so Store/Model.
- **Actions**
  Sú jednoduché objekty obsahujúce nové dáta a identifikáciu ich typu.



Obr. 2.5: Architektúra Flux

2.3.3 Redux

Myšlienku architektúry Flux ďalej rozvíja a zjednodušuje knižnica Redux, ktorá sa sústredí na tri hlavné princípy:

- Stav celej aplikácie je uložený v jednom Store
- Stav aplikácie je iba na čítanie - jediná cesta ako ho zmeniť je zaslanie akcie
- Akékoľvek zmeny stavu sa vykonávaju čistými (pure) funkciami nazývanými Reducer

Od Flux-u sa Redux líši použitím jediného Store, ktorý uchováva stavový strom celej aplikácie a ktorý je nemenný. Pri vyvolaní akcie - zmene dát, sa vytvorí nový, upravený strom reprezentujúci súčasný stav. Ďalšia odlišnosť je, že pre zmeny dát sa nevyužíva Dispatcher, ale funkcie zvané Reducer. Tieto funkcie musia byť deterministické - pre rovnaký vstup musia vždy vytvoriť rovnaký výstup.

2.4 Návrhové vzory modulárneho JavaScriptu

Pod modulárnym JavaScriptom rozumieme aplikáciu, zloženú z množiny oddelených častí funkčného kódu uchovaného v moduloch. JavaScript však až do poslednej stabilnej verzie ES6 z roku 2015 nemal oproti tradičným jazykom vstavanú podporu pre moduly. Štandardom sa tak stali knižnice vytvorené komunitou. [6]

- **CommonJS**
  Používa synchrónne, teda blokujúce načítanie modulov a je orientovaný hlavne na serverové aplikácie (využívaný v Node.js). Podporuje iba export objektov ako modulov.
- **AMD** (Asynchronous Module Definition)
  Formát AMD pracuje na rozdiel od CommonJS s návrhom asynchrónneho načítavania modulov a závislostí. Využíva sa hlavne v aplikáciach pre webové prehliadače. Okrem objektov podporuje export funkcií, konštruktorov a množstvo iných typov ako modulov.
- **ES6**
  Jednoduchšie riešenie ponúka nový štandard jazyka JavaScript - ECMA Script 6, ktorý pridáva natívnu podporu pre moduly. Integrovanie priamo do jazyka navyše umožnilo použiť kompaktnejšiu syntax a lepšiu podporu pre cyklické závislosti. [7]

```
1  //Export v subore lib.js
2     import sqrt from 'math';
3     export function square(x) {
4        return math.sqrt(x, x);
5     }
6     export function multiple(x, y) {
7        return x * y;
8     }
9
10 //Import zo suboru lib.js
11    import * as lib from 'lib';
12    lib.square(11);
13    lib.multiple(4, 3);
```

Kód 2.2: Import a export funkcie v ES6 notácií.

2.5 Transpiléry

JavaScript je popri ActionScript a JScript a iných jayzkoch iba jednou z mnoha implementácií štandardu ECMAScript. Najnovšou stabilnou verziou je ECMAScript 6 z roku 2015. Súčasné prehliadače však stále nepodporujú všetky jeho funkcie ani vo svojich najnovších verziách. Pri tvorbe webových aplikácií je však potrebné zaistiť, aby aplikácie fungovali správne vo všetkych bežne používaných prehliadačoch. Riešenie ponúkajú transpiléry (source-to-source kompiléry), ktoré preložia ES6 kód do staršieho, plne podporovaného štandardu ES5.

Medzi transpiléry s najširšou podporou patria Babel a Traceur. Použitie najnovšieho štandardu ES6 spolu s JSX syntaxou zatiaľ podporuje iba Babel. [8]

2.6 Knižnice pre užívateľské rozhranie

V tejto časti budú uvedené knižnice pre vytváranie užívateľských prostredí. Ich výber bol limitovaný použitými technológiami v systéme zadávateľa.

React

Ide o JavaScript knižnicu, výlučne pre popis užívateľského rozhrania - v zaužívanej MVC architektúre by predstavovala iba vrstvu "V" – view. [9] Oproti zaužívaným riešeniam (jQuery, Nette) sa líši použitím deklaratívneho paradigmatu – v Reacte nepopisujeme, ako sa majú jednotlivé DOM uzly zmeniť, ale popisujeme iba to, ako ma výsledný DOM model vyzerať. Funguje na princípe virtuálneho modelu DOM, ktorý sa upravuje na základe zmien a následne porovnáva s pôvodným modelom. Podľa ich rozdielov sa vypočítajú najmenšie možné zmeny potrebné pre udržanie aktuálnosti pôvodného modelu. Tento prístup umožňuje podstatné zrýchlenie behu aplikácii a jednoduchšiu implementáciu.

Kód v React môžeme okrem štandardnej JavaScript syntaxe zapisovať pomocou JSX. Ide o zápis syntaxou podobný XML (HTML), čo kód sprehľadňuje, zmenšuje riziko chýb a umožňuje vkladať HTML tagy priamo do JavaScriptového kódu. Použitie zápisu JSX nie je povinné a môže sa použiť klasický Javascript formát, no táto možnosť sa v reálnych aplikáciách pre jeho nevýhody takmer nepoužíva. Nakoľko však JSX nie je valídna JavaScript syntax, musí sa do nej pred spustením kompilovať pomocou transpiléru.

- **Javascript zápis**

```
1  var HelloMessage = React.createClass({
2    displayName: "HelloMessage",
3
4    render: function render() {
5      return React.createElement(
6        "div",
7        null,
8        "Hello ",
9        this.props.name
10     );
11   }
12 });
13
14 ReactDOM.render(React.createElement(HelloMessage, { name: "John" }), mountNode);
```

Kód 2.3: Hello World v JavaScript zápise. [9]

- **JSX zápis**

```
var HelloMessage = React.createClass({
  render: function() {
    return <div>Hello {this.props.name}</div>;
  }
});

ReactDOM.render(<HelloMessage name="John" />, mountNode);
```

Kód 2.4: Hello World v JSX zápise. [9]

Bootstrap

Bootstrap je HTML, CSS a čiastočne JavaScript framework pod licenciou MIT, ktorý umožňuje jednoduchší návrh responzívneho layoutu webových stránok. Nepopisuje ako by mal layout vyzerať, ale poskytuje množstvo predpripravených formulárov, tlačítok, modálnych okien a iných komponentov užívateľského rozhrania pre urýchlenie vývoja a zjednotenie postupov medzi vývojarmi.

2.7 Canvas

Je to element zo špecifikácie HTML5, ktorý umožnuje vykresľovanie 2D grafických prvkov pomocou JavaScriptu. Predstavuje rastrové plátno, na ktoré je možné pomocou nízkoúrovňového API vykresľovať grafické primitíva, obrázky, objekty, text a jednoduché animácie. [10]

```
ctx.arc(100, 100, 70, 0, 2*Math.PI);
ctx.fillStyle = "yellow";
ctx.fill();
ctx.arc(100, 100, 40, 0, 2*Math.PI);
ctx.fillStyle = "white";
ctx.fill();
```

Kód 2.5: Vykreslenie prstenca na Canvas.

Nevýhodou je nízkoúrovňové API podporujúce iba základné funkcie pre prácu s grafickými objektami. Preto je v reálnych aplikáciách vhodné použiť JavaScript knižnice, ktoré zapuzdrujú toto API do objektového modelu.

- **Fabric.js**
  Open-source projekt pod licenciou MIT. Medzi hlavné výhody patrí podpora dotykových zariadení, HW akcelerácia, serializácia a de-serializácia canvasu s podporou formátov JSON a SVG, vďaka čomu možno výsledný canvas uložiť pre ďalšie spracovanie.

- **Konva**
  Knižnica s takmer identickou funkcionalitou ako Fabric.js a licenciou MIT. Rozdielom je vylepšená práca s vrstvami a objektami. Umožňuje naviac vyhľadanie objektu podľa jeho druhu alebo podľa identifikátora. Nevýhodou je problematická podpora SVG a serializácia iba do JSON.

```
new Konva.Ring({x: 100, y: 100, innerRadius: 40, outerRadius: 70, fill: 'yellow',});
```

Kód 2.6: Vykreslenie rovnakého prstenca pomocou knižnice Konva.

2.8 Behové prostredie

Pri práci s vytvorenými šablónami vo webovom prehliadači (front-end) a ich následným spracovaním na serveroch (back-end) je potrebné zaručiť rovnaké výsledky v oboch prostrediach - rozloženie grafických elementov, formát písma, spracovanie obrázkov a iné.

Riešením je vývoj aplikácie v behovom prostredí, ktoré umožní spustenie JavaScript kódu mimo prehliadač, vďaka čomu je možné použiť rovnaký kód v oboch prostrediach. To zaručí rovnaký výsledok všetkých algoritmov a urýchli sa vývoj vďaka znovupoužiteľnosti kódu a možnosti používať rovnaké knižnice.

Node.js

Je to udalosťami riadené behové prostredie postavené na JavaScript interpréte V8 z prehliadača Google Chrome. Je možné ho použiť ako interprét pre lokálne aplikácie a zároveň môže obsluhovať požiadavky na serveri.

- **Rýchlosť**
  Interpret V8 od spoločnosti Google prekladá JavaScript pred jeho spustením do strojového kódu, čo má za výsledok veľmi rýchle spracovanie.

  Ďalej sú v Node.js použité udalosťami riadené vstupno-výstupne operácie. To znamená, že typicky blokujúce volania ako čakanie na databázu, sú spracované asynchrónne a tak neblokujú ďalší kód. Keď vstupno-výstupná operácia skončí, zavolá sa spätné volanie funkcie s výsledkom. [11]

- **npm**
  Súčasťou Node.js je správca balíčkov `npm` , ktorý predstavuje jednotné riešenie pre zdielanie kódu v Javascript ekosystéme. Jedným príkazom je možne nainštalovať a začať používať všetky potrebné knižnice (React, Bootstrap, Konva ...). V roku 2016 presiahol počet balíkov 250 000. [12]

Kapitola 3

Návrh rozhrania editora

Kapitola popisuje **postup návrhu webového nástroja**, ktorého ciele boli popísane v časti 2.1. Najprv sa venuje analýze problému, na základe ktorej je ďalej navrhnutá požadovaná funkcionalita a rozhranie aplikácie. Potom nasleduje tvorba pilotnej implementácie a jej testovanie. Tento proces vychádza z odporúčaní kapitoly 2 v [13] a bol pravidelne konzultovaný so zadávateľom a upravovaný na základe jeho požiadavok.

3.1 Analýza problému

Pre návrh rozhrania editora je najprv potrebné špecifikovať jeho účel a konkrétne požiadavky na funkčnosť a implementáciu.

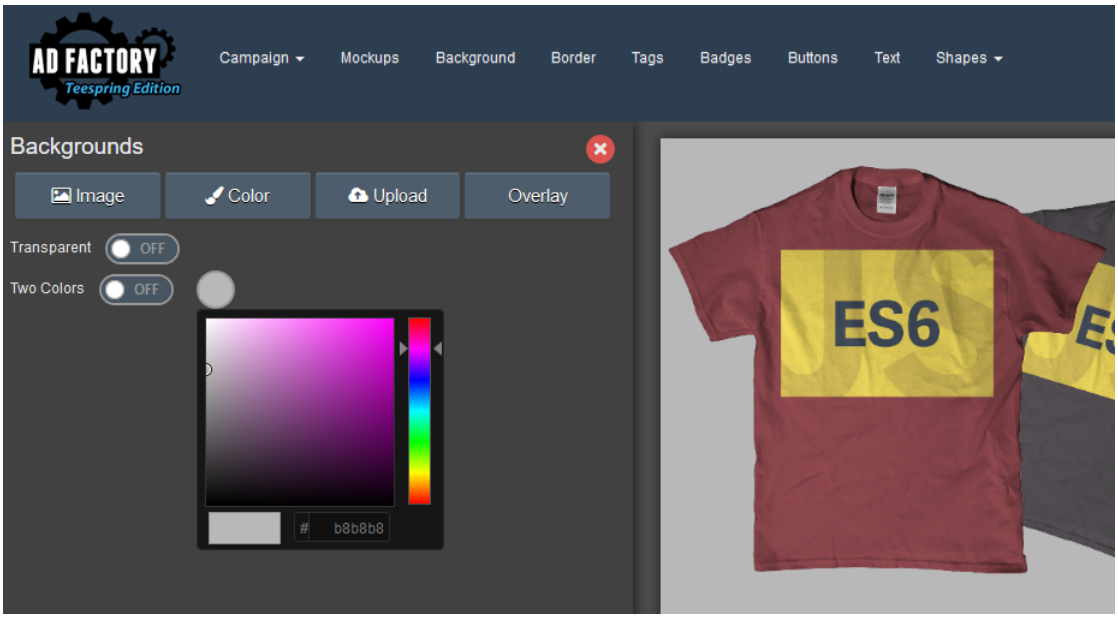
Ako už bolo spomenuté v teoretickej časti, editor bude slúžiť iba na vytváranie šablón, ktoré následne backend aplikácie spojí s dátami produktových katalógov a vytvorí tak grafické prezentácie pre každý z produktov. Napriek tomu, že generovanie prezentácií zabezpečí backend, musí navrhovaná aplikácia vedieť pracovať s produktovými katalógmi. Vytvorené šablóny sa totiž budú aplikovať na celý vybraný produktový katalóg, vrátane dynamického načítania textových (názov produktu, cena ...) a grafických (1 až N obrázkov produktu) prvkov. Užívatelia tak musia mať pri navrhovaní šablón možnosť pracovať s týmito dátami.

Podobné riešenia

Momentálne žiadny z Facebook marketing partnerov nedisponuje generátorom reklám s podporou produktových katalógov – dynamického načítavania cien, obrázkov a iných informácií o produktoch. Existuje však množstvo editorov, v ktorých je možné inšpirovať sa kvalitným užívateľským rozhraním pre frontend časť aplikácie.

- **Photoshop, Gimp**
  Množstvo potencionálnych užívateľov aplikácie používa k tvorbe svojích reklám známe grafické editory. Chýba im, ale akékoľvek prepojenie so systémom zadávateľa a podpora produktových katalógov. Preto je vhodné držať sa zaužívaných princípov z týchto aplikácií, ako je práca s vrstvami a objektami, rozloženie prvkov editora a iné, aby neboli klienti nútení zaúčať svojich zamestnancov pre prácu s ďalšou aplikáciou.

- **Ad Factory**
  Jedná sa o editor s podobným zameraním ako požaduje zadávateľ a rovnako slúži na tvorbu reklám pre sociálne média s podporu produktových katalógov. Nevýhodou je špecializácia iba na predaj oblečenia a neprehľadné užívateľské rozhranie.



Obr. 3.1: Rozhranie editora Ad Factory

- **Canva**
  Webová platforma pre grafické návrhy s množstvom funkcií a prepracovaným rozhraním, ktoré bolo inšpiráciou pri návrhu. Obsahuje veľké množstvo predpripravených tvarov, textov, ale aj hotových návrhov pripravených pre reklamné použitie na sociálnych sieťach. Veľká časť funkcií je však spoplatnená. Ďalším nedostatkom je chýbajúca podpora produktových katalógov.

Cieľová skupina

Návrh aplikácie musí reflektovať požiadavky a schopnosti cieľovej skupiny. Editor budú používať klienti spoločnosti – internetoví obchodníci, ktorí chcú prezentovať svoje produkty na sociálnych sieťach. Typických používateľov môžeme rozdeliť do dvoch kategórií:

- Najpočetnejšiu skupinu tvoria marketingoví pracovníci, ktorí maju znalosti z oblasti predaja a reklamy, no ich technické zručnosti a grafické cítenie môže byť limitované, a preto je potrebné zachovať čo najjednoduchšie a intuitívne užívateľské rozhranie.
- Malú časť tvoria grafici, pod vedením marketingových pracovníkov. Pre tento prípad musí aplikácia disponovať dostatočným množstvom funkcií, ktoré umožnia previesť ich predstavy do skutočnosti a musí poskytovať postupy známe z iných grafických editorov.

3.2 Návrh funkčnosti

Nasledujúcim krokom bolo identifikovanie funkcionality, ktorú by mal editor poskytovať, aby v ňom bolo možné vytvárať požadované návrhy šablón.

Príklad použitia

Pre lepšiu predstavu fungovania nasleduje príklad typického použitia.

1. Užívateľ si v editore vyberie produktový katalóg, s ktorým bude ďalej pracovať
2. Vytvorí základný návrh šablóny – vizuálne rozdelenie, nastavenie farieb ...
3. Umiestni do editora statické prvky ktoré budú rovnaké v každej vygenerovanej reklame – logo firmy, textový popis (Dnes zľava:)
4. Za textový popis vloží dynamický text – premennú z produktového katalógu s aktuálnou zľavou
5. Vloží do návrhu dynamický obrázok produktu z katalógu
6. Uloží vytvorenú šablónu
7. V systéme zadávateľa pri vytváraní reklamnej kampane pre Facebook zvolí vytvorenú šablónu a produktový katalóg, na ktorý sa má aplikovať.
8. Backend podľa vytvorenej šablóny vygeneruje reklamu pre každý produkt zo zvoleného produktového katalógu
9. Výsledkom je N rastrových obrázkov, ktoré budu použité v reklamnej kampani (N = počet produktov vo zvolenom produktovom katalógu)

Špecifikácia požadovyných funkcií

Podrobný popis funkcií, ktoré by mala aplikácia zvládať.

- **Grafické primitíva**
  Tvorba základných geometrických útvarov – kocka, obdĺžnik, kruh, elipsa. Ďalej tvorba zložitejších tvarov – hviezda, n-uholníky a iné. Užívateľ by mal mať na výber z veľkého množstva elementárnych tvarov, z ktorých bude môcť vytvárať zložitejšie návrhy.
- **Statické a dynamické texty**
  Možnosť vytvárať texty, ktoré užívateľ zadá ručne (statické) a možnosť pridať texty premenných z produktového katalógu (dynamické). Napríklad premenná `g:price`, za ktorú backend pri spracovaní dosadí cenu produktu. Pri oboch typoch textov bude možnosť nastaviť ich veľkosť, farbu, pozíciu, zarovnanie, typ písma.
- **Statické a dynamické obrázky**
  Umožniť nahrávanie vlastných obrázkov z počítača alebo URL – statické obrázky (logo firmy a iné), ktoré budú po generovaní v každej výslednej reklame rovnaké. Ďalej možnosť vložit fotku produktu z produktového katalógu – dynamické obrázky, ktoré budú v každej vygenerovanej reklame jedinečné. Užívateľ bude môcť obrázkom meniť ich umiestnenie a veľkosť, aplikovať obrazové filtre a orezávať ich.

- **Práca s vrstvami**
  Všetky objekty (tvary, texty, obrázky) budú reprezentované vrstvami, ktorým bude možné nastavovať ich viditeľnosť a poradie (usporiadanie na ose Z), duplikovať ich a mazať.
- **Výber produktového katalógu**
  Užívateľ bude môcť zvoliť produktový katalóg a konkrétny produkt z neho, s ktorým chce pracovať.

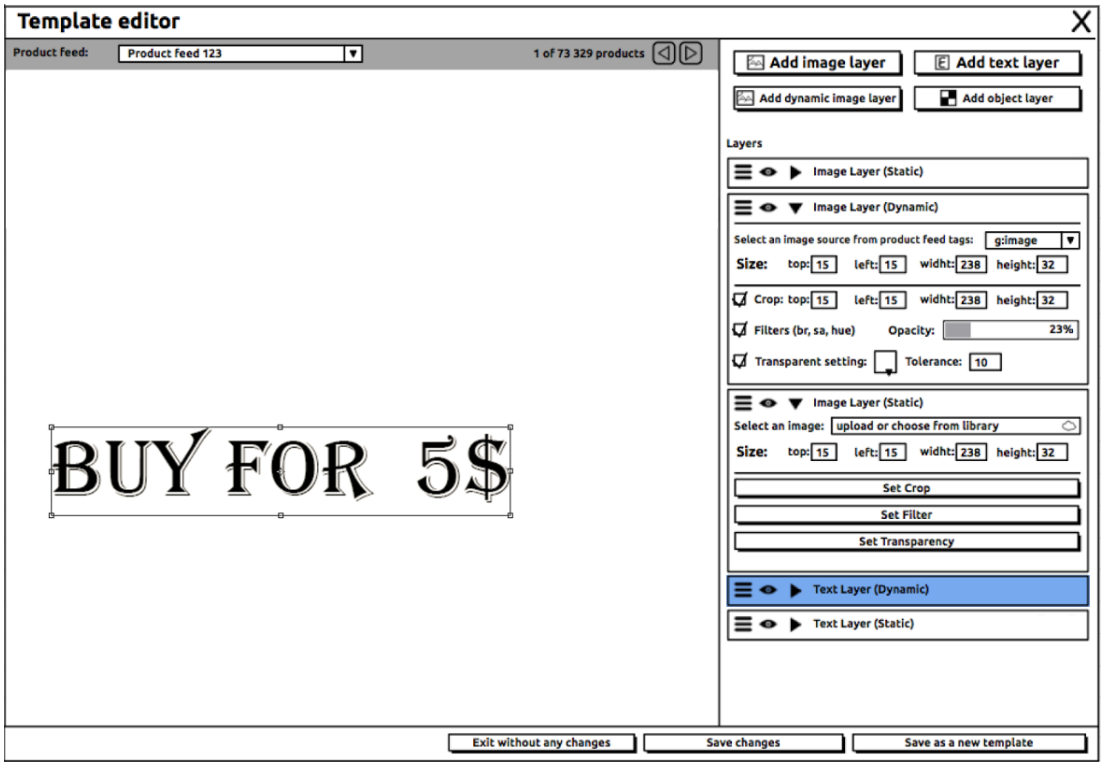
Špecifikácia požiadavkov na implementáciu

- Použitie JavaScript knižnice pre Canvas
- Serializácia návrhu šablóny
- Použitie open-source nástrojov a otvorených formátov

3.3 Návrh užívateľského rozhrania – 1. iterácia

Aplikácia editora bude mať iba jednu obrazovku, s ktorou bude užívateľ pracovať. Napriek tomu je dôležité venovať dostatok času procesu vytvárania užívateľského rozhrania. Aplikáciu bude totiž po nasadení do systému zadávateľa používať množstvo súčasných klientov, ktorí očakávajú jednoduché a intuitívne ovládanie.

Na základe vyššie uvedených požiadaviek, bol v spolupráci so zadávateľom vytvorený prvý návrh možnej podoby aplikácie. Tento návrh je zobrazený vo forme wireframu na obrázku 3.2.



Obr. 3.2: Prvý návrh možnej podoby rozhrania.

Rozhranie editora bolo rozdelené do troch častí. Užívatelia sú zvyknutý príjmať informácie od hora dole a zľava doprava a v tomto poradí boli usporiadané aj prvky editora po zohľadnení ich významnosti a následnosti ich používania.

Výber produktového katalógu

Výber konkrétneho katalógu, s ktorým bude užívateľ pracovať je často prvým krokom pri tvorbe alebo úprave existujúcej šablóny. Preto bol umiestnený priamo nad pracovnú plochu spolu s tlačítkami pre posun po produktoch vo zvolenom katalógu.

Pracovná plocha

Najväčšiu časť tvorí pracovná plocha, kde bude sústredená všetká práca užívateľov. Obsahuje plátno kde budú umiestňované všetky grafické elementy, ktoré budú tvoriť návrhy šablón. Kedže ide o najhlavnejšiu komponentu editora, bola umiestnená do ľavej až strednej časti rozhrania. Pri označení objektu sa okolo neho zobrazia ovládacie prvky pre zmenu veľkosti, polohy a natočenia aby boli dodržané konvencie z bežných grafických editorov.

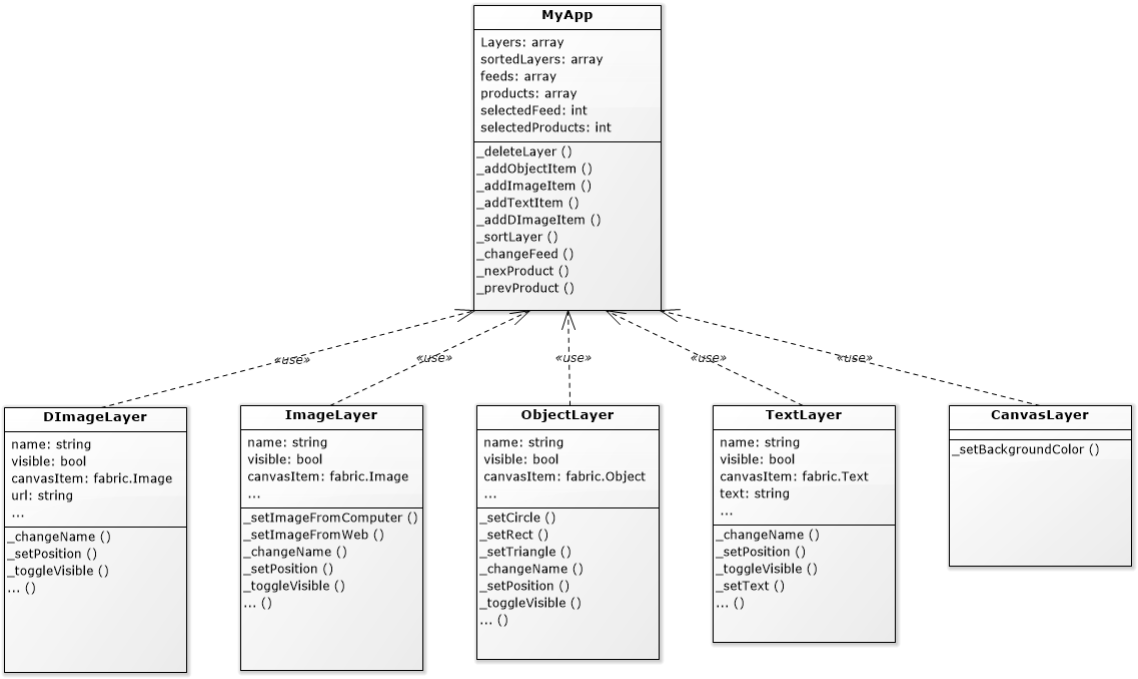
Panel vrstiev

Panel vrstiev bude slúžiť pre vytváranie a úpravy vrstiev objektov umiestňovaných na pracovnú plochu. Do hornej časti boli umiestnené tlačítka pre pridanie vrstiev textu, objektu a dve tlačítka pre pridanie dynamického alebo statického obrázku. Po kliknutí na tieto tlačítka sa príslušny objekt vykreslí na plátno a do panelu sa pridá vrstva, ktorá ho reprezentuje. Tá bude obsahovať nastavenia špecifické pre daný typ objektu a ovládacie prvky vrstvy (odstránenie, zmena poradia).

3.4 Pilotná implementácia

Ďalším krokom po dokončení návrhu, bolo vytvorenie pilotnej implementácie, ktorá slúžila na testovanie grafického rozhrania, overenie navrhnutej funkcionality zadávateľom a v neposlednom rade overenie možností navrhnutých knižníc pre prácu s elementom Canvas. Kedže išlo o prototyp, nebola implementovaná všetka funkcionalita a návrh sa sústredil hlavne na overenie vyššie popísanych problémov.

Prototyp bol implementovaný za pomoci HTML5, CSS3, JavaScript a doplnkových knižníc Bootstrap a React. Pre prácu s elementom Canvas bola zvolená knižnica Fabric.js, pretože ponúka vačšie množstvo hotových riešení.



Obr. 3.3: Komponenty React v pilotnej implementácií.

- **MyApp**
  Je hlavnou komponentou, ktorá slúži na vykreslenie rozhrania editora. Ďalej zabezpečuje pripojenie knižnice Fabric.js na vytvorený Canvas po jeho vykreslení, zabezpečuje správu vrstiev a simuluje produktové katalógy pre potreby testovania.

- **ImageLayer**
  Umožňuje načítanie obrázku z počítača alebo URL, jeho umiestnenie na pracovnú plochu editora a následnú prácu s ním - aplikacia filtrov, zmena veľkosti, polohy, natočenia...

- **DImageLayer**
  Reprezentuje dynamické obrázky – ich načítanie zo simulovaných produktových katalógov a ďalej poskytuje rovnakú funkcionalitu ako komponenta statických obrázkov.

- **ObjectLayer**
  Umožňuje pridávanie grafických primitív na pracovnú plochu editora. Pre základné tvary ako kocka, kruh, elipsa a iné, používa objekty z knižnice Fabric.js. Zložitejšie tvary sú definované pomocou SVG. Všetkým tvarom ďalej umožňuje základné transformácie a zmenu farieb prípadne orámovania.

- **TextLayer**
  Umožňuje pridávanie a prácu s textovými objektami ako predošlé komponenty.

- **CanvasLayer**
  Slúži iba na nastavenie vlastností elementu Canvas.

Zhrnutie

Pri tejto implementácií bolo identifikovaných niekoľko ďalších problémov, ktoré bolo nutné vyriešiť.

- Pri serializácií je potrebné jednotlivým vrstvám pridávať jedinečné ID kvôli ich identifikácií na backende a ďalšie dodatočné informácie. Knižnica Fabric.js však neumožňuje jednoduché pridávanie týchto informácií do objektov.
- Knižnica Fabric.js nepodporuje modifikáciu ovládacích prvkov. Z týchto dôvodov bolo ďalej uprednostnené použitie knižnice Konva.js. Je v nej potrebné doimplementovať viacero funkcií ako pri použití Fabric.js, no ponúka lepšiu prispôsobitelnosť.
- Implementácia obrazových filtrov v knižniciach pre prácu s elementom Canvas je blokujúca – počas aplikovania filtra sa rozhranie editora zasekne a užívateľ musí čakať kým sa výpočet ukončí. Pri výpočetne náročnejších filtroch to môže byť niekoľko desiatok sekúnd a preto je potrebné tieto filtre spracovávať v samostatnom module.
- Pri použití tvarov vo formáte SVG by mohlo dôsť k problémom s ich vykreslením na strane servera. To by malo za následok odchýlky medzi zobrazením v prehliadači a výslednou vygenerovanou reklamou. Riešenie ponúka možnosť definovania vlastných tvarov v knižnice Konva.js a funkcii elementu Canvas.
- Pri pridávaní obrázkov z URL môže nastať problém s mechanizmom CORS. Ide o mechanizmus zdieľania zdrojov webových stránok (obrázky, kód ...) pre aplikácie na inej doméne. Pokiaľ importujeme obrázok zo servera, ktorý tento mechanizmus nepodporuje, uzamkne funkcie Canvasu (serializácia, úprava obrazkov a iné). Problém bol vyriešený vytvorením PHP skriptu, ktorému sa predá adresa požadovaného obrázka. Skript obrázok stiahne a pošle ako odpoveď editoru. Týmto sa mechanizmus CORS obíde, pretože aplikácia editora a PHP skript bežia na rovnakej doméne.

```php
<?php
  if(($imgType = exif_imagetype($_GET["url"])) != false) {
    if(($data = file_get_contents($_GET["url"])) !== false){

      if ($imgType == IMAGETYPE_GIF) {
        header('Content-type: image/gif');
        echo $data;
      }
      elseif ($imgType == IMAGETYPE_JPEG) {
        header('Content-type: image/jpeg');
        echo $data;
      }
      elseif ($imgType == IMAGETYPE_PNG) {
        header('Content-type: image/png');
        echo $data;
      }
    }
  }
?>
```

Kód 3.1: PHP skript pre vkladanie obrázkov.

3.5 Testovanie

Pre vyhodnotenie správnosti návrhu a odhalenie prípadných problémov, bola implementácia podrobená uživateľskému testovaniu. Cieľom bolo zistiť, či je navrhnuté rozhranie dostatočne jednoduché a použiteľné aj pre ľudí bez skúseností s grafickými editormi. Ďalšou úlohou testovania bolo získať od užívateľov spätnú väzbu. Konkrétne, aké funkcie by v editore privítali a naopak, ktoré využívať nechcú. Najefektívnejšia cesta k týmto informáciam je pozorovanie užívateľov pri práci s aplikáciou. [14]

Priebeh testovania

Pre testovanie bolo vybraných päť užívateľov, ktorí sa vo svojom študijnom alebo profesijnom živote stretávajú s oblasťami marketingu a sú teda potencionálnymi užívateľmi aplikácie aj v reálnom svete. Toto číslo je považované za optimálny počet testovacích subjektov pre kvalitatívne testovanie. [15]

Užívateľské testovanie bolo rozdelené do dvoch častí.

1. časť

V prvej časti boli užívatelia stručne oboznámení s problematikou, ktorú sa editor snaži riešiť a s produktovými katalógmi. Následne dostali za úlohu, predstaviť si, že sú marketingový pracovníci a majú vytvoriť prezentáciu ľubovoľne zvoleného produktu, určenú pre sociálne siete, čím sa oboznámili s rozhranim editora. Pre získanie kvalitatívnych výsledkov testovania boli užívatelia požiadaní aby o svojich krokoch premýšľali nahlas a bolo pozorované, na aké problémy pri práci s aplikáciou narážajú.

Potom boli požiadaní o zodpovedanie dvoch otázok:

1. Aké funkcie Vám v editore chýbajú?
2. Ktoré funkcie sú Vám nejasné alebo neprehľadné?

2. časť

V druhej časti mali užívatelia vytvoriť prezentáciu produktu podľa pripravenej predlohy. Počas ich práce bol zaznamenávaný počet chýb a čas potrebný pre dokončenie úlohy. Výsledky týchto testov boli iba orientačné nakoľko pre vyššiu presnosť kvantitatívnych testov je potrebné zahrnúť do testovania viac ako päť testovacích subjektov.

Interpretácia výsledkov

Z pozorovania a komentárov užívateľov v prvej časti testovania boli vyhodnotené nasledovné závery:

- Užívateľom robili problémy dve oddelené tlačítka na pridávanie obrázkov – statických a dynamických
- Užívatelia strácali prehľad o vytvorených vrstvách
- Užívatelia sa ťažko orientovali v paneli vrstiev, ktorý obsahuje množstvo informácií a ovládacich prvkov

Spätná väzba

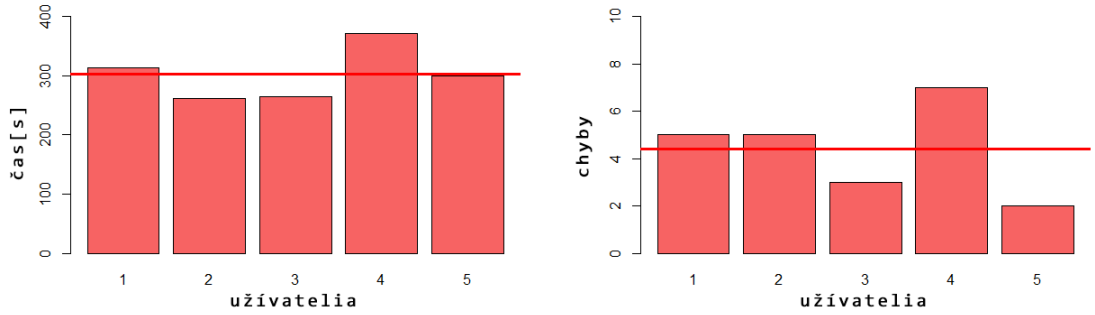
Nápady užívateľov na nové funkcie:

- Pozícovanie objektov pomocou klávesnice.
- Predpripravené hotové riešenia.
- Pokročilejšie úpravy textu.

Odpovede na druhú položenú otázku boli nasledovné:

- Štyria užívatelia označili prácu s vrstvami ako neprehľadnú.
- Traja ľudia nerozumeli nastaveniam pri efekte odstránenia bieleho pozadia.

Výsledky merania v druhej časti testovania boli nasledovné:

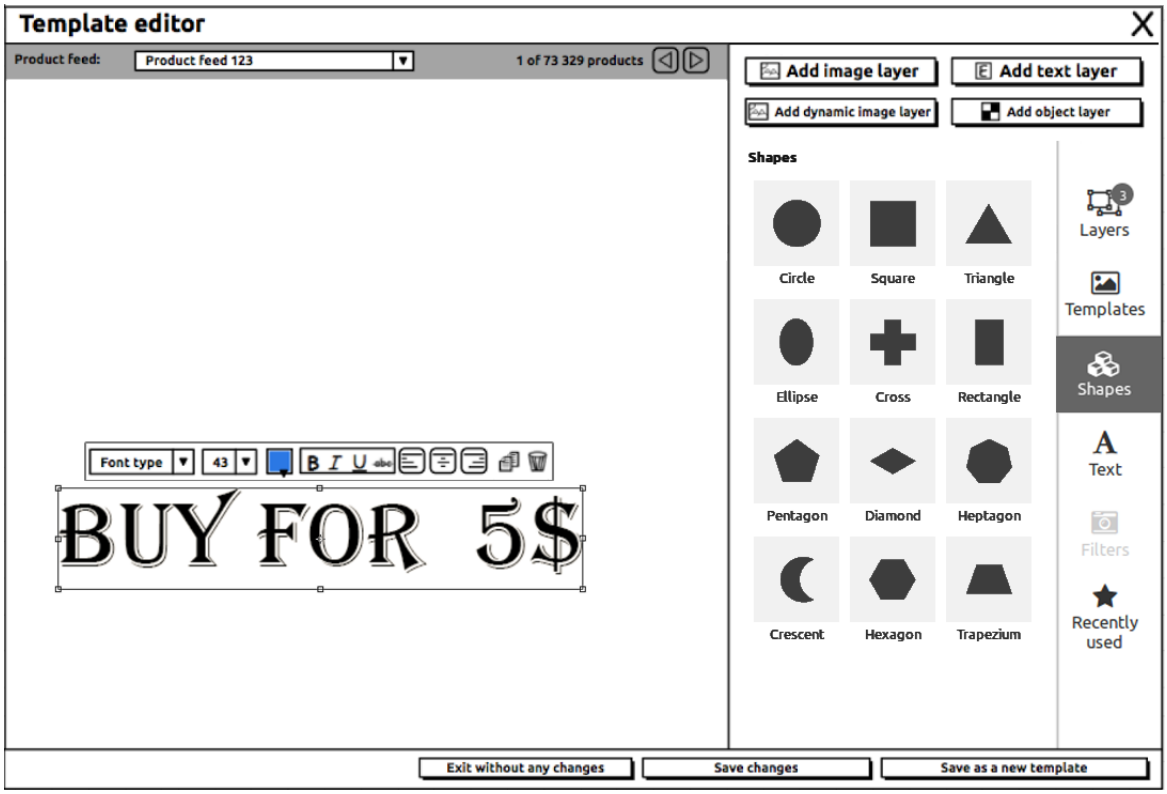


Obr. 3.4: Čas a počet chýb pri druhej časti testu.

3.6 Návrh užívateľského rozhrania – 2. iterácia

Po vyhodnotení prvých testov bolo potrebné návrh zjednodušiť a viac sa tak priblížiť požiadavkám užívateľov. V spolupráci so zadávateľom bolo navrhnuté presunúť ovládacie prvky do kontextového menu zobrazovaného nad práve editovaným objektom a uvoľnený priestor v pravej časti editora využiť pre knižnice predpripravených objektov.

Ďalej táto úprava elegantne vyriešila požiadavok do budúcna – prehliadanie uložených šablón priamo v rozhraní editora.



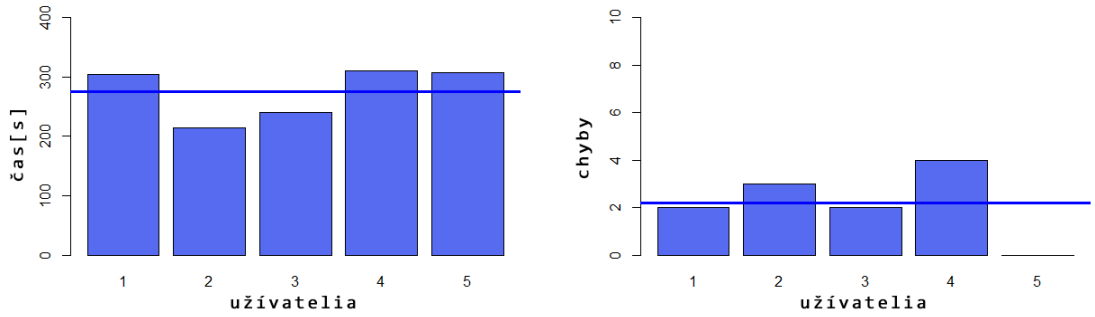
Obr. 3.5: Upravený návrh podoby rozhrania.

Rozmiestnenie prvkov editora zostalo rozdelené rovnako do troch častí. Zmenil sa však význam panelu vrstiev, ktorý bol rozdelený na niekoľko prepínateľných knižníc obsahujúcich geometrické tvary, štýly textov a do budúcna množstvo iných predpripravených objektov ako ikony, pozadia a iné. Naďalej obsahuje aj jednotlivé vrstvy, avšak umožnuje iba ich správu (mazanie, zmena poradia). Všetky nastavenia jednotlivých objektov sú zobrazované priamo nad konkrétnymi objektami na plátne. Toto usporiadanie zjednodušilo rozhranie editora, na ktorom prebehli ďalšie testy.

3.7 Testovanie

Nad druhým, upraveným návrhom bola prevedená rovnaká sada testov, s rovnakými užívateľmi ako pri prvej iterácií. Z pozorovaní užívateľov počas prvej časti testu bolo zistené, že v upravenom rozhraní neboli užívatelia tak nachýlní k chybám ako v prvej iterácií. Užívatelia popisovali rozhranie ako prehľadnejšie a "čistejšie".

Menšiu chybovosť potvrdzujú aj namerané výsledky z druhej časti testu zobrazené v nasledovnom grafe:



Obr. 3.6: Čas a počet chýb pri druhej časti testu.

Priemerný čas, ktorý užívatelia potrebovali pre vytvorenie testovacieho návrhu zostal takmer rovnaký, ako pri prvom návrhu. ZnížilaZnížila sa však chybovosť užívateľov a to takmer o polovicu.

3.8 Zhrnutie

Pilotnou implementáciou editora bolo odhalených niekoľko závažných problémov, ktoré bolo pre ďalší vývoj aplikácie potrebné vyriešiť. Ďalej sa upresnila požadovaná funkčnosť, implementačné požiadavky a na základe testovania s užívateľmi bolo upravené užívateľské rozhranie do jednoduchšej podoby.

V tomto kroku začal navrhnuté riešenia v rámci svojho systému implementovať zadávateľ. Vývoj prototypu bol ukončený a mojou ďalšou úlohou v rámci bakalárskej práce bolo pokračovať v práci na finálnej verzii editora v týme. Konkrétne bolo mojou úlohou navrhnúť a implementovať časť aplikácie spracúvajúcej grafické primitíva a ďalej súčasť obrazových filtrov. Vývoj tak prebiehal súčasne s vývojom editora, do ktorého boli tieto riešenia následne integrované.

Kapitola 4

Grafické primitíva

Ďalšou úlohou bolo vytvoriť súčasť pre prácu s grafickými primitívami do finálnej verzie editora. Kapitola obshauje špecifikáciu požiadavkov pre prácu so základnymi grafickymi objektami, návrh funkčnosti, jej implementáciu a následne integrovanie do aplikácie.

4.1 Analýza

Pri tvorbe grafických návrhov je potrebné mať k dispozícií základné geometrické tvary, z ktorých je ďalej možné vytvárať zložitejšie elementy, vizuálne rozdeľovať a zvýrazňovať časti návrhu.

Dôraz musí byť kladený na jednoduchosť použitia a možnosť uplatnenia zaužívaných postupov z existujúcich grafických editorov.

Špecifikácia požiadaviek na funkčnosť

- Tvorba základných grafických elementov (štvorec, obdĺžnik, kruh, elipsa)
- Tvorba komplexnejších elementov (n-uholníky, hviezda, srdce ...)
- Možnosť transformácie elementov (zmena veľkosti, pomeru strán, polohy)
- Možnosť dekorácie elementov (farba výplne, farba orámovania)
- Galéria dostupných tvarov
- Zmena usporiadania vrstiev tvarov (poloha na Z ose)

Špecifikácia požiadaviek na implementáciu

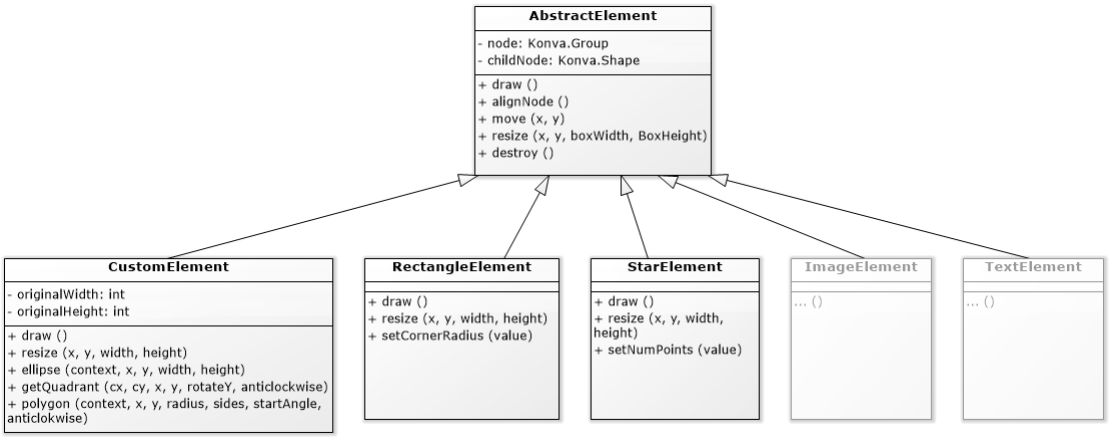
- História zmien (funkcia vpred/vzad)
- Oddelená implementácia od editora
- Použitie JavaScript knižnice pre Canvas

4.2 Návrh

Pre prácu s grafickými primitívami (neskôr aj s textovými elementami a obrázkami) bolo navrhnuté využiť funkcie knižnice Konva.js, ktorá implementuje množstvo základných operácií a poskytuje rozhranie pre prácu s nimi. Implementácia tried pre prácu s touto knižnicou bude oddelená do samostatného modulu pre lepšie logické členenie aplikácie. To umožnilo uľahčenie vývoja v tíme a hlavne umožnenie vývoja nezávisle od verzie editora (aktuálne nasadená verzia a rôzne vývojové verzie).

Štruktúra modulu

V tejto časti budú navrhnuté jednotlivé triedy, interakcie medzi nimi a popis ich funkcionality.



Obr. 4.1: Diagram tried modulu.

- **AbstractElement**
  Táto trieda bude obsahovať všetky spoločné funkcie (vykreslenie a odstránenie elementu, jeho posun, zarovnanie a zmenu veľkosti) a atribúty pre uloženie vykreslovaného elementu.

- **RectangleElement**
  Je abstrakciou pre element štvorca a obdĺžnika, ktorý je definovaný štyrmi základnými parametrami – pozíciou určenou súradnicami `X` a `Y` na plátne (Canvas), šírkou a výškou. Ďalej obsahuje tri doplnkové parametre – farba výplne, priehladnosť elementu a zaoblenie hrán.

- **StarElement**
  Abstrahuje element hviezdy tvorený piatimi hlavnými a štyrmi doplnkovými parametrami. Hlavné parametre definujú jej pozíciu, počet vrcholov, vnútorný a vonkajší priemer. Doplnkové parametre určujú jej farbu, priehladnosť, farbu orámovania a jeho šírku. Trieda bude tiež implementovať funkciu pre zmenu počtu vrcholov.

- **CustomElement**
  Objekt typu `CustomElement` tvorí pozícia a zoznam funkcií, špecifikujúcich tvar požadovaného objektu. Ako aj v predošlých objektoch obsahuje doplnkové parametre pre farbu, priehladnosť a orámovanie. Trieda primárne slúži pre abstrakciu komplexnejších elementov, ktoré nie sú definované v knižnici Konva.js. Tieto chýbajúce geometrické útvary je potrebné implementovať priamo pomocou nasledovných metód HTML5 `Canvas` 2D API: [16]

  - **moveTo** – posun počiatočného bodu na určené súradnice
  - **beginPath** – inicializácia cesty pred tvorbou nového útvaru
  - **lineTo** – vykreslenie čiary z aktuálnej pozície do bodu určeného súradnicami
  - **quadraticCurveTo** – vykreslenie kvadratickej Bézierovej krivky, ktorá je určená aktuálnou pozíciou, zadaným koncovým a Kontrolným bodom
  - **bezierCurveTo** – vykreslenie kubickej Bézierovej krivky, určenej aktuálnou pozíciou, zadaným koncovým a dvoma Kontrolnými bodmi
  - **arc** – vykreslenie oblúku so zadaným počiatočným a koncovým uhlom, polomerom a stredovými súradnicami
  - **closePath** – uzavretie cesty útvaru

- **ImageElement a TextElement**
  Triedy implementujúce textové a obrazové elementy.

4.3 Implementácia

V tejto časti je popísaná implementácia navrhnutých elementov a integrovanie modulu do editora.

- **RectangleElement a StarElement**
  Využíva objekty preddefinované v knižnici Konva.js a to konkrétne `Konva.Rect` a `Konva.Star`, ktoré implementujú požadovanú funkčnosť. Objektu je predaná štruktúra parametrov, na základe ktorých sa vytvorí požadovaný útvar.

- **CustomElement**
  Je realizovaný objektom `Konva.Shape`, ktorý zapúzdruje Canvas API. Pre vykreslenie každého útvaru sa mu predáva funkcia s odpovedajúcimi príkazmi, popisujúcimi požadovaný tvar. Kedže je potrebné vytvárať rôzne objekty, musia sa tieto funkcie vytvárať dynamicky.

```
1 [{ action: "beginPath" },
2 { action: "arc", x: 75, y: 75, r: 75, sAngle: -0.12*Math.PI, eAngle: 1.41*Math.PI
      , counterclockwise: false},
3 { action: "arc", x: 85, y: 55, r: 60, sAngle: -0.67*Math.PI, eAngle: -2.01*Math.
      PI, counterclockwise: true},
4 { action: "closePath" }],
```

Kód 4.1: Príklad datovej štruktúry pre popis tvaru mesiaca.

Vytváranie pravidelných mnohouholníkov bolo uľahčené vytvorením funkcie `polygon`. Namiesto ručného popisu príkazov popisujúcich tvar pre Canvas, ich táto funkcia generuje automaticky, podľa zadaných parametrov (polomer, počet strán).

```
function polygon(context, x, y, radius, sides, startAngle, anticlockwise) {
  if (sides < 3) return;
  let a = (Math.PI * 2) / sides;
  a = anticlockwise ? -a : a;

  ...

  for (var i = 1; i < sides; i++) {
    context.lineTo(radius*Math.cos(a*i), radius*Math.sin(a*i));
  }
  ...
}
```

Kód 4.2: Funkcia generujúca pravidelné mnohouholníky.

Ovládacie prvky

Okolo každého vybraného tvaru je potrebné vykresliť ovládací rámik pre zmenu veľkosti. Jeho rozmer je odvodený od veľkosti vykresleného tvaru. Pri Custom elemente nastáva problém s určením jeho veľkosti, ktorú je nutné na rozdiel od základných elementov vyrátať ručne pre každý z tvarov. Súradnice krajných bodov sa zisťujú počas vytvárania funkcie vykreslujúcej útvar.

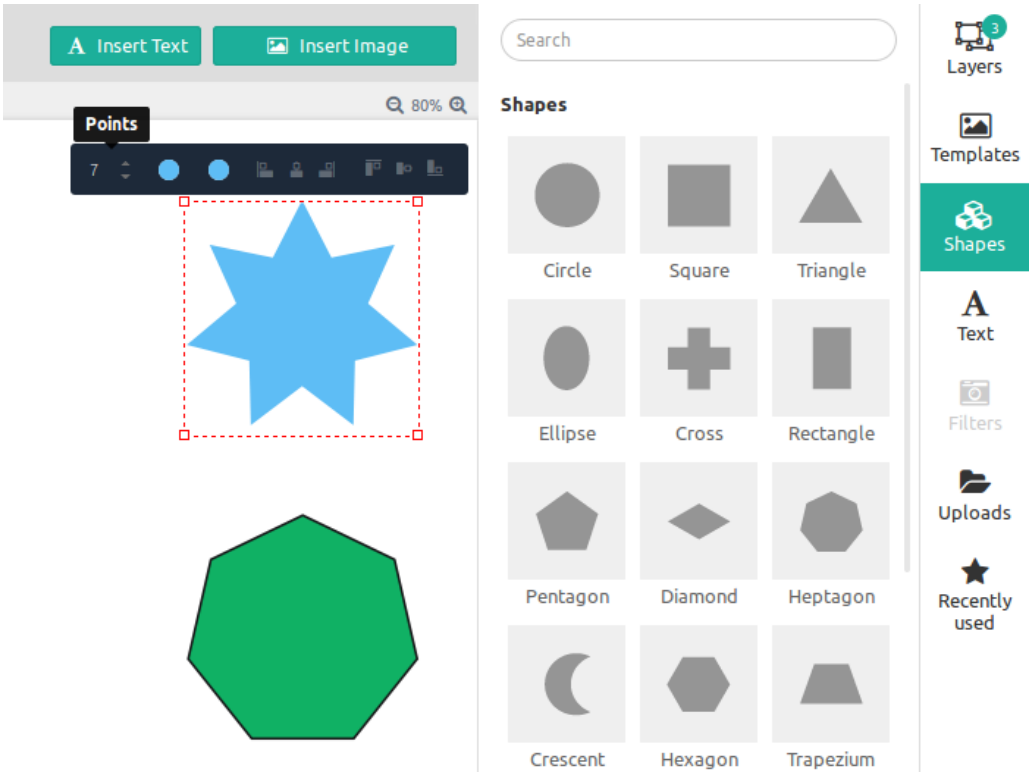
Funkcionalita vpred/vzad

Pre funkcie vpred/vzad je nutné uchovávať históriu zmien (zmena veľkosti tvaru, zmena farby ...). Objekt knižnice Konva.js však uchováva iba aktuálny stav objektu. Preto sa nastavenia objektu musia navyše ukladať aj v datovom modeli editoru (states), ktorý je implementovaný architektúrou Redux. To zabezpečuje, že celá história zmien parametrov bude uchovaná v stavovom strome:

Obr. 4.2: Aktuálny stav editora a detail uložených parametrov elementu.

4.4 Zhrnutie

V tejto časti bolo navrhnuté a implementované riešenie grafických primitív. Výslednú implementáciu a jej integrovanie do aktuálnej verzie editora od zadávateľa je vidieť na obrázku 4.3. Do grafického rozhrania bolo potrebné pridať ovládacie prvky – galériu dostupných tvarov do bočného panelu a upraviť kontextové menu s nastaveniami parametrov jednotlivých objektov, zobrazujúce sa nad tvarmi.



Obr. 4.3: Rozhranie grafických primitív v editore.

Kapitola 5

Obrazové filtre

Kapitola sa venuje ďalšiemu modulu vytváreného do finálnej verzie editora. Jeho úlohou bude implementovať **obrazové filtre** pre úpravu obrázkov v editore. Kapitola popisuje požadovanú funkcionalitu, návrh riešenia a následne popisuje ako bol návrh implementovaný.

5.1 Analýza

Webový nástroj je primárne určený pre tvorbu grafických prezentácií, ktoré pozostávajú z veľkého počtu obrazových dát produktov a preto je žiadúce mať k dispozícií sadu filtrov pre ich úpravu. Obrázky často obsahujú nedokonalosti ako šum, nesprávne vyváženie farieb, kontrastu a iné. Je preto potrebné mať možnosť upraviť ich základné parametre – jas, kontrast, prípadne aplikovať štylistické filtre – rozostrenie, prevod do odtieňov šedej...

Obrázky v editore sa rozdeľujú do dvoch kategórií – statické obrázky manuálne vložené užívateľom a dynamické obrázky vkladané do návrhu z produktových katálogov. Tomuto rozdeleniu je vhodné prispôsobiť výber použitých filtrov. Pri práci s dynamickými obrázkami totiž nie je možné nastavovať jednotlivé korekcie ručne a je potrebné túto prácu automatizovať. Riešením je implementovať sadu automatických filtrov pre jednoduché korekcie obrázkov z celého produktového katalógu.

Špecifikácia požiadavkov na funkčnosť

- Sada manuálne nastaviteľných filtrov
- Sada automatických filtrov
- Možnosť kombinovať jednotlivé filtre

Špecifikácia požiadavkov na implementáciu

- Implementácia v samostatnom module
- Neblokujúca implementácia
- Možnosť jednoduchého pridávania ďalších filtrov
- Vyrovnávacia pamäť pre filtre

5.2 Návrh

Implementácia obrazových filtrov bude umiestnená v samostatnom module spolu so všetkými ostatnými triedami pracujúcimi s knižnicou Konva.js. Knižnica poskytuje niekoľko obrazových efektov, avšak tieto filtre nie sú implementované pre asynchrónne spracovanie. Pri výpočetne náročnejších filtroch by tak blokovali vykreslovacie jadro editora, čo by malo za následok jeho zaseknutie. Je teda potrebné navrhnúť vlastnú, asynchrónnu implementáciu filtrov.

Asynchrónne spracovanie

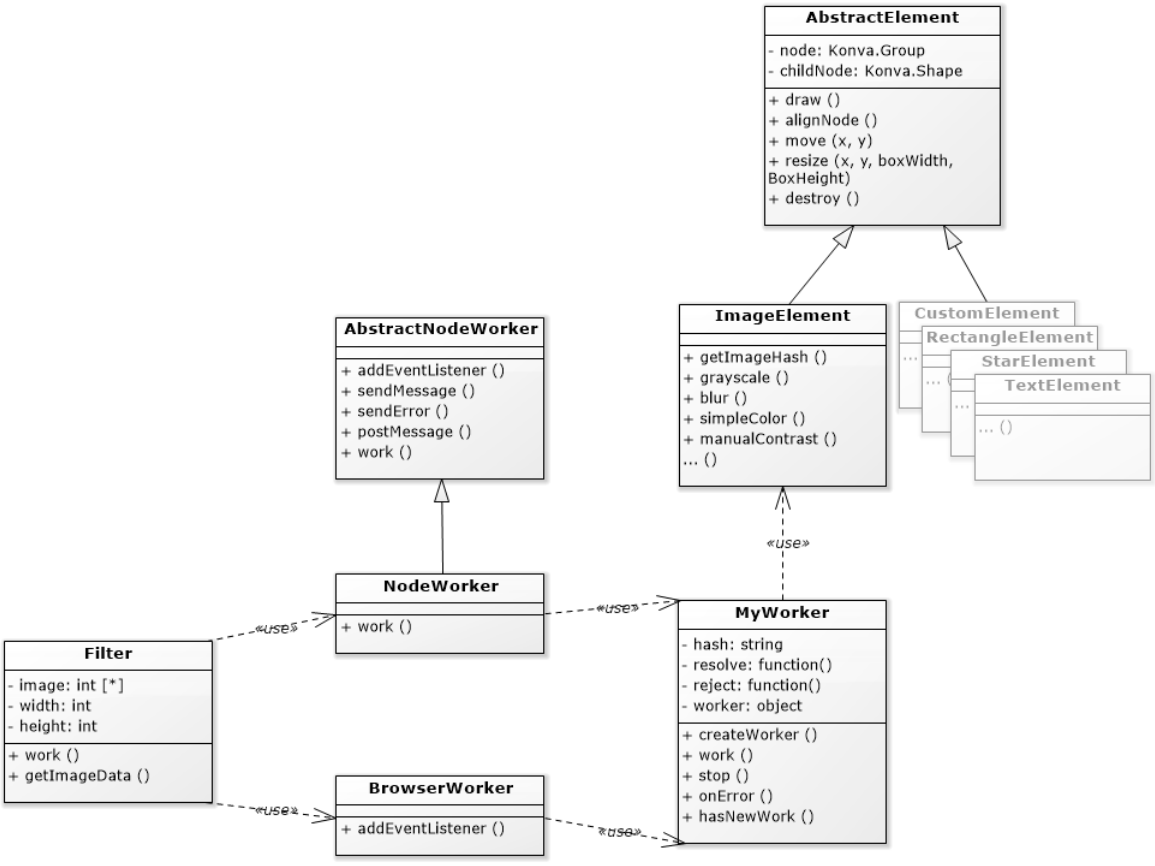
Asynchrónne spracovanie filtrov je možné zabezpečiť pomocou objektov **Web Workers** a **Promise**. Web Workers umožňujú beh JavaScript kódu v samostatnom vlákne na pozadí. Worker môže vykonávať úlohy a vstupno výstupné operácie nezávisle na užívateľskom rozhraní a pomocou zasielania správ komunikovať s procesom, ktorý ho vytvoril. [17]
Treba však zabezpečiť, aby proces ktorý Web Worker vytvoril, nečakal na jeho odpoveď. Pri vytváraní Workeru sa preto využije štandardného JavaScript objektu Promise. Tento objekt zastupuje hodnotu, ktorá v čase jeho vytvorenia ešte nie je známa, čo umožňuje asynchrónnym metódam vrátiť hodnoty rovnakým spôsobom ako metódy synchrónne. [18]

Logická štruktúra modulu

V editore sa pri každom použití filtra vytvorí nový Worker. Vytvárať ho bude továreň **MyWorker**, ktorá ďalej na základe predaných parametrov vytvorí nový Worker pre požadovaný filter. Podľa toho, či je aplikácia spustená v prehliadači alebo na servri Node.js sa použije **BrowserWorker**, respektíve **NodeWorker**. Továreň **MyWorker** sa vytvára pomocou objektu Promise, v triede **ImageElement**, čo zaručí asynchrónne spracovanie. Implementácia bola použitá z modulu odstraňovania bieleho pozadia poskytnutého zadávateľom.

Vyrovnávacia pamäť

Aplikácia automatických filtrov je výpočetne náročná operácia a trvá niekoľko sekúnd, kým sa vykonajú. Pre zvýšenie pohodlia sa preto výsledný obrázok po každom aplikovaní filtra, prípadne kombinácií filtrov uloží do vyrovnávacej pamäte.



Obr. 5.1: Diagram tried modulu.

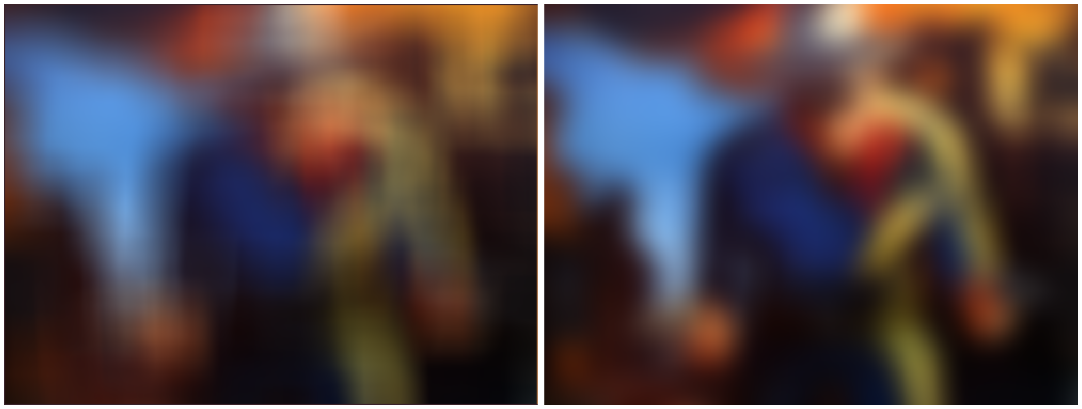
5.2.1 Filtre

Táto časť obsahuje popis filtrov, ktoré budú implementované v aplikácií a vysvetľuje princíp ich fungovania a výber konkrétnych algoritmov.

- **Jas a kontrast**
  Patria medzi základne obrazové korekcie a medzi najpoužívanejšie filtre pri úprave obrázkov produktov v editore. Ich manuálna varianta bude vhodná pre statické obrázky v šablónach.

  Princíp úpravy jasu spočíva v prirátavaní zmeny ku každej farebnej zložke jednotlivých pixelov. Pre úpravu kontrastu je nutné každú farebnú zložku jednotlivých pixelov násobiť korekčným faktorom.

- **Rozostrenie**
  Často používaný efekt pre estetické úpravy obrázkov alebo odstránenie šumu. Funkcie rozostrenia fungujú na princípe výpočtu váženého priemeru okolia pixelu v zdrojovom obrázku pre jeden pixel výsledného obrázku. Pre použitie v editore bol zvolený algoritmus StackBlur, ktorý poskytuje kompromis medzi známejšími Box alebo Gaussian blur. Ponúka lepší výsledok ako pri použití Box blur a zároveň je sedem krát rýchlejší ako Gaussian blur. [19]



Obr. 5.2: Porovnanie Box blur a Stack blur.

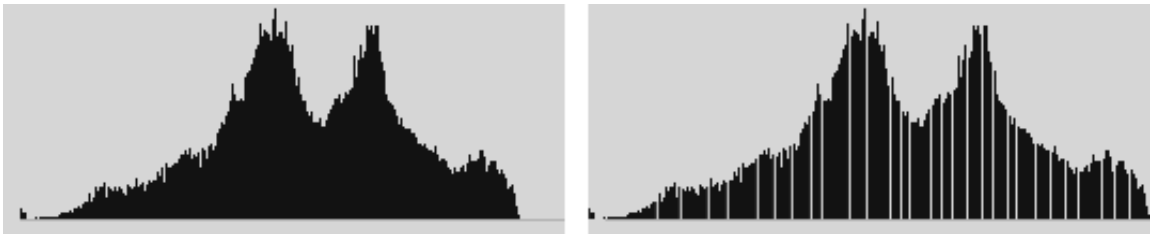
- **Grayscale**
  Estetický efekt so širokým spektrom použitia. Elementárny algoritmus prevodu do stupňov šedi (priemerovanie) vyrátava novú hodnotu pixelu spriemerovaním jeho farebných zložiek pred prevodom. Existuje množstvo pokročilejších variant prevodu, z ktorých bol vybraný algoritmus `luminosity` často používaný v grafických editoroch. [20] Jednotlivým farebným zložkám prikladá váhy zohľadňujúce vnímanie farieb ľudského oka, čím dosahuje prirodzenejšieho výsledku. Použité konštanty pre váhy sú podľa štandardu ITU-R BT.601. [21]

$$Gray = (Red * 0.299 + Green * 0.587 + Blue * 0.114)$$

- **Sépia**
  Ďalší estetický efekt pre prevod obrázku do teplých odtieňov hnedej. Algoritmus funguje obdobne ako priemerovanie pri prevode do stupňov šedi, navyše však zvýši zelenú zložku pixelu o hodnostu `n` a červenú zložku o `2n`. Algoritmus použitý v editore je rozšírený o nastavenia intenzity pomocou modifikácie modrej zložky.

- **Automatický kontrast**
  Keďže nastavenie kontrastu patrí v editore medzi často používané úpravy, je žiadúce implementovať jeho automatickú verziu, ktorá by vhodne upravila parametre obrázkov zo všetkých produktov vo zvolenom produktovom katalógu.

  Jednou z najjednoduchších a zároveň efektívnych metód pre úpravu kontrastu je *Contrast stretching* tiež známy ako normalizácia. [22] Princíp je založený na roztiahnutí rozsahu hodnôt intenzity pre jednotlivé body obrázka, čím umožní plné využitie možných hodnôt. [23] Na rozdiel od metódy *ekvalizácie histogramu* je obmedzená na lineárne mapovanie vstupných hodnôt na výstupné. Výsledný efekt je tak menej intenzívny ale vyzerá prirodzenejšie. [24]



Obr. 5.3: Roztiahnutie histogramu.

Prvým krokom je nájdenie minimálnych a maximálnych hodnôt, zvlášť pre každú farebnú zložku obrázka (R,G,B). Tieto určia interval, ktorý sa roztiahne na šírku histogramu, aby sa využil celý rozsah jeho hodnôt. Nové hodnoty pixelov obrazu sa pre každú farebnú zložku vyrátajú podľa nasledovného vzorca: [23]

$$out_{RGB}(x,y) = 255 \times \left[\frac{in_{RGB}(x,y) - min_{RGB}}{max_{RGB} - min_{RGB}} \right],$$

kde $\boldsymbol{min_{RGB}}$ a $\boldsymbol{max_{RGB}}$ sú minimálna a maximálna nájdená hodnota, $\boldsymbol{in_{RGB}}$ je pôvodná hodnota pixelu a $\boldsymbol{out_{RGB}}$ je jeho nová hodnota.

Vzorec teda pre každú farebnú zložku posunie rozsah na nulu a podelí ho rozdielom minimálnej a maximálnej hodnoty danej zložky. Výsledok sa vynásobením roztiahne na celú šírku histogramu.

- **Automatické vyváženie bielej**
  Ako ďalší automatický filter bolo zvolené vyváženie bielej. Obrázky v produktovom katalógu môžu mať rôznu teplotu farieb, čo by vo výslednej reklame vyzeralo neesteticky.

  Existujúce algoritmy pre automatické vyváženie bielej (ďalej iba AWB) by sa dali rozdeliť do dvoch kategórií. Globálne AWB algoritmy, odhadujúce teplotu obrázka zo všetkých jeho pixelov a lokálne algoritmy, odhadujúce teplotu iba podľa pixelov spĺňajúcich určitú podmienku. [25]

  Jeden z najjednoduchších AWB algoritmov je algoritmus gray world, ktorý predpokladá, že v dobre vyváženom obraze je priemer zo všetkých farebných zložiek obrazu (R,G,B) neutrálna šedá. [26] Korekcia sa následne vyráta porovnaním tohto priemeru obrazu so šedou farbou.

  Tento globálny algoritmus však nepracuje dobre s obrazmi, kde dominuje jedna alebo dve farby. V editore bude použitý algoritmus Robust Auto White-Balance, ktorý sa snaží tento nedostatok riešiť. [25]

  Algoritmus sa najprv snaží odhadnúť teplotu obrazu. Prevedie ho do farebného modelu YUV vynásobením maticou:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.299 & -0.587 & 0.886 \\ 0.701 & -0.587 & -0.114 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

  a odhadne iluminant nájdením pixelov podobných šedej v rámci nastaveného prahu:

$$(|U| + |V|)/Y < T = 0.3$$

Podľa odhadu potom nastaví vyváženie bielej zvýšením červeného a modrého kanálu. Tento postup sa opakuje až pokým nedosiahne nastaveného maximálneho počtu iterácií alebo úplnej korekcie.

5.3 Implementácia

Spracovanie obrázkov

Jednotlivé obrázky sú uložené v JavaScript objekte Image, v ktorom sú obrazové dáta uložené v poli. Pre každý pixel sú v ňom uložené 4 hodnoty obsahujúce informácie o každej farebnej zložke (R, G, B) a priehľadnosti (alfa kanál).

Pred použitím filtra AWB sa toto pole ďalej prevádza na objekt Matrix z knižnice Math.js, nad ktorým sú definované funkcie pre prácu z maticami.

Po aplikovaní filtra je potrebné maticu spracovaného obrázka previesť späť do poľa, aby ho bolo možné vykresliť alebo naň bolo v prípade potreby možné aplikovať ďalší filter. Pre vykreslenie sa dátove pole uloží do objektu Image z knižnice Konva.js.

Vyronávacia pamäť

Obrázky vo vyrovnávacej pamati sú uložené v mape – nezotriedenej dátovej štruktúry binárnych relácií (kľúč, hodnota), z knižnice Immutable. Ako kľúč sa používa hash, vytvorený z parametrov aplikovaných filtrov, čo zabezpečí uloženie obrázkov aj pre rôzne nastavenia rovnakého filtra alebo ľubovoľnú kombináciu viacerých filtrov.

Pred každou aplikáciou obrazového filtra sa najprv skontroluje, či vo vyrovnávacej pamäti neexistuje záznam pre zvolený filter a jeho parametre, prípadne kombináciu filtrov.

Poradie aplikovania filtrov

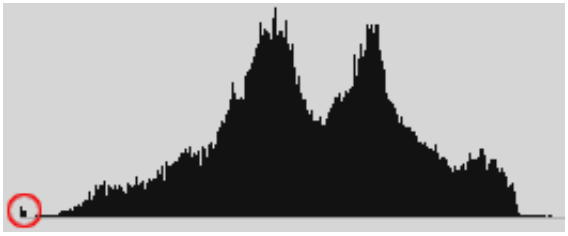
Pri aplikovaní viacero filtrov na jeden obrázok je potrebné určiť v akom poradí sa tieto filtre vykonajú. Niektoré poradia by totiž užívateľom nepriniesli očakávané výsledky – napríklad pri aplikovaní efektu sépia a následnom použití Automatického vyváženia bielej. Pre zachovanie jednoduchosti aplikácie je poradie aplikovania filtrov implementované na pevno.

```
1 .then((image) => this.rawb(image))
2 .then((image) => this.simpleColor(image))
3 .then((image) => this.manualContrast(image))
4 .then((image) => this.manualBrightness(image))
5 .then((image) => this.removeBackground(image))
6 .then((image) => this.autoContrast(image))
7 .then((image) => this.sepia(image))
8 .then((image) => this.grayscale(image))
9 .then((image) => this.blur(image))
```

Kód 5.1: Poradie vykonávania filtrov.

Automatický kontrast

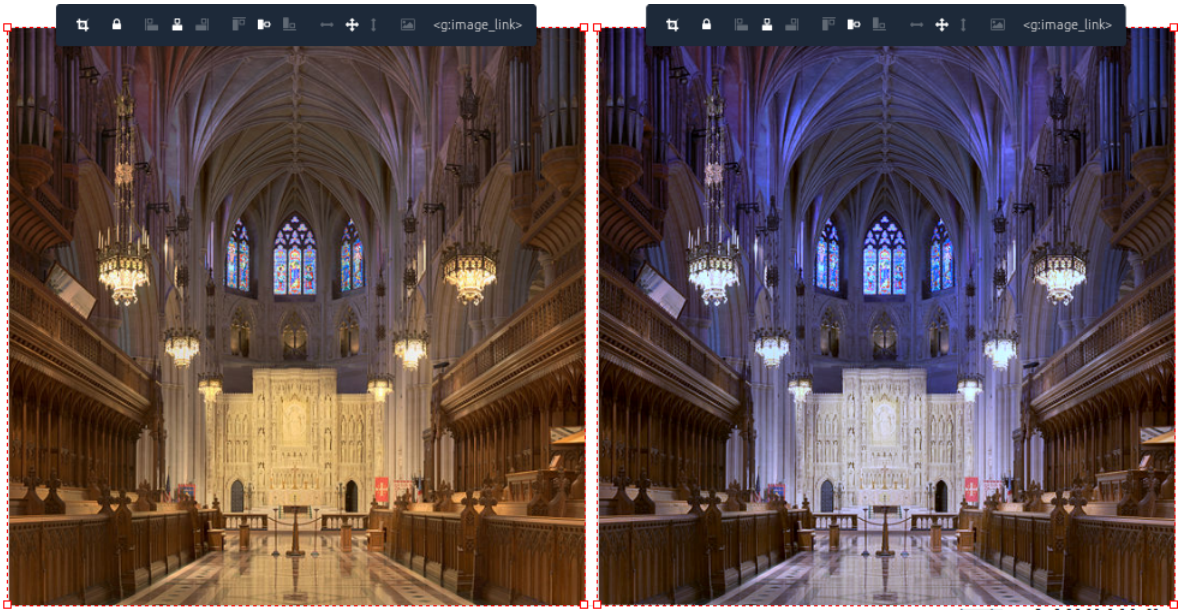
Algoritmus popísaný v návrhu môže strácať efektivitu, ak sa v histograme nachádzajú extrémy, čo má za následok určenie príliš širokého intervalu. Riešením je ignorovať určité množstvo pixelov ležiacich na okrajoch histogramu. [24]



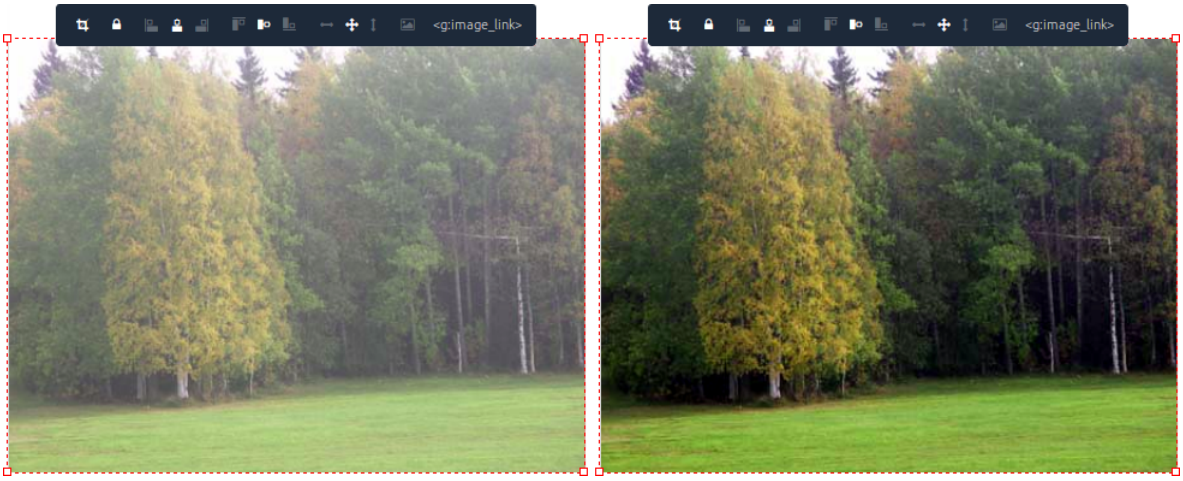
Obr. 5.4: Extrém v histograme.

V implementácií sa tento problém rieši vyrátaním vrchného a spodného kvantilu pre každú farebnú zložku. Ten určí hodnotu na histograme, pod respektíve nad ktorou leží nami zadané percento pixelov, ktoré chceme ignorovať.

Ukážky implementovaných automatických filtrov



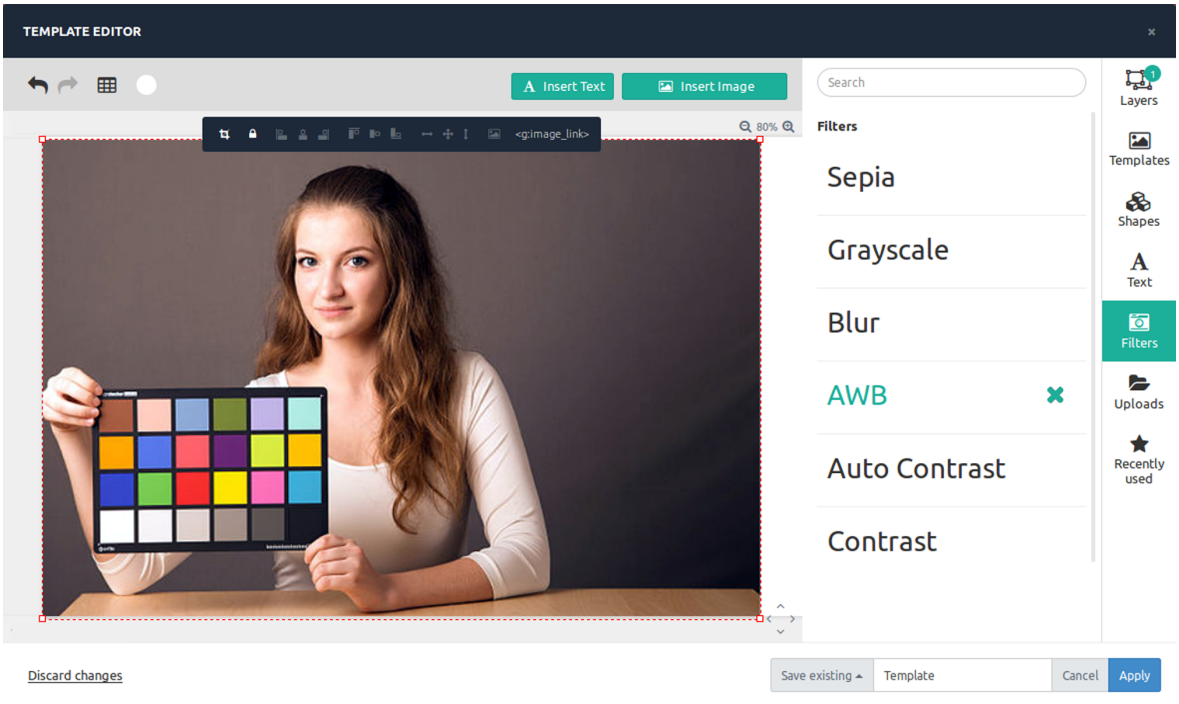
Obr. 5.5: Aplikácia algoritmu AWB v editore.



Obr. 5.6: Aplikácia automatického kontrastu v editore.

5.4 Zhrnutie

V tejto kapitole bola navrhnutá implementovaná sada filtrov pre korekcie obrázkov. Výsledná implementácia je zobrazená na obrázku 5.7. Integrovanie do editora spočívalo v pridaní galérie filtrov do grafického rozhrania. Galéria filtrov je v menu dostupná, iba ak je označený práve jeden obrázok.



Obr. 5.7: Rozhranie obrazových filtrov v editore.

Kapitola 6

Záver

Cieľom práce bolo v spolupráci so zadávateľom navrhnúť webový nástroj pre tvorbu prezentácií produktov s podporou produktových katalógov.

Pre riešenie zadania bolo nutné naštudovať si problematiku tvorby webových aplikácií, tvorby užívateľského rozhrania a identifikovať požiadavky zadávateľa a potencionálnych užívateľov. Na základe týchto informácií bolo navrhnuté riešenie, podľa ktorého bola vytvorená pilotná implementácia tohto nástroja. Táto implementácia pomohla získať cenné informácie, vďaka ktorým bolo možné upresniť požiadavky, identifikovať množstvo technických problémov a navrhnúť ich riešenie.

Ďalej sa práca zamerala na implementáciu grafických primitív a obrazových filtrov. Boli identifikované funkčné a nefunkčné požiadavky, na základe ktorých boli tieto časti implementované a integrované do riešenia zadávateľa.

Vývoj nástroja je v sučasnosti stále aktívny a mojou úlohou je ďalej pracovať na vylepšení obrazových filtrov a ich rozšírení, napríklad o automatické odstraňovanie bieleho pozadia z obrázkov produktov. Pre rýchly rozvoj webových technológií„ pripadá v dlhšom časovom horizonte do úvahy možnosť generovať prezentácie vo forme HTML kódu, namiesto rastrových obrázkov.

Okrem informácií získaných štúdiom problematiky, je veľmi pozitívne hodnotená práca na projekte v reálnom prostredí a zapojenie sa do procesu vývoja aplikácie od počiatočného formulovania požiadavok až po následnú implementáciu.

Literatúra

[1] *Data feeds overview [online].* [cit. 2016-05-07]. Dostupné z: <https://support.google.com/merchants/answer/188478?hl=en>

[2] *MVC Architecture [online].* [cit. 2016-05-07]. Dostupné z: <https://developer.chrome.com/apps/app_frameworks#top>

[3] Osmani, A. *Learning JavaScript Design Patterns.* O'Reilly Media, 2012. ISBN 978-1-4493-3486-4. Dostupné z: <https://addyosmani.com/resources/essentialjsdesignpatterns/book/#detailmvcmvp>.

[4] Salihefendic, A. : *Flux vs. MVC (Design Patterns) [online].* 2015-2-10 [cit. 2016-05-07]. Dostupné z: <https://medium.com/hacking-and-gonzo/flux-vs-mvc-design-patterns-57b28c0f71b7#.ngcdujf05>

[5] *Flux - Overview [online].* [cit. 2016-05-07]. Dostupné z: <https://facebook.github.io/flux/docs/overview.html>

[6] Osmani, A. : *Writing Modular JavaScript With AMD, CommonJS & ES Harmony [online].* [cit. 2016-05-07]. Dostupné z: <https://addyosmani.com/writing-modular-js/>

[7] Rauschmayer, A. : *Exploring ES6 - Upgrade to the next version of JavaScript [online].* 2016-04-11 [cit. 2016-05-07]. Dostupné z: <http://exploringjs.com/es6/ch_modules.html>

[8] *Babel* [online]. [cit. [2016-05-08]]. Dostupné z: <https://babeljs.io/>.

[9] *React* [online]. [cit. [2016-05-08]]. Dostupné z: <https://facebook.github.io/react/>.

[10] Sheridan, M. : *The Developer's Guide to HTML5 Canvas [online].* [cit. 2016-05-07]. Dostupné z: <https://msdn.microsoft.com/en-us/hh534406.aspx>

[11] Glover, A. : *Node.js for Java developers [online].* [cit. 2016-05-07]. Dostupné z: <http://www.ibm.com/developerworks/library/j-nodejs/>

[12] DeBill, E. *Module Counts* [online]. [cit. [2016-05-07]]. Dostupné z: <http://www.modulecounts.com/>.

[13] Hartson, R. *The UX Book: process and guidelines for ensuring a quality user experience.* Morgan Kaufmann, 2012. ISBN 0123852412.

[14] Group, N. N. : Turn User Goals into Task Scenarios for Usability Testing. 2014-1-12 [cit. 2016-05-10]. Dostupné z: <https://www.nngroup.com/articles/task-scenarios-usability-testing/>

[15] Nielsen, J. : How Many Test Users in a Usability Study? 2014-6-4 [cit. 2016-05-10]. Dostupné z: <https://www.nngroup.com/articles/how-many-test-users/>

[16] W3C. *HTML Canvas 2D Context* [online]. 2015-11-19 [cit. 2016-05-09] Dostupné z: <https://www.w3.org/TR/2dcontext/#building-paths>.

[17] Using Web Workers. 2016-4-29 [cit. 2016-05-09]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers>

[18] Promise. 2016-5-2 [cit. 2016-05-09]. Dostupné z: <https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Promise>

[19] Govett, D. : *StackBlur: A Fast Gaussian Blur Algorithm In JavaScript [online].* 2011-10-11 [cit. 2016-05-09]. Dostupné z: <http://badassjs.com/post/1298940200/stackblur>

[20] Christopher Kanan, G. W. C. : *Color-to-Grayscale: Does the Method Matter in Image Recognition? [online].* 2012-1-10 [cit. 2016-05-09]. Dostupné z: <http://tdlc.ucsd.edu/SV2013/Kanan_Cottrell_PLOS_Color_2012.pdf>

[21] *Studio encoding parameters of digital television [online].* 2015 [cit. 2016-05-07]. Dostupné z: <https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.601-7-201103-I!!PDF-E.pdf>

[22] Normazida Jusoh, R. A. S. Color Image Enhancement using Contrast Stretching on a Mobile Device. *International Journal of Image Processing and Visual Communication* [online]. 2012-12 [cit. 2016-05-09], vol. 1, iss. 3 Dostupné z: <http://www.ijipvc.org/article/IJIPVCV1I302.pdf>. ISSN 1462-0332.

[23] Abdul Nasir, A. S. Modified Global and Modified Linear Contrast Stretching Algorithms: New Colour Contrast Enhancement Techniques for Microscopic Analysis of Malaria Slide Images. *Computational and Mathematical Methods in Medicine* [online]. 2016-1 [cit. 2016-05-09]. DOI: 10.1155/2012/637360., vol. 2012 Dostupné z: <http://www.hindawi.com/journals/cmmm/2012/637360/>. ISSN 1748-670x.

[24] Fr. Don Matthys, S. : Contrast stretching. 2001-1-31 [cit. 2016-05-09]. Dostupné z: <http://academic.mu.edu/phys/matthysd/web226/Lab01.htm>

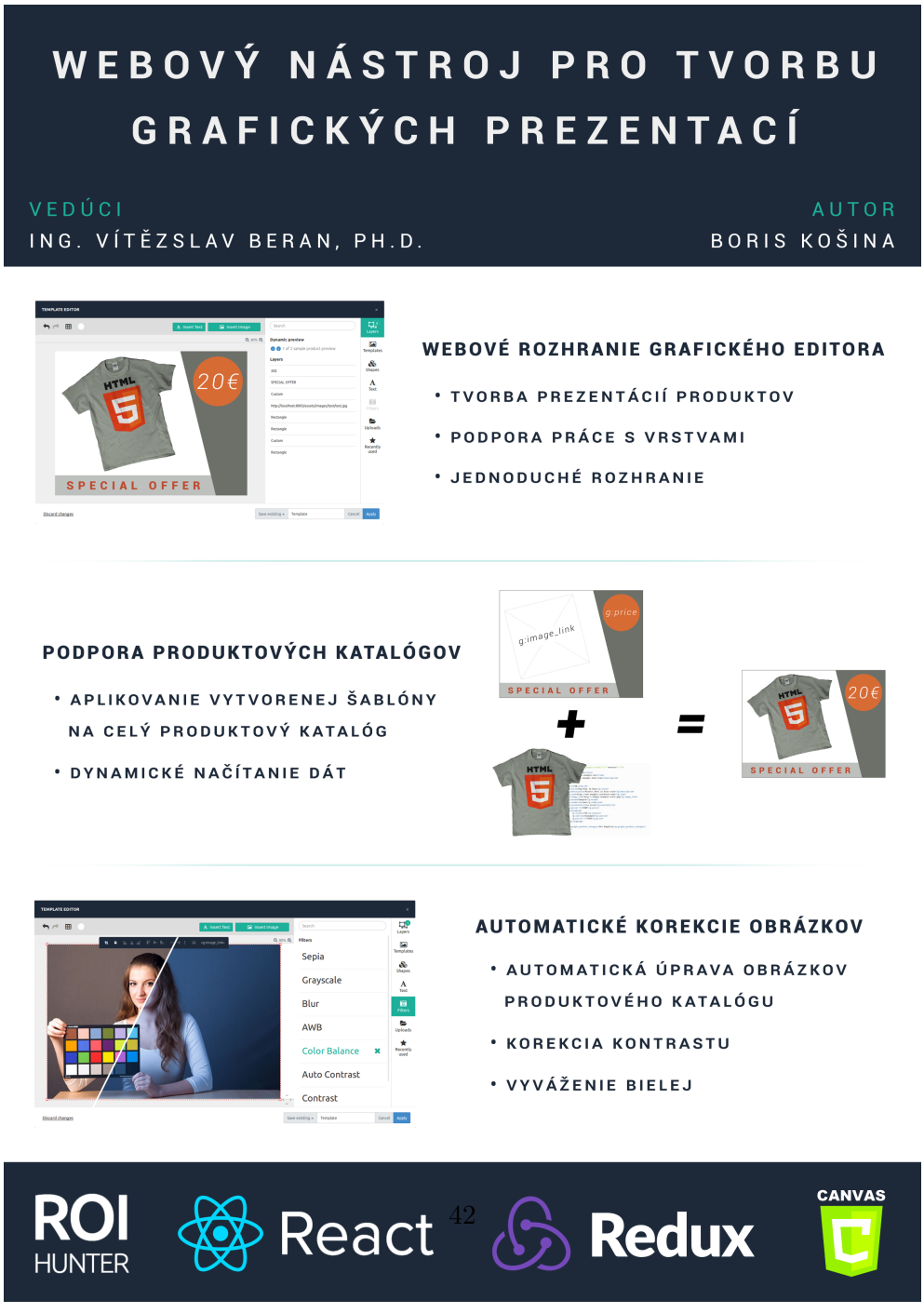
[25] Jun-yan Huo, J. W., Yi-lin Chang. Robust Automatic White Balance Algorithm using Gray Color Points in Images. *IEEE Transactions on Consumer Electronics* [online]. 2006-7-5 [cit. 2016-05-09], vol. 52, iss. 2 Dostupné z: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1649677>. ISSN 0098-3063.

[26] Su, J. : Illuminant Estimation: Gray World. [cit. 2016-05-09]. Dostupné z: <http://web.stanford.edu/~sujason/ColorBalancing/grayworld.html>

Prílohy

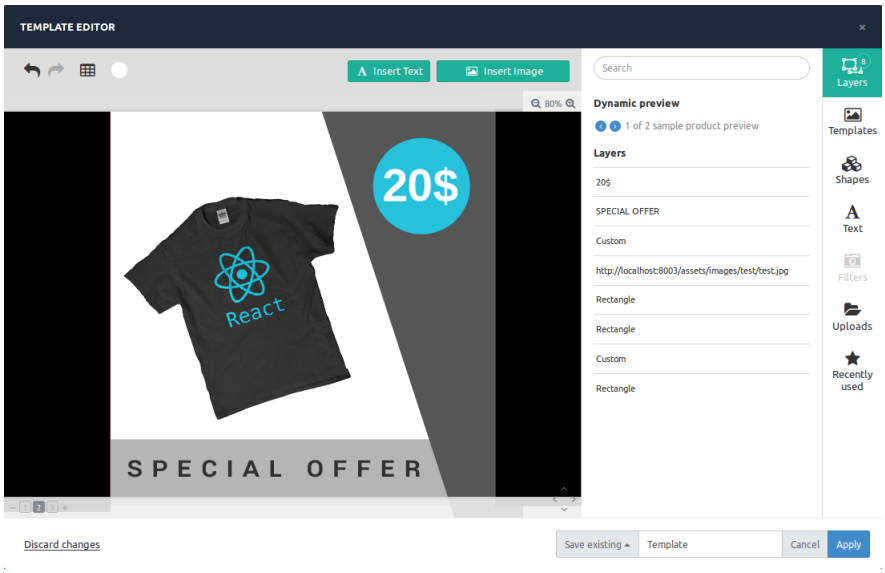
Príloha A

Plagát

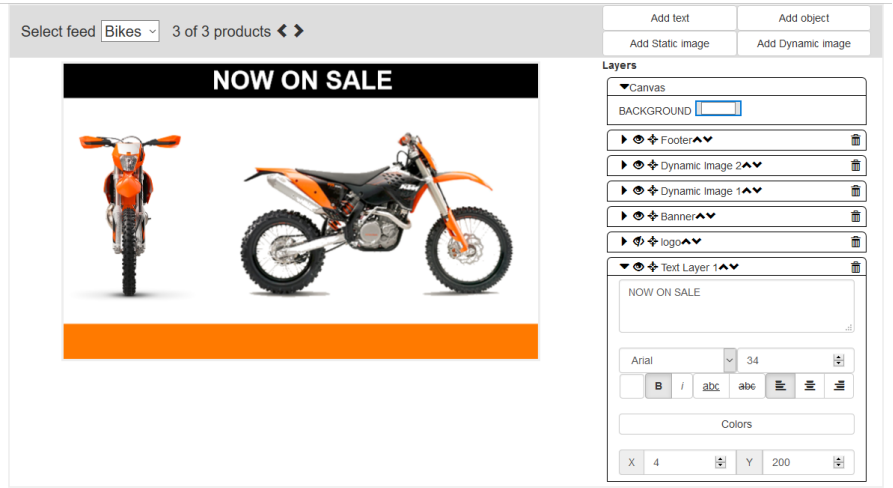


Príloha B

Snímky obrazovky



Obr. B.1: Finálna implementácia.



Obr. B.2: Pilotná implementácia.

Príloha C

Obsah CD

Přiložené CD obsahuje:

- adresár `build` s buildom aktuálnej verzie editora;
- adresár `src` so zdrojovými kódmi vytvorenými v rámci tejto práce;
- adresár `text` so zdrojovými kódmi tejto technickej správy;
- video a plagát prezentujúci túto prácu;
- túto technickú správu v PDF;