VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

FOTOSOUTĚŽ NA FACEBOOK

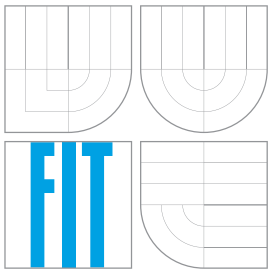
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETR POLOLÁNÍK

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

FOTOSOUTĚŽ NA FACEBOOK

PHOTOCONTENT ON FACEBOOK

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PETR POLOLÁNÍK
AUTHOR

VEDOUCÍ PRÁCE — Ing. VÍTĚZSLAV BERAN, Ph.D.
SUPERVISOR

BRNO 2015

Abstrakt

Tato práce se zabývá teorií vytváření moderních webových aplikací, jejichž logika je převážně v klientském prohlížeči. Jsou rozebrány různé technologie, které se dnes běžně využívají. Práce se převážně zabývá tvorbou konkrétní aplikace pro snadné vytváření fotosoutěží umístěných na Facebook stránce. Tyto fotosoutěže pořádají firmy pro zviditelnění své značky. Jsou specifikovány požadavky na výslednou aplikaci a vytvořen návrh. Poté je popsána implementace aplikace a možnosti rozšíření.

Abstract

This bachelor's thesis deals with the theory of creating modern web applications, whose logic is solved predominantly on server. Various technologies that are commonly used are analyzed. Thesis mainly deals with creating specific application for managing photo-contests on Facebook site. These photo-contests are organized by companies to expose their brand. Requirements for the application are specified and a final draft is created. Then the implementation and application scalability are described.

Klíčová slova

web, REST, React, Flux, Facebook, JavaScript, PHP, Nette

Keywords

web, REST, React, Flux, Facebook, JavaScript, PHP, Nette

Citace

Petr Pololáník: Fotosoutěž na Facebook, bakalářská práce, Brno, FIT VUT v Brně, 2015

Fotosoutěž na Facebook

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vítězslava Berana, Ph.D.

.....
Petr Pololáník
19. května 2015

Poděkování

Chtěl bych poděkovat svému vedoucímu bakalářské práce Ing. Vítězslavu Beranovi, Ph.D. za odborné vedení, za pomoc a rady při zpracování této práce.

© Petr Pololáník, 2015.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Kapitola 1

Úvod

V dnešní době jsou sociální sítě velice rozšířené a velká část uživatelů je aktivně využívá každým den. Sociální síť Facebook patří do té nejrozšířenější. Aby firmy byly blíže svým potencionálním klientům, přesouvají svůj marketing na Facebook. Facebook jim k tomu poskytuje speciální nástroje pro analýzu přístupu a placených reklam, které mohou být cíleně zaměřené na uživatele. Firmy na své stránky vkládají informace o novinkách a můžou vést diskuze s uživateli. Jeden z nástrojů, který Facebook firmám poskytuje, je vložení své vlastní aplikace, které mohou firemní profil obohatit. Mezi nejčastější aplikace patří například jídelní lístek, youtube kanál a nebo soutěže. Soutěže jsou marketingově velice efektivní, protože uživatelé za vidinou výhry mají důvod vstoupit na firemní profil a zajímat se o obchodní značku, nebo službu. Soutěže jsou většinou založené na zasílání fotografií a následným hodnocením uživatelů o nejlepší z nich. Firmy si tyto fotosoutěže nechají dělat na zakázku a nebo využijí speciálních aplikací, díky kterým si mohou soutěž vytvořit během pár minut bez znalosti programování. Tyto aplikace jsou nejčastěji používané, neboť jsou na rozdíl od aplikací tvořené na zakázku, daleko levnější a poskytují více možností. Jejich nevýhodou je pouze to, že jsou velice obecné a nedokážou proto splnit některé unikátní požadavky klienta.

Dokument se zabývá vytvořením webové aplikace, která poskytne firmám nástroj pro jednoduché vytvoření fotosoutěže, její vložení na svůj profil a následně její správu. Aplikace je rozdělena na serverovou a klientskou část, které spolu komunikují přes standardizované rozhraní REST, takže je možné si dopsat vlastního klienta.

Kapitola 2

Teorie

Na začátku první kapitoly bude popsaná sociální síť Facebook, její vztah k firmám a nástroje, které jim nabízí 2.1. Dále budou popsány základní webové technologie a některé architektury, které se při stavbě aplikace využijí 2.2. Následně se popíše moderní přístup k vývoji webových aplikací 2.3. Poté bude popis možných architektur pro snažší vytváření uživatelského prostředí 2.4. V podkapitolách 2.5 a 2.6 budou rozebrány frameworky, které usnadňují vývoj. Pro poskládán aplikace z více frameworků, lze využít správců balíčku, jeden z nich bude popsán v podkapitole 2.7. Na kapitoly bude průzkum 2.8 o existujících službách, které podobnou aplikaci nabízí, jejich vzájemné porovnání a zhodnocení.

2.1 Facebook a aplikace

Facebook je webový systém, pomocí kterého lze například vytvářet sociální sítě, komunikovat nebo sdílet obsah. Je to jedna z největších společenských sítí na světe. Počet aktivních uživatelů dosáhl 1,5 miliardy (k únoru 2015). Facebook firmám dává možnost vytvořit si pro komerční využití svůj oficiální firemní profil a propagovat tak svoji značku. Firma je díky tomu blíže svým potencionálním klientům a může s nimi navazovat vztahy [11].

V Česku tuto službu již využívá 60% obyvatel. Pro různé firmy má Facebook odlišný význam. Některým přináší nové zákazníky, jiným podporuje prodej a některým slouží jako zákaznický servis. Problém je že většina firem jeho potenciál neumí využívat a veškerý obsah, který propagují jeho prostřednictvím jsou pouze reklamy na zboží či služby [1].

Firmy na své facebookové stránce mají v základu 4 aplikace pro své návštěvníky a to Fotky, Události, Videa a To se mi líbí?. Navíc mají možnost přidat si aplikace vytvořené 3. stranou (nikoliv Facebookem). Firmy si buď nechají aplikaci vytvořit od externí firmy na zakázku (popřípadě sami, pokud mají vlastní vývojáře). Nebo mají možnost využít některou ze služeb nabízející již několik předem vytvořených aplikací, které se dají snadno bez znalosti programování nainstalovat na facebook. Některé aplikace slouží jako prázdné plátno, kam si pomocí online editoru lze vložit vlastní obsah. Jiné mohou být widgety pro různé webové služby, jako například youtube a nebo twitter. Další skupinou jsou aplikace, které přináší již kompletní řešení a to například Soutěže [9].

Facebook vývoj pro svoji platformu silně podporuje a nabízí vývojářům mnoho nástrojů. Jedním z nich je proces přihlašování, díky kterému se uživatel může přihlásit do aplikací třetích stran pomocí svého Facebook účtu. Pokud uživatel povolí, může aplikace číst informace z uživatelského účtu, nebo s ním manipulovat. Aplikace s Facebookem komunikují pomocí http dotazů. Pro jednoduší implementaci této komunikace je pro mnoho platfo-

rem dostupné SDK. Aplikace lze obohatit o facebookové pluginy, které umožňují vložit na stránku komentáře, tlačítko Like, příspěvky a mnoho dalších prvků [5].

Facebookové aplikace stárnou daleko rychleji než běžné webové aplikace z důvodu rychle se vyvíjejícího Facebooku, na kterém jsou závislé. Tento problém již není tak závažný jako tomu bylo dříve, když aplikace teprve začínaly a Facebook neměl jasno v tom, jak tyto aplikace mají vypadat. K možnostem aplikací byl benevolentnější, avšak kvůli bezpečnosti uživatelů muselo dojít k jistým omezením.

2.2 Webové technologie

Nejprve si řekneme, z čeho se skládají webové aplikace a jaké technologie se pro ně používají. Webové aplikace jsou aplikace, které jsou dostupné z webového serveru po síti internetu. Jejich zobrazení a interakce s uživatelem probíhá ve webovém prohlížeči, který slouží jako tenký klient. Přídavek tenký, je z toho důvodu, že pro většinu aplikací platí, že velkou část své logiky řeší na serveru[13].

Celou webovou aplikaci lze tedy rozdělit do dvou části na klientskou, která pracuje v na klientském prohlížeči a serverovou, která běží na vzdáleném serveru a klient s ní komunikuje. Až na několik výjimek se technologie na obou stranách liší. Toto rozdělení je označováno jako architektura klient-server.

Pro zobrazení dokumentu (aplikace) se tedy použije webový prohlížeč. Standartním jazykem pro webové dokumenty je HTML. Dokumenty by měli mít přijemný a specifický vzhled, k tomu slouží jazyk CSS. Při větším popisu vzhledu se stává zdrojový kód náročný jak na čitelnost, tak i na modifikaci. Z toho důvodu lze využíy jazyk LESS, kterým lze dokument snáze popsat a pak si vygenerovat adekvátní CSS. Je vhodné a někdy i nutné, aby aplikace byly dynamické a dokázaly reagovat na uživatelské vstupy. K tomu slouží scriptovací jazyky JavaScript a Flash, jenž se již moc nepoužívá. Napsaný kód jazyce JavaScript může být i pro jednoduchou činnost obsáhlejší. Je ovšem možné použít framework, díky kterému lze stejnou věc napsat snáz.

Níže budou popsány jednotlivé technologie.

HyperText Markup Language (HTML) je značkovací jazyk pro vytváření hypertextových dokumentů, které jsou nezávislé na platformě. Jeho zápis vychází z jazyka SGML. Slouží pro zobrazování informací a používá se pro tvorbu webových stránek. Skládá se z množiny značek, které mohou obsahovat atributy[1].

JavaScript je objektově orientovaný interpretovaný scriptovací jazyk, používaný převážně pro webové stránky. Podporuje mnoho stylů programování. Dá se použít jak na straně klienta, tak i na straně serveru. Dokáže reagovat na interakce uživatele a dynamicky upravovat HTML dokument. Spouští se souběžně s webovou aplikací. Kód napsaný v tomto jazyce na straně klienta je viditelný. Klientský JavaScript pro kumikaci používá tzv. AJAX. Jeho interpret obsahují veškeré moderní prohlížeče. Další použití tohoto jazyka je kupříkladu vytváření rozšíření pro desktopové aplikace. [2].

CSS je jazyk určený k popisu vzhledu dokumentu napsaného v HTML, nebo XML. Jazyk popisuje jakým způsobem má být prvek vykreslen na obrazovce, vytištěn a nebo jinak interpretován. Aktuální verze jazyka je CSS3 Popis zobrazení se skládá z množiny pravidel. Tyto pravidla se skládají z dvojic selektor a deklarační blok. Deklarační blok popisuje vzhled a chování elementů, který vyhovují selektoru. CSS jde upravovat i JavaScriptem, což dokáže

[1] http://www.w3.org/html/

[2] http://www.w3.org/standards/webdesign/script

obohatit web[3].

LESS je dynamický jazyk, preprocesor jazyka CSS. Na rozdíl od CSS má více možností pro zápis, například podporuje proměnné, funkce a vnořování pravidel do sebe. Používá se dvěma způsoby, buď se předem sestaví a odesílá klientovi již v CSS, nebo je možné ho sestavit za běhu aplikace pomocí JavaScriptu[4].

Webový server je odpovědný za vyřizování HTTP požadavků, které mu přichází od klienta, ve většině případech z klientského prohlížeče. Server má přístup k databázi a jeho scripty jsou z venku nepřístupné. Klientovi na požadavek přepošle konkrétní soubor nebo dynamicky vytvoří vlastní obsah. Tyto procesy se většinou kombinují. Pro vytváření dynamického obsahu existuje mnoho technologii například PERL, PHP, nebo ASP [12].

2.3 Moderní přístup

Ze začátku webové aplikace byly stránky. Jednotlivé stránky odpovídaly určitým úkolům. Model webových stránek se postupem časů změnil z dokumentů, propojených hypertextovými odkazy na aplikační platformu. Bylo potřeba změnit celý přístup. Vznikli ajaxové dotazy, díky kterým se aplikace mohou dynamicky dotazovat na server, bez nutnosti stahovat veškerý obsah.

Události propojují interakce uživatele s vizuální reprezentací HTML. Sledují se akce uživatelů a na základě nich se přizpůsobuje uživatelské rozhraní. Události činí moderní aplikace tím čím jsou. Pomocí základní kostry HTML a událostí, se dá stavět dynimaka celé aplikace. Příkladem dynamiky může být různé přesouvání prvků a animace[8].

Model, kdy je veškerá funkcionalita webu obsažena jen v jedné stránce na straně klienta, která se dotazuje serveru se nazývá **Single Page Application**, dále jen **SPA**. Hlavní výhody tohoto přístupu je nízká kapacita přenesených dat. Oproti klasickému prolinkovanému přístupu, se složitěji řeší ukládání určitého stavu aplikace do záložky, nebo přístup pro mapující roboty vyhledávačů. U menších aplikacích lze dokonce opustit úplně od logiky zpracovávané na serveru a pouze si nechat serverem posílat statické souboru [6].

Pokud mezi sebou chtějí server s klientem v SPA komunikovat, je potřeba určit způsob, kterým si budou vyměňovat informace. Jedny z používaných přístupů jsou SOAP a REST. REST se od SOAP liší tím, že není orientován procedurálně, ale datově.

REST je architektura rozhraní pro distribuované prostředí, slouží pro přístup ke zdrojům. Zdrojem jsou stavy aplikace, nebo jimi mohou být data. REST implementuje 4 metody pro přístup k datům, které se hromadně označují CRUD (creat, read, updade, delete). Metoda creat slouží pro vytvoření dat, read pro získání, update provádí úpravu a delete data maže. Jedná se o základní operace, které jsou součástí HTML. [5].

SOAP je založen na XML a je základem webových služeb. Díky použití XML, jsou zprávy čitelné i pro člověka[6].

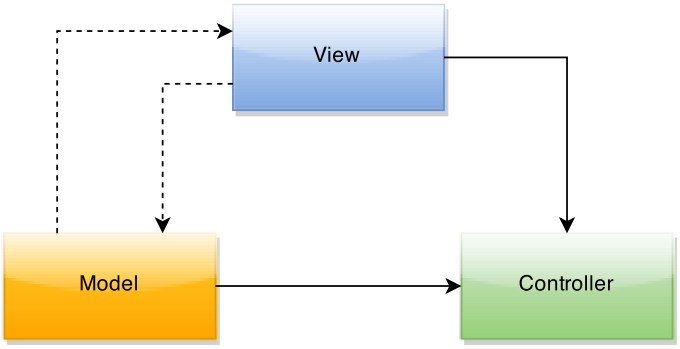
Ajax (asynchronní JavaScript a XML) je technologie pro komunikaci klientské aplikace v jazyce JavaScript se serverem. Jedná se o komunikaci kdy je vytvářen dodatečný požadavek od klientské aplikace k serveru. Jeden z požadavků může být například přístup k datům v databázi. Data, která chce server odeslat jako odpověd na dotaz, jsou nejčastěji formátováno do řetězce JSON, neboť ho JavaScript dokáže dekódovat zpět na data, kterým rozumí [8].

[3] http://www.w3.org/Style/CSS/specs
[4] http://lesscss.org/
[5] http://www.w3.org/2001/sw/wiki/REST
[6] http://www.w3.org/TR/soap/

2.4 Architektura aplikace

Proto kvalitnější architekturu aplikace extistují tzv. Architektonické vzory, díky nim je možné aplikaci lépe vyvíjet a testovat. Architektonický vzor je obecné, opakovatelně použitelné řešení běžně se vyskytujících problémů v softwarové architektuře, jsou podobné návrhovým vzorům, ale na rozdíl od nich mají širší záběr [10].

Model view controler je architektonicky vzor pro realizaci uživatelského prostředí. Pomáha například s lepším návrhem struktury aplikace, automatizovaným testováním a snažším hledání chyb. Jeho myšlenku lze pochopit velice snadno. Rozděluje architekturu aplikace do třech částí, a to model, view a controller. Model má na starost business logiku a data aplikace. View slouží jako prostředek k zobrazení dat z modelu uživateli aplikace. Controller řeší logiku aplikace a má na starost tok dat. Stará se o vzájemné propojení funkčních prvků v aplikaci. Propojení komponent lze vidět na obrázku 2.1



Obrázek 2.1: Schéma vzoru Model View Controler [4]

V aplikaci jsou pouze dva příme přístupy a to controller k modelu, aby upravil jeho data a view k modelu, aby data mohl zobrazit. Výstup aplikace má na starosti view, navíc má view na starost i odchytávání vstupú uživatele.

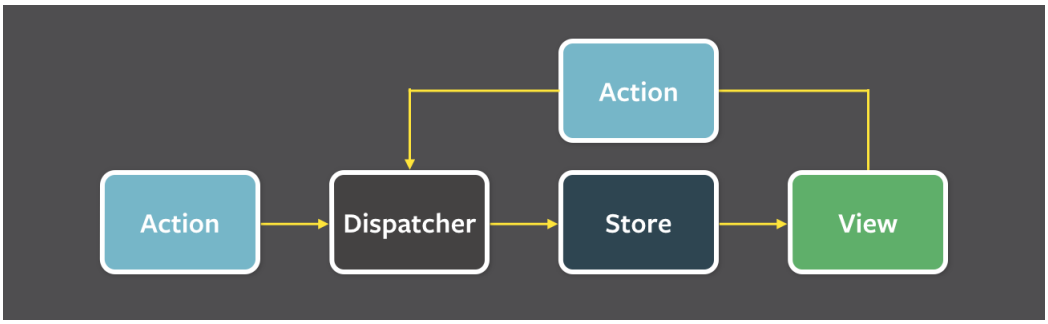
Princip fungování je následující. Uživatel provede vstup (vykoná akci) v uživatelském rozhraní. Controller tento vstup zachytí a rozhodne co s ním udělat. Nemusí udělat nic, může upravit data v modelu, anebo ovlivnit samotné view. View poté zobrazí nový pohled. Upravení dat v modelu není řešeno samotným controllerem, ten pouze pošle příkaz k provedení. Příkladem může být vynásobení dvou čísel, kdy controller pouze pošle dvě hodnoty a model provede ono vynásobení.

MVC má spousty variací, variace se používají daleko častěji, samotný MVC převážně u webových aplikací. Jediné co u variací zůstává stejný je samotný model.

Ve webových aplikací může být MVC řešeno více způsoby. První je, kdy aplikace klientský prohlížeč bere pouze jako zobrazovací zařízení a všechno ostatí řeší na serveru. Dalším typem je aplikace, která se serverem komunikuje pouze přes AJAX a velkou část logiky řeší na klientovi. Posledním typem je aplikace, která AJAX využívá jen částečně, jako doplněk pro klasický model.

První popisovaný typ má následující rozdělení logiky.Model zůstává stejný a běží na serveru. View je kód běžíčí na serveru, který se stará o generování HTML, XML, či JSON. Controllery se dělí na dva. První který má na starost odchytávání HTML požadavků, ty zpracuje a pošle konkrétnímu dalšímu controlleru, který již upraví model a prováže s view. [4].

Flux je implementace architektonického vzoru v jazyce JavaScript, který používá Facebook pro klientské aplikace. Byl vytvořen primárně pro React, který bude popsán v následující kapitole 2.5. Dá se ovšem použít i samostatně. Skládá se ze tři základních části: Dispatcher, Store a View, což se nedá zaměnit za Model View a Controller. Dalším prvkem mouhou být Action creators, které pomájí vytvářet akce pro dispatcher a popisují tak všechny možné akce, které se mohou vyskytnout v a aplikaci.



Obrázek 2.2: Schéma vzoru Flux[2]

Na rozdíl od MVC má jednosměrný tok dat. View odchytává akce uživatele, které následně přes Dispatcher šíří do Stores, jenž uchovávají data a business logiku aplikace. Stores poté zašlou nové data Views, které byly ovlivněny akcí. Dispatcher, Stores i Views jsou izolované a komunikují spolu přes vlastní vstupy a výstupy. Zprávy pro komunikaci jsou jednoduché objekty, které obsahují nové údaje a typ vlastnosti.

Dispatcher nemá žádnou vnitřní logiku, slouží pouze pro distribuci nových akcí do Stores. Každý Store se sám registruje a poskytuje zpětné volání. Když Action creators poskytně přes Dispatcher novou akci, všechny Stores v aplikaci tuto akci obdrží přes zpětné volání. Některé Store mohou čekat na jiné Store, než provedou zpětné volání.

Stores obsahují stav aplikace a její logiku. Jsou podbné jako Modely v tradičním MVC, reprezentují stav mnoha objektů, ne jenom jednoho jak je to u ORM.

View lze vytvořit za pomocí již zmíněného React, který dodáva mechanismus pro překreslování zobrazovaných informací. Zvláštním View, je tzv. View-Controller, který má na starost získávání dat ze Store a šíření je do svých potomků (View).

Action je způsob, který umožňuje vyvolávat odeslání zprávy určené pro Store. Ke zprávám kromě jejich typu, přibaluji i tzv. payload (data). Zprávy se nemusí posílat jen View, ale může tak učinit i zpětné volání při žádosti dat ze serveru[2].

2.5 Aplikační frameworky

Jak již bylo zmíněno, tak pro Single Page Application platí, že většina operací probíhá v JavaScriptu v klientském prohlížeči. Pro zrychlení vývoje a lepší udržovatelnosti a čitelnosti kódu je dobré použít některý framework pro architektonický vzor. Frameworky mohou obsahovat implementace různých návrhových vzorů a šablonovacích systémů. Mezi nejpoužívanější v dnešní době patří AngularJS a React.

AngularJS je open-source webový aplikační framework pro vytváření dynamických webových aplikací. Rozšiřuje syntaxi HTML. Je kompletně zpracováván v prohlížeči a nevyžaduje prvotní kompilaci. Řeší problémy vyskytující se při vývoji single page aplikací. Poskytuje framework pro klientskou část MVC architektury. Obsahuje mnoho komponent,

které se běžně používají v rozsáhlejších webových aplikacích. Aplikace jsou snadno testovatelné.[7].

React poskytuje knihovnu pro uživatelské rozhraní. Nediktuje, jakým způsobem má vypadat model aplikace. Drží se toho, že kód aplikace se nevkládá do HTML, jako to například dělá AngularJS, ale do JavaScriptu. Aby se jeho kód lépe zapisoval, vznikla s ním nástavba nad JavaScriptem nazývaná JSX. Sám o sobě ReactJS neposkytuje kompletní aplikační framework, ale dá se spojit spolu s jiným frameworkem ,a tím vytvořit kompletní aplikační framework. Vhodný je pro již zmíněný Flux[8].

V předchozí části byly popsané některé frameworky, pracující na klientovi. V této části budou popsány frameworky, používaná na serveru. Při vytváření větších aplikací nám serverové frameworky dokáží ušetřit čas a zkvalitnit kód. Důležitým aspektem je i bezpečnost proti útokům, neboť server pracuje s databází, jejíž informace mohou být velice citlivé.

Nette má kolem sebe obrovské množství českých vývojářů. Je doplněn speciálním ladícím nástrojem, díky kterému se dají lehce debegovat aplikace. Obsahuje v sobě šablonovací systém, nazývaný Latte. Podporuje práci s AJAXem a s MVC. Je lehce zvládnutelný na naučení. Obsahuje spoustu pluginů a rozšíření[9].

Zend je další frameworkem, jehož vývoje se účastní stovky programátorů a stovkami tisíc uživateli jej používán. Kromě MySQL, dokáže komunikovat i s dalšími druhy databázi, jako například NoSQL, Oracle, nebo PostgreSQL. Aplikace je možné testovat pomocí unit testů. Má podporu pro mnoho emailových protokolů. Je pro něj napsáno velké množství rozšiření a dokonce i šablonovacích systémů[10].

NodeJS je platforma postavená na JavaScriptovém interpretu z prohlížeče Google Chrome. Je to událostmi řídíci framework. Je velice rychlí a podporuje automatizované testy. Kód aplikace se píše v jazyce JavaScript. Kromě toho že může obsluhovat požadavky na serveru, je možné ho využit i jako interpret pro lokálně používané aplikace[11].

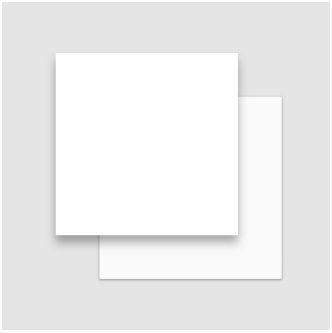
2.6 Frameworky pro uživatelské prostředí

Nesmíme zapomenout na vzhled celé aplikace, ten dokáže používání aplikaci jak zpříjemnit, tak i zhoršit. Frameworky pro layout a design stránek zajistí rychlejší a kvalitnější vývoj aplikace, pokud se použijí tak jak mají. V této podkapitole bude popsán jeden framework a jeden návrhový jazyk pro GUI.

Bootstrap je jeden z nejpoužívanějších frameworků zahrnující CSS, HTML a částečně JavaScript pro obsluhu komponent. Klade důraz na responsivní design, což je přizpůsobení se layoutu stránky dle velikost displeje. Obsahuje mnoho komponent. Dá se pomocí LESS částečně přizpůsobit konkrétním požadavkům. Nepředepisuje jak přesně má layout celého webu vypadat. Pouze dodáva již přichystané komponenty a nástroje pro snadné vytvoření layoutu. Jeho CSS kód se dělí do dvou části. První část určuje layout a základní vzhled. Další část konkretizuje vzhed prvků a může být nahrazena jinou částí, která jiným způsobem vykreslí vzhled prvků. Mezi jeho komponenty patří formulářové prvky, tabulky, modální okna, tlačítka, menu a mnoho dalšího. Jeho zdrojový kód se skládá pouze z popisu v CSS a JavaScriptu [12].

[7] https://angularjs.org/
[8] https://facebook.github.io/react/
[9] http://nette.org/
[10] http://framework.zend.com/
[11] https://nodejs.org/
[12] http://getbootstrap.com/

Material design je návrhový jazyka jehož cílem je vytvořit jeden základní systém napříč různými platformami a velikosti displejů. Kombinuje principy dobrého designu, moderních technologií a vědy. Je inspirován klasickým papírem a inkoustem. Snaží se působit, tak jak jsou lidé zvýklí z reálného života, jak je to pro ně přirozené. Provádí to formou různých animace při dotyku, vzhledem, nebo manipulací s prvkem. Prostředí, které používá je 3D prostor, ve kterém je několik vrstev. Lidé jsou na toto prostředí zvyklí z reálného světa. Vrstvy jsou osvícené zdroji světla a tvoří tak stíny. Stíny definují hranice jednotlivých prvků [13]. Na obrázku 2.3 lze vidět jak stíny vytváří hranici prvku. Material design, je využíván samotným Googlem, v jeho operačním systému Android a několika webových aplikací. Implementace tohoto jazyka je v mnoha webových frameworcích, ať už pouze čisté CSS a nebo pro již zmíněné architektury AngularJS a React. Mnoho vývojářů postupně předělávají své aplikace určené pro Android do Material designu.



Obrázek 2.3: Definice hranic prvku pomocí Material Design [3]

Material ui je CSS framework a sada komponent, který implementují Material Design. Kamponenty jsou vytvořené pro React a jejich použití je možné tedy pouze s Reactem[14].

2.7 Správce balíčků npm

V této podkapitole bude probrán správce balíčků **npm**, který je součástí platformy NodeJS. Balíčky v tomto případě budou moduly, které aplikace může využívat.

Moduly v NodeJS se dají rozdělit na dva druhy, a to předinstalované, které jsou součástí samotného NodeJS a externí, které se dají doinstalovat. Pro doinstalování slouží balíčkovací systém **npm**, díky kterému lze pomocí příkazového řádku doinstalovat balíčky. Jednotlivé moduly mohou obsahovat závislosti na jiných modulech, které se také doinstalují, bez nich by modul nebyl funkční. Moduly se instalují pomocí souboru package.json, který je obsažen v kořenovém adresáři projektu. V tomto souboru jsou sepsány základní informace o projektu, jeho verze, název a převážně moduly na kterých je závislý. U jednotlivých modulů je kromě názvu i verze, kterou aplikace vyžaduje, tím je zaručeno, že po instalaci bude vždy stejná verze. Soubory *package.json* obsahují i jednotlivé moduly. Ukázka souboru *package.json* z modulu *socket.io* [15].

[13] http://www.google.com/design/
[14] http://material-ui.com/
[15] http://socket.io

```
{
  "name": "socket.io",
  "version": "0.9.10",
  "description": "Real-time apps made cross-browser & easy with a WebSocket-
      like API",
  "homepage": "http://socket.io",
  "author": {
    "name": "Guillermo Rauch",
    "email": "guillermo@learnboost.com"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/LearnBoost/socket.io.git"
  },
  "dependencies": {
    "socket.io-client": "0.9.10",
    "redis": "0.7.2"
  },
  "devDependencies": {
    "expresso": "0.9.2",
    "should": "*"
  }
}
```

Část `dependencies` udává, na kterých modulech je aplikace či modul závislí. Pod ní vidíme část `devDependencies`, která udává balíčky důležité pro vývoj. Většinou obsahuje testovací nástroje, nebo server [7].

2.8 Existujicí řešení

Tato podkapitola se bude zabývat webovými službami, které fotosoutěž umožňují vytvořit a propagovat. Většina následujících služeb kromě samotné fotosoutěže nabízí i mnoho dalších aplikací, které je možné vložit na stránku. Tato kapitola rozebere již existující řešení fotosoutěží. Většina zmíněných stránek kromě fotosoutěží nabízí i další aplikace, které lze vložit na stránku. Ceny uvedené u soutěží jsou platné k dubnu 2015.

Photo Contest Facebook App je fotosoutěž se základním nastavením. Při zakoupení uživatel dostane aplikaci jako script, který si vloží na svůj server a ručně nakonfiguruje. Obsahuje integrované google analytics, což umožňuje velice podrobný sběr informací o aplikaci. Velkou výhodou je podpora i starších verzí prohlížeče internet explorer. Umožňuje všem účastníkům soutěže zaslat hromadný email a informovat je tak o případných výhercích. Licence se platí jednorázově a stojí 33$[16].

Brandapps nabízí kromě fotosoutěže mnoho dalších aplikací. U každé fotografie jsou sociální pluginy, které umožňují sdílet fotografii na nejpoužívanějších sociálních sítích. Umožňuje stažení získaných informací o účastnících. Licenci samostatné soutěže nelze zakoupit, je potřeba si objednat balík několika aplikací, který stojí 990 Kč při platbě na jeden měsíc. Všechny aplikaci je ovšem možné vyzkoušet na 7 dní zdarma[17].

Woobox celkově nabízí 21 aplikací. Jeho fotosoutěž může obsahovat vstupní stránku, která před samotným zobrazením fotografií může informovat o soutěži. Je možné si vytvořit formulář s vlastními poli pro nahrávání fotek a tak posbírat další informace o účastnících.

[16] http://photocontestfbapp.com/
[17] https://www.brandsapps.com/

Lze nastavit i možnost, při které soutěž stahuje fotografie z Instagramu. Aplikace si je podle hastagu dokáže vyhledat a stáhnout. Podporuje verzi pro mobilní telefony. Uživatelé, kteří vstoupí přes mobilní aplikaci, budou přesměrování na mobilní verzi soutěže. Dále má možnost nahrané fotografie nezobrazovat, dokud je administrátor neschválí. Dokáže omezit účastníky soutěže dle věku. Licence se stejně jako u Brandapps platí hromadně za více aplikací, a to 29$ při platbě na jeden měsíc[18].

Agorapulse nabízí 10 aplikací. Fotosoutěž má o něco méně možností než Woobox, stále ale nabízí rozsáhlé nastavení. Umí posílat emailem upozornění administrátorovi po nahrání nové fotografie. Cena licence 29$ při platbě na měsíc a zahrnuje několik aplikací[19].

Shrnutím lze říct, že Woobox nabízí jednoznačně nejvíce možností, zároveň za nejnižší cenu. Jeho aplikace působí moderním dojmem. Naopak Photo Contest Facebook App je velice zastaralá a možnosti má velice omezené. Všechny zmíněné aplikace mají nástroj pro překlad do libovolného jazyka. Služeb, které fotosoutěže nabízí, existují desítky. Většina z nich je ale méně kvalitních a již se neudržují.

[18] https://woobox.com/photocontests
[19] http://www.agorapulse.com/

Kapitola 3

Návrh

V následující kapitole si navrhneme výslednou aplikaci. Nejprve bude potřeba si v podkapitole 3.1 specifikovat požadavky na aplikaci a rozdělit si uživatele dle toho, jak aplikaci budou používat. Popíšeme si několik konkrétních případů užití aplikace. Poté si ji v podkapitole 3.2 logicky rozdělíme do jednotlivých částí a popíšeme si, jak tyto celky spolu budou komunikovat. Následně si navrhneme jak přibližně bude vypadat grafické uživatelské prostředí, zvlášť pro soutěž 3.3 a pro administraci 3.4. Ke konci si navrhneme datový model pro uchování stavu aplikace 3.5.

3.1 Specifikace požadavků na aplikaci

Aplikace bude umožňovat vytvořit svou vlastní fotosoutěž, spravovat ji a vložit na facebookovou stránku. Tyto fotosoutěže budou vytvářet administrátoři, kteří mají na starost správu stránky na Facebooku, pro kterou má být aplikace. Soutěžícími budou návštěvníci stránky, ze které bude aplikace přístupná. Princip soutěže spočívá v tom, že návštěvník (uživatel Facebooku) nahraje přes speciální formulář na stránce svoji fotografii. Ostatní uživatelé budou udělovat své hlasy jednotlivým fotografiím a tím rozhodnou o nejlepší z nich. Tato fotografie bude výherní. Soutěž bude probíhat vždy v určitém časovém intervalu. Z důvodů logického rozdělení na vytváření fotosoutěží a samotné fotosoutěže, si aplikaci rozdělíme dvě části a to na administraci a jednotlivé soutěže.

Administrace naší soutěže bude dostupná ze specifické URL. Uživatelé si v administraci budou moci vytvářet nové soutěže a nebo upravovat jimi již vytvořené. Instance soutěže se budou moci nastavit dle požadavků na výslednou soutěž. Nastavit půjdou následující vlastnosti.

- Název soutěže (objeví se na facebookové stránce a zároveň bude indentifikovat danou soutěž v nastavení),
- Začátek a konec soutěže (doba po kterou uživatelé budou zasílat fotografie a hlasovat pro ně),
- Informace o soutěži a pravidla soutěže (textový popis)
- Obrázek zobrazený v soutěži
- Vlastní CSS styly

Dále bude administrace umožňovat pohled na nahrané fotografie, včetně filtrování položek. Může se stát, že uživatel nahraje nevhodnou fotografii. Pro tento případ musí být v administraci možnost smazání fotografie. My ovšem tuto možnost rozšíříme i o skrytí fotografie, kterou využijeme v případě, že si nebudeme jisti zda chceme fotografii trvale smazat, ale musíme ji dočasně vyloučit ze soutěže. Aplikace bude umožňovat detailní pohled na konkrétní fotografie, ve kterém budou informace o tom, kdo fotku nahrál a kdy tak učinil. Navíc by měla být možnost zjistit, kteří uživatelé pro danou fotografii hlasovali.

Výsledná soutěž bude dostupná na facebookové stránce, takže další nezbytnou součástí aplikace musí být rozhraní pro vložení konkrétní soutěže na konkrétní stránku. Samozřejmostí je i odebrání aplikace ze stránky.

Poslední součástí aplikace by měl být pohled na uživatele, kteří se účastnili soutěže nebo hlasovali pro fotografie. Měla by být možnost si vyfiltrovat jen některé z nich.

Druhou aplikací bude již zmíněná výsledná soutěž, která bude dostupná z facebookové stránky. Tato aplikace bude přístupná pro všechny uživatele stránky. Bude zobrazovat zaslané fotografie v soutěži včetně jména uživatele, který fotografii nahrál a počtu hlasů, které fotografie dostala. U každé položky bude možnost pro její hlasování. Fotografie půjdou řadit dle počtu hlasů nebo data vložení. V aplikaci bude zároveň možnost nahrát svoji vlastní fotografii přes formulář a tím se zúčastnit soutěže. Uživatel by měl mít možnost vidět několik informací o soutěži (např. výhry) a pravidla soutěže, které musí dodržovat.

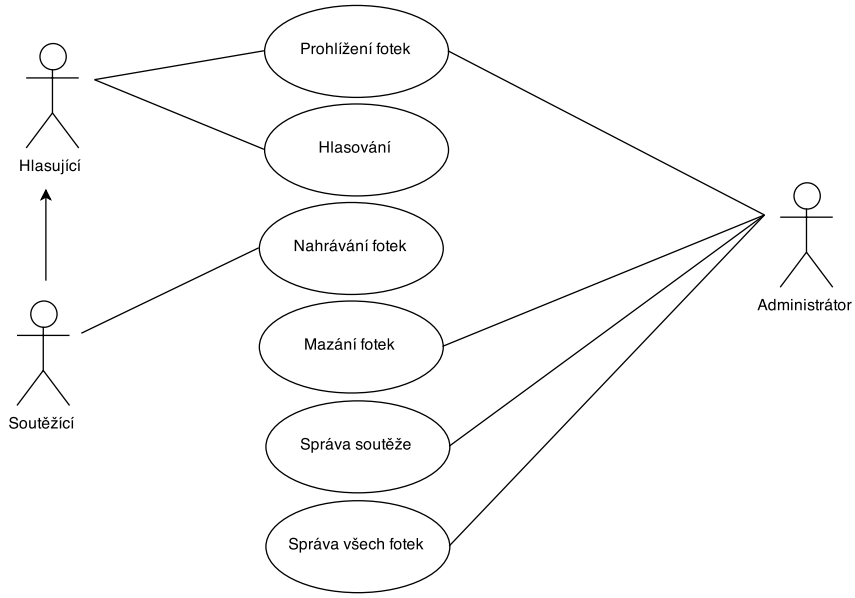
Shrnutím si lze říct, že aplikaci budou používat dva typy uživatelů. Správci facebookových stránek a návštěvnicí facebookových stránek. Návštěvník stránky může mít navíc dvě role a to, roli hlasujícího nebo soutěžícího. Přehled typů a jejich rolí je následující.

- Role správce v administraci
  - Vytvoření soutěže
  - Nastavení soutěže
  - Vložení na stránku
  - Zobrazení soutěžních fotografii
- Návštěvník stránky
  - Zobrazení soutěžních fotografií
  - Udělení svého hlasu pro soutěžní fotografii
  - Vložení jedné fotografie do soutěže

Tyto role lze ještě vidět v grafu případů užití 3.1.

Následně budou popsány konkrétní případy užití aplikace. Vstupním předpokladem je, že uživatelé mají vedený účet na Facebooku. U všech případů se uživatel autentizuje přes Facebook. Tato autentizace probíhá tak, že při kliknutí na přihlášení v aplikaci se uživateli objeví nové okno, kde potvrdí, že aplikaci uděluje právo ke čtení základních informací o jeho profilu.

První případ bude vytvoření první instance a její spuštění na Facebooku. Uživatel vstoupí do aplikace a přes přihlašovací formulář se autentizuje. Zobrazí se mu stránka s krátkým popisem, jakými kroky musí projít, aby mohl soutěž dokončit. Následně se proklikne do editace nově vytvořené soutěže. Vyplní základní nastavení (název, datum, pravidla, vlastní grafiku, css). Následně uživatel v menu vybere sekci pro vkládání na stránku.



Obrázek 3.1: UML

Zobrazí se mu tabulka, ve které vybere konkrétní stránku, na kterou si přeje soutěž vložit a on tak učiní

Dalším případem užití bude účast v soutěži. Uživatel se přihlásí přes svůj účet na stránku Facebook. Otevře si facebookobou stránku, ve které je fotosoutěž. V záložce si vybere aplikaci. Zobrazí se mu seznam nahraných fotografií od ostatních uživatelů. Otevře si okno pro nahrání fotky a přihlásí se. Následně vybere fotografii a nahraje. Nová fotografie se uživateli zobrazí.

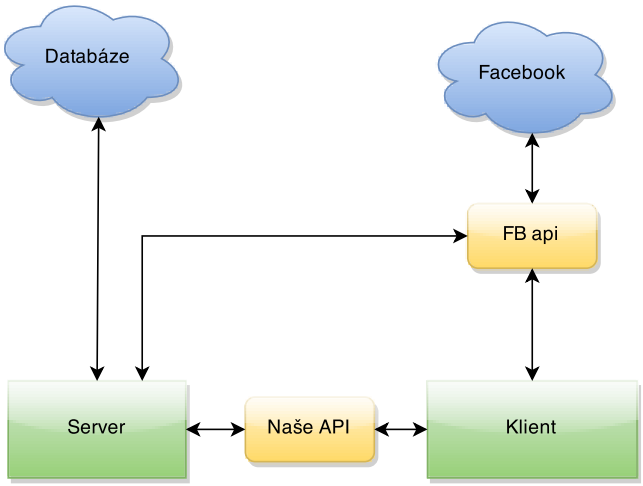
Posledním případem, který si ukážeme, bude hlasování pro konkrétní fotku v soutěži. Stejně jako v předchozím případě se uživatel přihlásí na Facebook a vstoupí do aplikace. Vybere si tu fotografii, které chce udělit svůj hlas. Vyhledá konkrétní fotografii (pomoct mu k tomu může řadící prvek). Po kliku se mu fotografie otevře v novém okně. Stejně jako v předchozích krocích se přihlásí. Po přihlášení se u překreslí tlačítko na Hlasovat. Kliknutím udělí hlas. Poté se mu zobrazí informace, že již hlasoval pro danou fotografii.

3.2 Struktura aplikace

Pro lepší rychlost, dynamičnost a menší náročnost na přenos dat, budou aplikace vytvořeny jako SPA viz. 2.2 . Tyto aplikace budou potřebovat server, se kterým si budou vyměňovat data. Server bude oddělený od podaplikací a pro komunikaci s ním bude sloužit patřičné rozhraní. Propojení jednotlivých částí lze vidět na obrázky 3.2. Výhodou tohoto rozdělení je i to, že části jsou na sobě téměř nezávislé a lze je tedy lehce měnit. Dalším důvodem pro rozdělení je i možnost napojení cizích aplikací. Příkladem je webová stránka, která může svým vlastním způsobem zobrazovat soutěžní fotografie. To lze učinit tak, že si přes rozhraní zažádá server o fotografie k dané soutěži. Sever tedy může sloužit i jako služba.

Abychom uživatele od sebe mohli odlišit a indetifikovat, je potřeba integrovat autentizaci (přihlašování). Vzhledem k tomu, že aplikace je určená pro Facebook, využijeme možnosti použít jeho nástroje, který nám tento proces ulehčí. Pro nás to bude znamenat to, že naše aplikace budou muset komunikovat se samotným Facebookem viz. 3.2, který nám bude

sdělovat, zda je uživatel přihlášen, spolu s informacemi o jeho účtu. Podaplikace jsou tedy klienti, kteří se dotazují serveru.



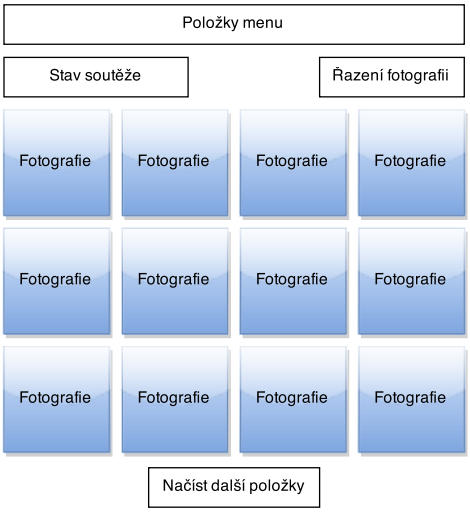
Obrázek 3.2: Služby

3.3 Návrh uživatelského rozhraní (Soutěž)

Velice důležitou částí je správný návrh grafického uživatelského prostředí. Pro uživatele musí být lehce srozumitelné a intuitivní. V této části si navrhneme, z jakých částí se prostředí bude skládat a kam umístíme jednotlivé prvky.

První aplikací, jejíž uživatelské prostředí si navrhneme, bude vytvořená fotosoutěž. Ta bude vložena v záložce stránky na Facebooku, takže bude mít jisté omezení, například její šířka bude pevně daná, kterou předepisuje Facebook a to pouhých 810px. Zároveň s aplikací bude uživateli zobrazen i okolní obsah Facebooku. Jak už z názvu Fotosoutěž vyplývá, největší důraz budeme klást na zobrazení fotek. Při vstupu do aplikace bychom tedy měli vidět jejich seznam. Fotografií chceme zobrazit co nejvíce, takže vytvoříme pouze náhledy s krátkým popiskem. Musíme počítat s možností, že fotografií může být klidně i 50 a proto nemůžeme všechny zobrazit zároveň. Klasicky by se tento problém řešil listováním v seznamu. My ovšem použijeme tzv. lazyloading (postupné dočítání fotografií), pomůže nám s ním tlačítko *Načíst další*, které nám pod současně zobrazené fotografie načte několik dalších. Výhodou tohoto přístupu je, že stránka působí dynamičtěji a uživatel nepřichází o předchozí načtený obsah. Pouze náhled k fotografii nestačí, uživatel ji bude chtít vidět celou a proto po kliknutí na ní musíme zvětšit.

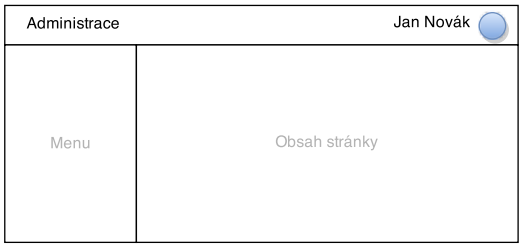
Uživatelé musí být schopni si najít konkrétní fotografii a my jim v tom ulehčíme tím, že jim umožníme si změnit řazení fotek dle určitého kritéria. Zároveň by bylo příjemné vidět, v jakém stádiu je soutěž . Posledním prvkem bude menu, které bude sloužit k otevření formuláře pro nahrávání, nebo přečtení informací o soutěži, nebo pravidel. Menu umístíme jak je zvykem na vršek aplikace, aby nám zbytečně nebralo místo. Pod něj umístíme informace o hře a řazení fotografii. Zbytek místa už budou zabírat fotografie viz náhled. Z důvodů, abychom nemuseli při otevírání detailu fotografie, nebo konkrétní stránky překreslovat celý obsah, zavedeme modální okno. Toto okno se otevře jako další vrstva nad aplikací a částečně překryje seznam fotografií. Rozdělení soutěže lze vidět na obrázku 3.4.



Obrázek 3.3: Wirefame soutěže

3.4 Návrh uživatelského rozhraní (Administrace)

Administrace bude složitější než samotná soutěž. Prvků v menu bude více a menu jako takové se bude častěji používat, protože se pomocí něj bude přecházet na jednotlivé stránky. Menu si tedy umístíme do levé části webu, kde bude fixně po celou dobu. Nad menu vložíme panel ve kterém budou informace o přihlášeném uživateli, možnost odhlášení a jeho obrázek. Ve zbylém místě bude obsah konkrétní stránky viz. obrázek 3.4.



Obrázek 3.4: Wireframe administrace

Administraci rozdělíme do několika stránek. První kterou uživatel uvidí bude **přihlašování** z této stránky bude odebráno menu, protože nebude mít pro nepřihlášeného uživatele význam.

Další stránka bude sloužit pro **vybrání konkrétní instance** aplikace, nebo vytvoření nové instance aplikace, takže bude obsahovat pouze seznam již vytvořených a tlačítko pro vytvoření nové. I v této stránce zatím nepotřebujeme menu, takže se nebude zobrazovat.

První co uživatel bude chtít udělat při vytvoření nové instance bude jí nastavit, jedna ze stránek proto bude sloužit pro **nastavení**. Bude se jednat o formulář jehož prvkami bude nastavení soutěže, který bude mít na konci potvrzovací tlačítko pro uložení.

Administrátor musí mít **přehled o fotografiích**, k tomu bude sloužit další stránka která bude zobrazovat jejich seznam s krátkým popiskem stejně jako soutěž vložená na stránce. Pro lepší procházení přidáme filtr a možnost řazení dle kritéria. Filtr nám bude sloužit pro rozlišení viditelných, zmazaných, nebo skrytých fotografií.

Po rozkliknutí fotografie v přehledu fotografií, se otevře nová stránka s **detailem fotografie**. Ta bude obsahovat větší rozměr fotografie a podrobnější informace. Chybět nesmí prvky pro smazání a nebo skrytí. Smazání je nevratný proces, takže k němu bude tlačítko. Skrytí se dá vypínat a zapínat, proto pro něj použijeme prvek typu přepínač, který uživatele navede k tomu, že se jedná o stav, který lze opakovaně měnit. Pod fotku si umístíme tabulku, ve které bude seznam všech uživatelů, kteří pro fotografii hlasovali.

Vkládání soutěže na stránku bude další část aplikace, kterou budeme potřebovat. Uživatel bude pouze vybírat na kterou stránku aplikaci přidá. Bude se jednat pouze o tabulku, kde každý řádek bude právě jedna stránku a uživatel si kliknutím na tlačítko na patřičném řádku vybere stránku, na kterou se aplikace nahraje.

Poslední stránkou bude **přehled všech uživatelů** v soutěži. Obsahovat bude seznam uživatelů v tabulce, nad kterým bude filtr pro třídění uživatelů.

Několik stránek administrace bude obsahovat tabulky a ty by měli mít v celé aplikaci jednotný styl. Je tedy potřeba si je standardizovat. Každá tabulka bude obsahovat hlavičku, která bude říkat jaká data jsou v daném sloupci. Buňky v tabulce se dají zarovnat na jednu ze stran nebo vycentrovat. Celý sloupec (včetně hlavičky) v tabulce by měl mít stejný styl zarovnání. Data, která tabulka bude zobrazovat, si rozdělíme na následující (v závorce je uvedeno zarovnání)

- číslo (vpravo),
- ikonka (vlevo),
- datum (doprostřed),
- text (vlevo),
- tlačítko (vlevo)

Již jsme zmínili, že seznam fotografií budeme mít v soutěži i v administraci. Jednotlivé položky by u sebe měli mít krátký popis. Pro uživatele je nejdůležitější, aby viděl jméno uživatele, který fotografii nahrál, počet hlasů pro fotografii a pochopitelně i samotný náhled fotografie. Na stránku by se nám mělo vejít co nejvíce těchto položek. Proto budou náhledy fotografií obdélníkové výřezy jejich skutečných fotek. Díky tomu dosáhneme toho, že nebudou žádná prázdná místo kolem fotek a my je budeme moct vedle sebe naskládat. Nesmíme zapomenout na počet hlasů a jméno uživatele. Chceme, aby uživatel při pohledu poznal, že se jedná o číslo určující počet hlasů, ale zároveň k tomu nechceme přidat text, neboť opakující se text působí nepříjemně. Přidáme tedy vedle čísla ikonku, která znázorní o co se jedná. Obě informace sami o sobě taky můžou zabrat hodně místa a z toho důvodu je umístíme do fotky, přičemž musím umožnit jejich čitelnost. To docílíme přidáním průhledné vrstvy. Na obrázku 3.5 lze vidět návrh miniatury fotografie s popiskem.

3.5 Datový model

Aplikace nám musí zachovávat svůj stav a to konkrétně nastavení jednotlivých soutěží, informace o hlasování a o uživatelích. K tomu, aby jsme zachování stavu docílili, použijeme databázi, která nám bude sloužit k ukládání a přístupu informací. Před samotnou implementací si musíme navrhnout konceptuální datový model, který nám bude popisovat strukturu dat (entity) a vztahy mezi nimi, na základě něho pak budeme moct vytvořit patřičné tabulku v databázi.

Obrázek 3.5: Náhled fotografie

Z důvodů identifikace přihlášených uživatelů si vytvoříme entitní množinu **users**, která nám bude sloužit pro zachování základních informací o uživatelích. Ukládat budeme jméno, facebook id a email.

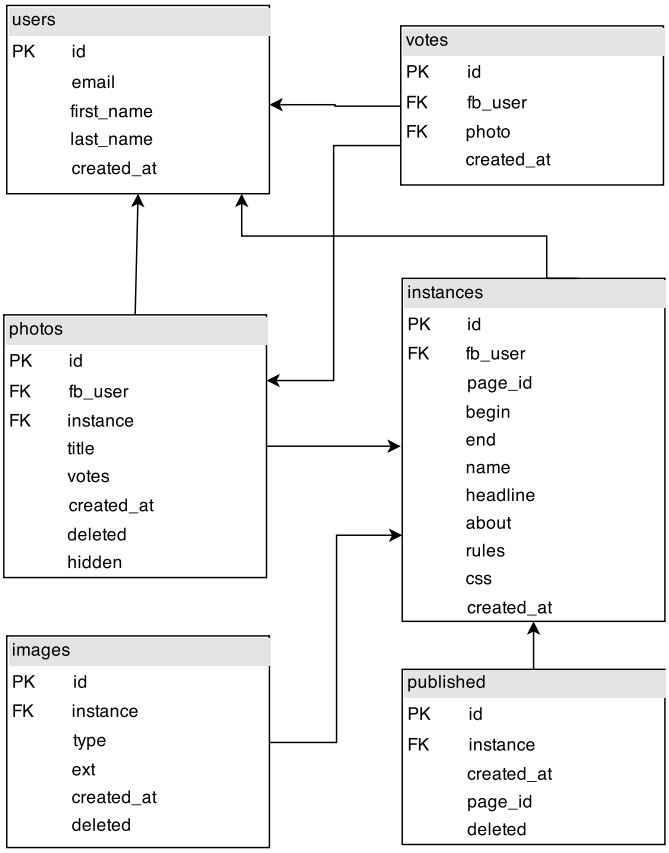
Pro instance soutěží vytvoříme entitní množinu **instances**, ve které budeme udržovat obecné informace o instancích. Soutěže budou mít svůj název, datum zahájení, datum ukončení, popis, pravidla a také mohou obsahovat vlastní CSS. Administrátoři mohou mít více instancí, ale každá instance pouze jednoho administrátora. Z tohoto důvodu přidáme vztah mezi entitní množinu Users a Instances.

Instance soutěží budou publikovány na Facebookové stránky, kteří si sami uživatele vyberou. Je tedy potřeba rozlišit na kterou stránku uživatel soutěž vložil. Pro tuto informaci si vytvoříme další entitní množinu **published**. Tato entitní množina by nemusela existovat, protože instance může být vždy jen na jedné stránce. My ji však vytvoříme z důvodů pro budoucí rozšíření, které by vkládání na více stránek mohlo umožňovat.

Administrátorům dáme možnost obohatit layout soutěže vložením vlastní grafiky (obrázky). Budeme potřebovat entitní množinu **images**, která popisuje o jaký druh obrázku se jedná a jaký má formát. Instance mohou mít více obrázků, ale každý obrázek patří k právě jedné instanci, tím nám vzniká další vztah Images s Instaces.

Soutěžící budou nahrávat k instancím soutěží fotografie. Vytvoříme další entitní množinu **Photos**. Stejný uživatel může nahrávat fotografie do různých instancí soutěží, ale do každé z nich pouze jednou. Budeme tedy potřebovat dva vztahy. Entity fotografii se vážou k uživatelům, kteří je nahráli do soutěže. Druhým vztah je k instancím soutěží do kterých byly nahrány.

Pro každou fotografii může hlasující hlasovat pouze jednou, z toho plyne že potřebujeme entitní množinu **votes**. Tato množina nám popisuje, který uživatel hlasoval pro kterou fotografii. Z toho vyplývá, že potřebujeme další vztahy mezi novou entitní množinou votes a entitími množinami users a photos. Entitní množiny, jejich vztahy a atributy popisuje entit relationship diagram na obrázku 3.6.



Obrázek 3.6: ERD l

Kapitola 4

Implementace

V poslední kapitole je popsaná implementace aplikace. Nejprve bude rozebraná struktura celé aplikace 4.1, její rozdělení a komunikace a poté budou popsány všechny kroky k tomu, aby se aplikace mohla nainstalovat. Bude následovat detailněji popsaná architektura klientů 4.2. Pro možnost napsat rozšíření v komunikaci bude vysvětlena implementace REST v podkapitole 4.3. Aplikace administrace obsahuje komponenty, které lze obecně využívat, jejich použití lze vyčíst v podkapitolách 4.5 a 4.6. Přesný postup pro přidání další stránky do administrace je rozebrán v podkapitole 4.7. Ke konci bude popsán způsob jakým probíhalo testování na uživatelích a vyhodnocení testů 4.9.

4.1 Struktura aplikace a instalace

Jak již bylo zmíněno v předešlých částech, tak se aplikace dělí na 3 části, které jsou na sobě nezávislé implementovány. Jejich vzájemnou komunikaci obstarává rozhraní implementované jako **REST**. Dvě klientské aplikace jsou napsané v JavaScriptu. Serverová aplikace je implementovaná v jazyce PHP a je pro ni použit framework **Nette**. Server jako databázi používá MySQL.

Logika klientských aplikací je rozdělena do modulů. Z důvodů obstarávání většiny logiky aplikace na straně klienta a lepší přehlednosti kódu je každý modul jako samostatný soubor. V hlavičce takového souboru je předpis závislostí na další moduly. Navíc jsou v aplikaci použity některé externí moduly od dalších vývojářů.

Klientské aplikace jsou projekty v balíčkovacím systému **npm**, který zajišťuje jejich sestavení.

Pro běh celé aplikace je potřeba jí nejprve správně nakonfigurovat, sestavit a zaregistrovat na stránkách Facebooku určené pro vývojáře. Požadavky na server jsou následující

- PHP verze 5.3.1 nebo vyšší
- MySQL
- šifrovaná komunikace HTTPS

V první řadě je potřeba aplikaci registrovat na stránkách [1]. K tomu je důležité mít aktivní profil a být registrovaný jako vývojář. V sekci *My apps*, ve které je přehled všech registrovaných aplikací. Přidání nové aplikace se provádí kliknutím na tlačítko *Add new app*,

[1] https://developers.facebook.com/

následně se z možnosti použití aplikace vybere webová stránka. Tutoriál lze přeskočit. Poté se vybere název aplikace, který je libovolný a kategorie což je v našem případě Aplikace pro stránku. Položka *namespace* je v případě naší aplikace nepotřebná. V detailu již vytvořené aplikace je potřeba správně nastavit adresu serveru. To se provede v záložce *Settings*, kde je potřeba nastavit položku *App Domains*, což jsou domény na kterých aplikace poběží. Pokud budou na doménách třetího řádu jedné mateřské domény, nastavíme pouze její adresu. Dále na stránce přidáme platformy. Pro administraci přidáme platformu *website*, protože poběží samostatně, u ní stačí nastavit pouze parametr *Site URL*, cože je url adresa stránky administrace. Dále přidáme platformu *Page tab*, u které vyplníme položku *Secure Page Tab URL*, adresou, na které bude soutěž. Poté doplníme do položky *Page Tab Name* název, který se objeví u záložky na Facebooku. Celý proces potvrdíme uložením *Save Changes*.

V záložce settings vidíme *App ID* a *App secret*. Tyto údaje se musí vložit do několika souborů v aplikacích. Přesný popis se nachází v souboru install.txt, který je umístěn ve složce se zdrojovými kódy.

Toto nastavení nám stačí pro testování, ale běžný uživatel stále k aplikaci nebude mít plný přístup. Pro plnou konfiguraci a použití musíme ještě v záložce *App Details* nastavit položku *Privacy Policy URL*, což je adresa k dokumentu zásad ochrany osobních údajů z důvodů přístupu k osobním údajům uživatelů. Poslední záložkou, se kterou budeme pracovat, je *Status & Review*, kde přepneme aplikaci do stavu k veřejnému používání.

Pro klientské části je nejprve potřeba stáhnout externí moduly, v kořenové složce těchto aplikací se provádí příkazem *npm install*. Následně již stačí aplikaci přeložit. Pro překlad, určený na produkční server, je příkaz *npm build* jenž provede i kompilaci. Pro vývoj slouží příkaz npm start, který zároveň spustí lokální http server [2]. Poté hlídá změny v souborech, na které reaguje novým sestavením. Díky tomu je zajištěn snadnější vývoj. Na produkční server se nahrávají po sestavení následující adresáře a složky

- index.html / index.php
- js/bundle.js
- fonts/
- img/

Pro vývoj, se nahrávají veškeré soubory, pro možnost snadnějšího debugování.

Serverová část se na produkční server nahrává kompletně celá. Je jen potřeba vytvořit patřičné tabulky v databázi, vytvoření se realizuje SQL scriptem.

4.2 Architektura klientů

Klientské aplikace jako architektonický vzor používají **Flux**. Tomuto rozdělení odpovídá i adresářová struktura aplikací.

Pohledy se nachází v adresáři */js/components* a mají příponu react vzhledem k implementací pomoci knihovny react. V administraci jsou opakující se komponenty a proto je doplněn další adresř */componentes/common/*, kde jsou k dispozici komponenty, které se obecně používají na více stránkách.

[2] http://localhost:8080

Pro komunikaci s externími zdroji se v adresáři */js/utils/* nachází soubory s příponou *API*. Tyto soubory obsahují moduly, které na žádost pohledů posílají ajaxové dotazy, ať už serveru aplikace nebo Facebooku.

Akce posílají zprávy z pohledů přes dispečera obchodům. Nachází se v adresáři */js/actions/* a jsou rozděleny podle typu události na události směřující k Facebooku, Serveru a nebo aplikace sama k sobě. Tyto akce jsou vyvolávány dvěma způsoby. Základní způsob je, že přímo pohled vytvoří akci. Druhým způsob se uplatňuje u ajaxových požadavků, kdy pohled zavolá patřičné API, které zašle ajaxový dotaz. Poté při zpracovávání odpovědi se dle jejího výsledku vytvoří akce.

Obchody se nachází v adresáři */js/stores/* a jsou logicky rozděleny podle typu stavu který uchovávají.

Dispečeři se nachází v souboru */js/dispatcher/AppDispatcher.js* v případě složitějšího klienta pro administraci jsou 3, kteří logicky odděluji akce které jsou vyvolány pohledem, serverem, nebo Facebookem. Klient pro aplikaci používá pouze jeden. ú

4.3 REST

Klientské aplikace mezi sebou komunikují přes protokol REST. Server dokáže pracovat s více verzemi zároveň, a to podle URL, kterým je požadavek volán. Dále je potřeba určit metodu. Požadavek se tedy skládá z následujících částí

- adresa serveru
- verze implementace
- zdroj dat
- potřebné parametry
- metoda GET, POST, DELETE, UPDATE

Následující příklad ukazuje volání pro verzi *v1*, kde žádá seznam fotografií pro instanci, jejíž *ID* je 5. Server je umístěn na adrese *https://api.server.com/*. Výsledkem tedy je adresa *https://api.server.com/v1/photos/5* s metodou *GET*. Tento požadavek vrátí seznam fotografií ve formátu JSON.

Běžně se typ žádosti určuje podle typu v HTTP hlavičce, avšak všechny prohlížeče s tímto neumí plně pracovat a proto je možné všechny žádosti poslat v HTTP hlavičce typem *POST* a následně přidat parametr *method*, který nabývá hodnot *GET, POST, DELETE*, nebo *update*. Server si to interpretuje stejně, jako by metoda byla přímo v HTTP hlavičce.

Verze implementace požadavků jsou na serveru v adresáři */app/*, kde další adresář určuje verzi, v případě implementované verze *v1*, je to adresář */app/v1Module/*.

Zpracovávání logiky požadavků je v případě aplikace rozděleno do dvou částí, *pressenteru* a *modelů*. Dle url, na které byl požadavek zavolán, se vybere patřičný pressenter. Dále podle metody CRUD (create, read, update, delete) se vybere odpovídající metoda v pressenteru. Tyto metody nemají přímý přístup k datům v databázi, ale volají modely, které jim ho zprostředkovávají. V aplikaci jsou dva modely, konkrétně `Facebook` a `PhotoContent`, nachází se v adresáři */app/model/*. Model *Facebook* obsahuje metody pro práci se službami Facebooku. `PhotoContent` slouží pro prací se samotnou soutěží a sám také využívá model `Facebook`.

Model `Facebook` má pouze dvě veřejné metody. Jedna z nich `setUserFromAccesToken()`, která dle access tokenu zjistí a nastaví uživatele. Další metodou je `getUser()`, která vrátí Facebook ID uživatele.

Model `PhotoContent` má mnoho veřejných metod, je zbytečné všechny popisovat. Slouží například pro zjištění, zda uživatel hlasoval, získání fotek k patřičné instanci, nebo skrytí obrázku v soutěži.

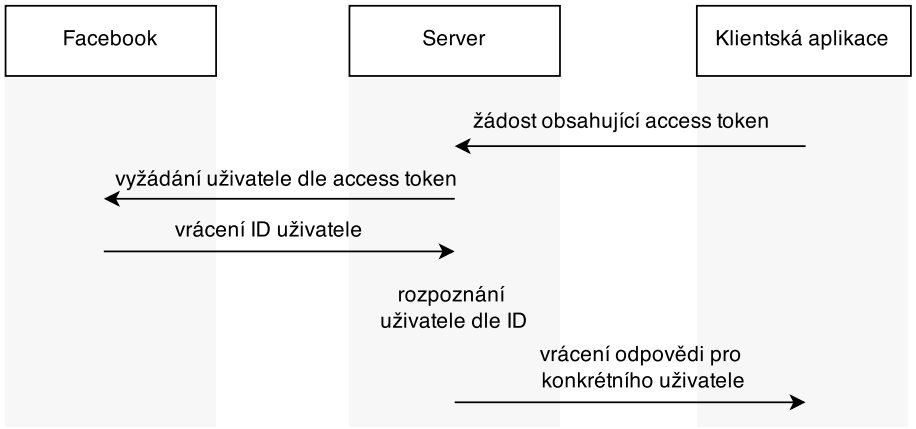
4.4 Autentizace uživatelů

Autentizace je důležitou součástí aplikace a je implementovaná ve všech třech částech. Jak již bylo v návrhu uvedeno, je použit nástroj od Facebooku, který nás oprošťuje od řešení registrace, zapomenutého hesla a mnoho dalších problémů spojených s autentizací. Komunikace s Facebookem probíhá pomocí oficiálního API od Facebooku.

Přihlašování probíhá následovně. Uživatel přes aplikaci zažádá o přihlášení. Aplikace žádost přepošle facebookovému API, které vytvoří přihlašovací okno pro uživatele. Jakmile uživatel přihlášení potvrdí, facebookové API vrátí *access token*. Tento *access token* si klientská aplikace uloží a je použitelný pro konkrétního přihlášeného uživatele. Při žádosti například o jméno uživatele, na které se aplikace dotazuje přímo Facebooku, tento *access token* musí být obsažen v žádosti. Facebook na základě *access tokenu* pozná zda je přihlášení platné a o kterého uživatele se jedná.

Při komunikaci se serverem (pouze při té, u které je vyžadována autentizace), se stejně, jako s Facebookem, posílá *access token*. Server poté, aby ověřil pravost uživatele, se sám zeptá Facebooku přes jeho API, o jakého uživatele se jedná a nechá si potvrdit jeho pravost, případně zaslat některé informace o uživateli.

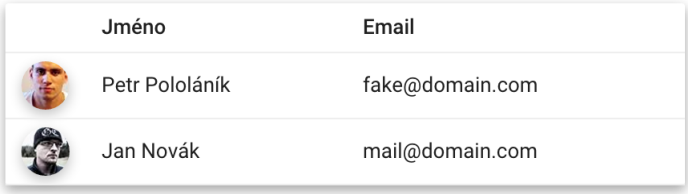
Access token se neukládá do databáze a nedá se dle něj dlouhodobě identifikovat uživatel, protože se při každém novém přihlášení mění a po určité době jeho platnost zaniká. Avšak každý uživatel má svůj veřejný identifikátor na Facebooku, který lze díky *access tokenu* získat. Tento identifikátor je použit i v databázi a díky tomu server identifikuje uživatele. Autentizaci lze vidět na obrázku 4.1.



Obrázek 4.1: Identifikace uživatele dle access token

4.5 Komponenta pro tabulku

V této části bude popsáno, jakým způsobem lze vytvořit v aplikaci tabulku, pomocí obecně používané komponenty. Její implementace je v souboru */js/components/common/Table.react.js*. V návrhu aplikace 3.3 již bylo navrhnuto rozdělení prezentované informace do různých typů číslo, text, datum, ikona a tlačítko. Typ obrázek byl implementován specifičtěji a to na obrázek uživatele, nebo stránky naFacebooku. V následujících krocích bude popsáno jak lze vytvořit tabulku viditelnou na obrázku 4.2.



Obrázek 4.2: Příklad tabulky

Tabulka je rozdělena do dvou části, a to definice dat, které budou zobrazené a samotná data. Její vytvoření se zavolá kódem `<Table data=tableData>`, `tableData` je objekt, který obsahuje předpis pro vykreslení. Předpis se skládá ze 3 částí. Definice zobrazených sloupců, samotná data a stav, zda je tabulka viditelná nebo nikoliv. Následující kód ukazuje data pro tabulku, avšak bez obsahu tabulky, kvůli lepší čitelnosti a pochopení.

```
{
 head: [
  {
    type: TableTypes.FB_PICTURE
   }, {
    name: 'Jméno',
    type: TableTypes.TEXT
   }, {
    name: 'Email',
    type: TableTypes.TEXT
   }
  ],
  content: content,
  status: true
}
```

V popisu tabulky lze vyčíst definici čtyřech sloupců. Každý z nich je definován typem a názvem, přičemž název je nepovinný. Konstanty pro typy se nachází v souboru, ve kterém jsou všechny konstanty použité v aplikaci */js/constants/AdminConstants.js*. Obsah tabulky pro příklad vypadá následovně:

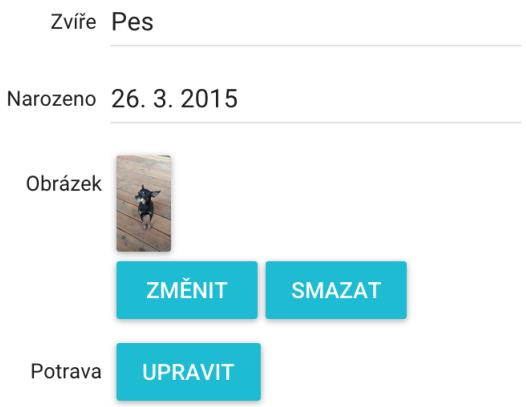
```
[
  [
   12345678, "Petr Pololáník", "fake@domain.com"
  ],[
   12345678, "Jan Novák", "mail@domain.com"
  ]
]
```

Sloupec typu `FB_PICTURE` očekává pouze ID obrázku, v našem případě doplněné 12345678. Typ `TEXT` zobrazí hodnotu řetězce, tak jak ji dostane. V příkladu nelze vidět typ tabulky

button (BUTTON), číslo (NUMBER) a datum (DATE). Number očekává klasické číslo a datum, objekt typu `date`. Tlačítko je složitější a očekává objekt, který obsahuje položky `label` (popisek tlačítka), `primary` (boolean zda je tlačítko primární) a `onClick` (funkce volaná po kliknutí).

4.6 Komponenta pro formulář

V teto podkapitole bude popsané použití komponenty, která vykresluje skupinu prvků formuláře. V administraci je použita na stránce editace instance a její implementace je v souborech */js/components/common/GroupItem.react.js*, */js/components/common/ImageInput.react.js* a */js/components/common/Input.react.js*. Vykreslení probíhá přes modul `GroupItem` a vypadá následovně:



Obrázek 4.3: Vykreslený formulář pomocí komponenty

Komponenta obsahuje tři parametry, `disabled`, `items` a `ref`. Paremetr `disabled` očekává *boolean*, který určuje zda je formulář aktivní a dají se položky upravovat, či nikoli. Druhý parametr `items` očekává pole, jehož položky definují prvky formuláře. Parametr `ref` slouží jako reference, pro následné čtení. Prvky mohou být těchto typů

- text (text)
- datum (date)
- výběr obrázku (image)
- tlačítko button

Následující zdrojový kód popisuje definici formulářových prvků všech čtyřech typů:

```
{label: 'Zvíře', name: 'animal', value: 'Pes', type: 'text'},
{label: 'Narozen', name: 'birth', value: birthDate, type: 'date'},
{label: 'Obrázek', name: 'img', value: dog.jpg, type: 'image'},
{label: 'Potrava', Upravit: 'Popisek', type: 'button', onClick : edit}
```

U všech typů položek lze definovat popisek, který se zobrazuje vlevo od položky. `name` slouží pro identifikaci při získávání hodnot, pouze u typu submit udává text v tlačítku.

U textu, data i obrázku udává `value` výchozí hodnotu. Poslední parametr je u tlačítka `onClick`, který je určen po funkci, která se zavolá po kliknutí na tlačítko.

Hodnoty lze získat přes metodu `getValue()`. Která vrátí hodnoty jednotlivých položek, kromě *tlačítka*, které nenastavuje žádnou hodnotu. Typ *obrázek*, na rozdíl od ostatních, nevrací stejnou hodnotu, kterou byl nastaven ale objekt který má dvě vlastnosti. První vlastnost `isChange` říká, zda došlo ke změně (odstranění nebo přepsání), další vlastnost `file` už odkazuje na konkrétní vybraný nový obrázek a nebo na nic, pokud nebyl žádný vybrán, nebo smazán výchozí. Odlišné řešení u *obrázku* je z toho důvodu, aby se při každém přeuložení neposílal nezměněný obrázek na server a nezpomaloval tak požadavek vzhledem ke své velikosti.

4.7 Rozšíření administrace

V této podkapitole si popíšeme, jak lze přidat další stránku do aplikace, čímž lze zároveň podrobněji pochopit strukturu aplikace. Je možné na nové stránce využít komponenty popsané v předchozích kapitolách.

Nejprve je potřeba vytvořit novou komponentu, která bude zastupovat stránku. V příkladu ji budeme nazývat *example*. Komponentu vytvoříme v adresáři */js/components/* a pojmenujeme ji *Example.react.js*. Pro správně routování komponenty na URL v souboru */js/routes.js* v hlavičce přidáme závislost na nové komponentě. Dále je v souboru potřeba přidat nový `<Route/>` a správně zanořit. Pokud se jedná o stránku, která má pracovat s instancí, zanoříme ji do `<Route name="admin»` v ostatních případech přidáme `<Route/>` do rodičovského. V příkladě je výňatek ze souboru s příkladem vložení do rodičovského `<Route/>`.

```
var First = require('./components/First.react');
var MyExample = require('./components/Example.react'); // nový řádek
var routes = (
  <Route handler={App}>
    <Route name="login" handler={Login}/>
    <Route name="init" handler={Init}/>
    <Route name="first" handler={First}/>
    <Route name="admin" path="admin/:instanceID/" handler={Admin}>
      <Route name="edit" path="edit/" handler={Edit} />
      <Route name="photos" path="photos/" handler={Photos}>
        <DefaultRoute handler={PhotosList} />
        <Route name="photoDetail" path="photoDetail/:photoID" handler={
            PhotoDetail}/>
      </Route>
      <Route name="dashboard" path="dashboard" handler={Dashboard} />
      <Route name="publish" path="publish" handler={Publish} />
      <Route name="example" path="example" handler={myExample} /> {/* nový
          řádek */}
    </Route>
    <DefaultRoute name="pick" handler={Pick} />
  </Route>
);
```

Parametr `name` udává název, kterým se bude routovat na danou stránku, `path` popisuje url adresu stránky a `handler` komponentu, která se má vykreslit.

```
  { route: 'example', text: 'Název položky v menu'}
```

Další části je vytvoření odpovídající komponenty, kterou chceme zobrazit na stránce. Následující zdrojový kód popisuje komponentu která neobsahuje levé menu.

```
var React = require('react');
var mui = require('material-ui');
var Authentication = require('../utils/Authentication')
var PageFull = require('./PageFull.react');

var Example = React.createClass({

  mixins: [ Authentication ], // tento řádek je volitelný a zajišťuje
      potřebu přihlášení

  render: function () {
    return (
      <PageFull headline="Ukázková stránka">
       Obsah stránky
      </PageFull>
    );
  }
});

module.exports = Example;
```

Pokud má stránka postranní menu `render` bude na rozdíl od předchozího kódu vracet následující kód

```
  render: function () {
    return (
     <div>
      <h2>Ukázková stránka</h2>
       Obsah stránky
     </div>
    );
  }
```

Pokud je vyžadováno mít odkaz na stránku jako položku v menu, tak je potřeba v souboru */js/components/Admin.react.js* do objektu `menuItems` přidat další řádek.

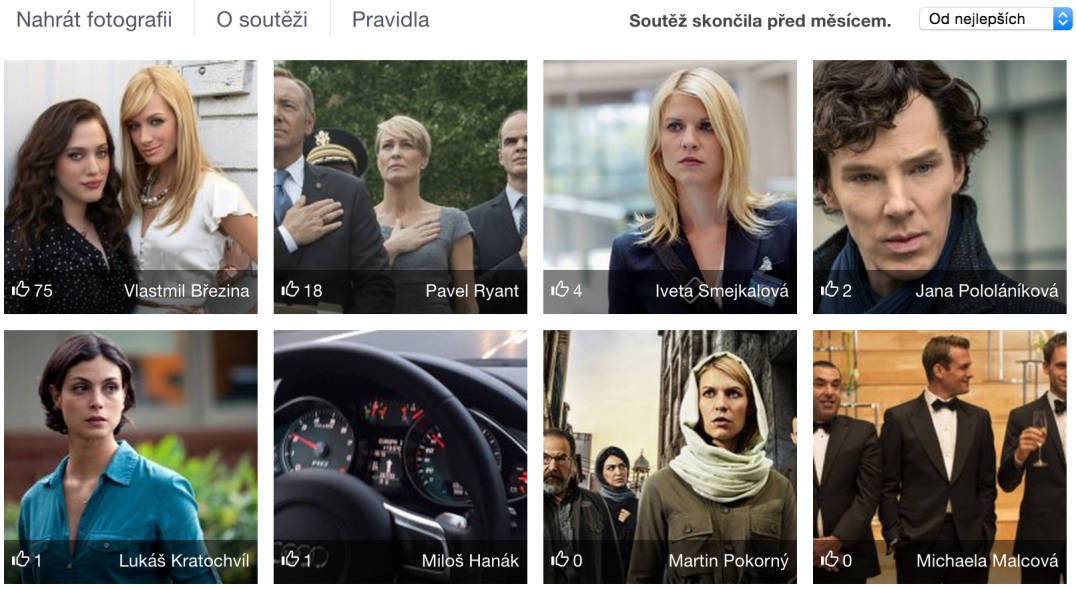
```
  { route: 'example', text: 'Název odkazu'},
```

Většina stránek v administraci zobrazují data, o které si musí nejprve požádat server nebo Facebook, což trvá určitou chvíli. Aby uživatel věděl, že aplikace pracuje, tak v horní části stránky zobrazuje pruh, který znázorňuje načítání a zároveň se změní kursor myši.

Rozpoznání, zda se čeká na data, má na starost `UpdateStore`. Při každém novém požadavku na data se zavolá metoda `UpdateStore.addActions()`, což inkrementuje proměnnou, která určuje zda se čeká, nebo ne. Pokud je proměnná rovna 0, všechna potřebná data jsou k dispozici. Pokud je větší než 0, znamená to, že se na data čeká. Následným přijetím dat se zavolá `UpdateStore.removeActions()`, který proměnnou dekrementuje. Pokud se čeká na více zdrojů dat, je možné přidat při volání metody `UpdateStore.addActions(numner)` parametr typu číslo, který proměnnou inkrementuje vícekrát. Obdobně funguje i metoda `UpdateStore.removeActions(number)`.

4.8 Řazení fotografií

V této podkapitole si probereme jakým způsobem je implementováno řazení fotografií v soutěži. Náhled implementovaného řazení lze vidět na obrázku 4.4.



Obrázek 4.4: Náhled soutěžních fotografií

V aplikaci má uživatel možnost řadit fotografie dle třech kritérií . Podle hlasů, data vložení nebo názvu. Toto řazení je k dispozici, jak v administraci, tak i v soutěžní aplikaci. Pro lepší požitek z aplikace je řazení v části soutěže doplněno o plynulou změnu pozic fotografií.

Tato vlastnost je implementovaná pomocí CSS3 a JavaScriptu. Jednotlivé náhledy jsou pozicovány absolutní hodnotou k elementu, který je obaluje, pomocí vlastností `top` a `left`. Při změně řazení se spočítají nové pozice náhledů a přepíší se za aktuální. Výňatek z kódu ze souboru */js/stores/PhotosStore.js*, má na starost výpočet nové pozice.

```
itemsFake.map(function (itemFake, indexFake) {
            _items.map(function (item, index) {
                itemFake.visible = indexFake < _visibleItems ? true : false;
                if (itemFake.id === item.id && itemFake.visible) {
                    item.posX = parseInt(indexFake / _itemsOnRow);
                    item.posY = indexFake % _itemsOnRow;
                }
            })
        });
```

`itemsFake` je pole, které obsahuje staré nastavení miniatur, `_items` je jeho kopie, která bude vrácena funkcí s nastavením nových pozic. Vlastnosti `posX` a `posY` jsou pouze indexy určující na kterém řádku a ve kterém sloupci se miniatura zobrazí. Vlastnost `visible` říká, zda je fotografie zobrazena, nebo skryta.

Díky vlastnosti `transition`, kterou mají náhledy, ke změně dojde plynule, nikoliv skokově. V případě, že uživatel vidí jen omezený počet fotografií, může dojít k tomu, že některé náhledy změně řazení nemají být vidět a jiné zase ano. Řešením tohoto problému je, že se miniatuře, která se má skrýt, nastaví vlastnost `opacity` průhlednost na hodnotu 0. Díky této vlastnosti společně s vlastností `transition` dojde postupnému zaniknutí. Opačným způsobem se fotografie, která neměla být vidět zobrazí. Společně s vlastností `opacity`, je potřeba změnit i vlastnost `z-index`, aby viditelné fotografie v popředí.

Kód v souboru */js/components/Photos.react.js*, který překresluje nové pozice, vypadá

následovně

```
var style = {
 top: (item.posX || 0) * PHOTO_SIZE,
 left: (item.posY || 0) * PHOTO_SIZE,
 zIndex: item.visible ? 1 : 0,
 opacity: item.visible ? 1 : 0,
}
```

Přičemž konstanta `PHOTO_SIZE` udává velikost výsledných náhledů.

V ukázce implementace tohoto řešení lze i vidět rozdělení logických celků, kdy komponenta počítá konkrétní pozici pro vykreslení, ale neřeší na jakým řádku nebo sloupci má náhled být. Řádek a sloupec si komponenta získá z obchodu *Photos*. Výpočet indexů je sice v souboru */js/utils/ItemsHelper.js* a né v souboru */js/stores/PhotosStore.js*, ale to je pouze z důvodů lepší přehlednosti, kdy *obchod* patřičnou funkci volá jako pomocnou.

4.9 Uživatelské testy

V této podkapitole budou popsány testy, které byly provedeny na uživatelích aplikace. Vzhledem k tomu, že aplikace jsou dvě, budou i testy rozděleny dle těchto částí tedy na testy administrace a výsledných soutěží. V první řadě bude popsán návrh. Poté budou prezentovány výsledky, včetně jejich zhodnocení.

Pro testování byl vytvořen testovací protokol. Ten nejprve popisuje, jakým způsobem musí být aplikace natavena, aby testování mohlo proběhnout. Ze začátku jsou základní informace o testovaném uživateli (pohlaví, věk, vzdělání). Následuje seznam úkolů, který musí uživatel provést. Ke konci jsou otevřené otázky k aplikaci, které mají za úkol zjistit, které prvky v aplikaci schází a které jsou naopak přebytečné.

Z důvodu, že je aplikace určena pouze pro uživatele Facebooku, byly i pro test vybráni pouze tito uživatelé. Konkrétněji uživatelé, kteří aktivně využívají Facebook (minimálně dvakrát týdně). Při testování uživatelů jsem vždy nejprve inicializoval aplikaci. Účastníci plnili jednotlivé úkoly a já jsem stopoval čas, který byl potřeba pro jejich zhotovení. Protokol jsem kompletně plnil za ně, abych si mohl dopisovat i případné poznámky k úkolům.

Pro soutěž jsem navrhl protokol o 4 úkolech, kdy uživatel poprvé přistupuje k soutěži. Administrace je složitější a z toho důvodu jsem vytvořil dva protokoly, který plní stejný uživatel. První protokol je určen pro testování prvního přístupu do administrace, vytvoření nové soutěže a její publikování. Druhý protokol má naopak zjistit chování uživatele, když je v soutěži již několik nahraných fotek, tedy průběh soutěže.

Testovací uživatelé se pohybovali ve věku 18 až 34 let. Z testů nelze jednoznačně říct, že by měl věk, pohlaví, nebo vzdělání vliv na rychlost plnění úkolů. S největší pravděpodobností záleží na počítačové gramotnosti uživatelů, kterou běžní aktivní uživatelé Facebooku maji dostatečnou.

Doby plnění úkolů v aplikaci soutěž popisuje tabulka 4.9. Z testů této aplikace vyplynulo, že uživatelé jsou zvyklí na symbol "To se mi líbí"jako na tlačítko, proto při hlasování pro fotku klikali na tento symbol. Symbol pouze naznačuje, že číslo vedle něj urxčuje počet hlasů. To že se otevřelo modální okno, způsobilo pouze překvapení. Uživatelům scházela možnost sdílení fotky prostřednictvím Facebooku. Naopak nadbytečné jim přišlo tlačítko Odhlásit.

Úkol	**průměrná doba (s)**
Udělit hlas první fotce v soutěži (včetně přihlášení)	9
Nahrát fotku, ale nehlasovat pro ni	11
Najít svoji fotku v soutěži a dát ji hlas	7
Otevřít si pravidla soutěže	2

Tabulka 4.1: Průměrná doba plnění úkolů uživatelských testů v soutěži

Testování administrace, jak již bylo zmíněno, probíhalo ve dvou částech. Uživatelé, kteří ji testovali, byly specifičtěji vybráni, a to podle toho, zda již měli v minulosti zkušenost s vedením firemní stránky na Facebooku. Doby plnění úkolů první části lze vidět v tabulce 4.9. Sledování uživatelů neodhalilo nějáký specifičtější problém. Druhý test, při kterém administrace obsahovala spoustu fotografií, odhalil málo výrazné tlačítko pro přejetí z náhledu fotografie na přehled všech fotografií. Uživatelé místo toho požívali tlačítko v menu, nebo tlačítko zpět v prohlížeči. Doby plnění úkolů lze vidět v tabulce 4.9. Z protokolu vyplynulo, že uživatelům scházela možnost ručního nahrávání fotografií do soutěže a změna jazykové mutace.

Úkol	**průměrná doba (s)**
Vytvoření nové soutěže	13
Nastavení aplikace	11
Publikování na Facebook stránku	7

Tabulka 4.2: Průměrná doba plnění úkolů uživatelských testů v administraci - první spuštění

Úkol	**průměrná doba (s)**
Skrytí nejhůře hodnocené fotografie	11
Zjištění kdo nahrál nejstarší fotografii, včetně skrytých	6
Smazání 2. nejlepší fotografie	4

Tabulka 4.3: Průměrná doba plnění úkolů uživatelských testů v administraci - po týdnu

Kapitola 5

Závěr

Cílem práce bylo vytvořit aplikaci, která by měla umožnit firmám vytvářet fotosoutěže a umisťovat je na svoji firemní Facebookovou stránku. Vytvoření fotosoutěže by mělo být rychlé, intuitivní a nezávislé na znalosti programování. Výsledná soutěž by měla působit moderním dojmem, být přehledná a srozumitelná pro soutěžící.

Nastudoval jsem moderní přístupy, které se používají při vytváření webových aplikací. Následně jsem provedl návrh, při kterém jsem na základě studií rozhodl přenést většinu logiky do klientského prohlížeče a nechat ji zpracovávat JavaScriptem, který se dotazuje serveru na konkrétní data. Na základě toho jsem musel rozdělit aplikaci do několika částí, které spolu komunikují přes rozhraní.

Implementace probíhala dle návrhu, přičemž největší problémy byly ty, které vznikly, kvůli přenesení logiky na klienta a to hlavně routování url a komunikaci se serverem. Routování jsem vyřešil použitím frameworků. Aplikaci se mi podařilo celou naimplementovat a otestovat na uživatelých.

Při vytváření aplikace jsem nabyl mnoha nových znalostí z oboru webových aplikací. Zjistil a vyzkoušel jsem si, že klientské prohlížeče mohou obstarávat většinu logiky aplikace a tím ušetřit práci serveru a zrychlit odezvu aplikace.

Literatura

[1] Průzkum: Malé firmy se stále učí používat Facebook. http://strategie.e15.cz/zpravy/pruzkum-male-firmy-se-stale-uci-pouzivat-facebook-1150052, 1. 11. 2008 [cit. 10. 2. 2015].

[2] Flux — Application Architecture for Building User Interfaces. https://facebook.github.io/flux/docs/overview.html, 2015.

[3] Material design. http://www.google.com/design/spec/material-design/, 2015.

[4] Bernard, B.: Úvod do architektury MVC. http://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/, 7. 5. 2009 [cit. 15.3.2015].

[5] Facebook: Facebook Developers. https://developers.facebook.com/.

[6] Malý, M.: Single Page Apps a řešení problémů s historií. http://www.zdrojak.cz/clanky/single-page-apps-a-reseni-problemu-s-historii/, 11. 6. 2011 [cit. 14.3.2015].

[7] Mrozek, J.: JavaScript na serveru: moduly a npm. http://www.zdrojak.cz/clanky/javascript-na-serveru-moduly-a-node-package-manager/, 12.10.2012 [cit. 20.3.2015].

[8] Resig, J.: *JavaScript a Ajax*, ročník 1. Brno: Computer Press, a.s., 2007, ISBN 978-80-251-1824-5.

[9] Vahl, A.: 8 Facebook Apps to Enhance Your Facebook Page. http://www.socialmediaexaminer.com/facebook-apps-for-custom-tabs/, 10. 7. 2013 [cit. 14. 1. 2015].

[10] Wikipedia: Architectural pattern — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Architectural_pattern&oldid=660567869, 2015, [Online; navštíveno 24. 3. 2015].

[11] Wikipedia: Facebook — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Facebook&oldid=658809927x, 2015, [Online; navštíveno 20. 4. 2015].

[12] Wikipedie: Skriptování na straně serveru — Wikipedie: Otevřená encyklopedie. http://cs.wikipedia.org/w/index.php?title=Skriptov%C3%A1n%C3%AD_na_stran%C4%9B_serveru&oldid=10123859, 2013, [Online; navštíveno 8. 2. 2015].

[13] Wikipedie: Webová aplikace — Wikipedie: Otevřená encyklopedie. http://cs.wikipedia.org/w/index.php?title=Webov%C3%A1_aplikace&oldid=12134315, 2015, [Online; navštíveno 12. 03. 2015].

Příloha A

Obsah CD

- /text/ - technická zpráva
- /src/server/ - zdrojový kód pro server
- /src/admin/ - zdrojový kód pro administraci
- /src/photo-content/ - zdrojový kód pro soutěž
- /src/install.txt - instrukce pro nastavení
- /src/create_db.sql - soubor pro vytvoření tabulek v databázi
- /poster/ - plakát
- /video/ - video
- /tests/ - testovací protokoly

Příloha B

Plakat