Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Napojení webu mojefavka.zcu.cz na sociální sítě

Plzeň 2017 Štěpán Ševčík

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 3. května 2017

Štěpán Ševčík

Poděkování

Tímto děkuji svému vedoucímu práce, Ing. Martinu Dostalovi, Ph.D., za vytrvalou podporu a cenné rady při vypracování této práce.

Abstract

The goal of this bachelor's thesis is to extend web application mojefavka.zcu.cz, which is used as informal presentation of studies of the Faculty of Applied Sciences. In theoretical part, thesis researches social media used by the faculty and it's departments and possible means of connection to MojeFavka application. In theoretical part, thesis also researches technologies related to web application development. In practical part, thesis describes created extensions of current application MojeFavka, which is based on AngularJS framework. Practical part also presents created administration part of MojeFavka, which is an PHP application used for management of emails, subscribers and other aspects of the application. In conclusion thesis presents testing frameworks Jasmine and PHPUnit, which are used for verification of correct functionality of the application.

Abstrakt

Cílem této bakalářské práce je rozšíření webu mojefavka.zcu.cz, který je provozován jako neformální představení studia na Fakultě Aplikovaných Věd. V teoretické části práce zkoumá sociální sítě využívané fakultou a jejími katedrami a jejich možné napojení do aplikace MojeFavka. V teoretické části se práce dále zabývá technologiemi potřebnými pro tvorbu webových aplikací. V praktické části práce popisuje vytvořená rozšíření stávající aplikace MojeFavka, která vychází z frameworku AngularJS. V praktické části je dále představena vytvořená administrační část, což je PHP aplikace, skrze kterou je realizována správa emailů, odběratelů a dalších aspektů aplikace. Na závěr práce představuje testovací frameworky Jasmine a PHPUnit, které jsou použity pro ověření správné funkčnosti této aplikace.

Obsah

1 Úvod

Fakulta aplikovaných věd provozuje webovou aplikaci MojeFavka, která má neformálním způsobem představit možnosti studia na této fakultě. Zároveň poskytuje službu Šancomat, která na základě informací od uchazeče vypočítá šanci na jeho přijetí.

Sociální sítě se v poslední době ukazují jako významná marketingová platforma [14]. Práce se proto zabývá hledáním způsobů, jak provázat aplikaci MojeFavka s různými sociálními sítěmi. Využití emailů pro udržování kontaktu s uživateli aplikace je jistou formou sociální sítě. Z tohoto důvodu práce řeší vytváření emailů a plánování jejich hromadného rozesílání uživatelům, kteří o ně mají zájem.

V teoretické části práce si představíme principy tvorby webových stránek včetně základních stavebních kamenů jako je HTML, CSS a protokolu přenosu dat HTTP/HTTPS. Dále si představíme skriptovací jazyk JavaScript, díky kterému lze vytvářet interaktivní aplikace spouštěné v prohlížeči, a JavaScriptové frameworky AngularJS a RequireJS.

Následně si představíme webové servery, které jsou nedílnou součástí komunikace pomocí protokolu HTTP, a zaměříme se na webový server Apache, který je využíván pro hostování webových aplikací na ZČU. Po webových serverech si stručně představíme služby realizované nad HTTP protokolem a tvorbu aplikací v PHP.

V této práci dále prozkoumáme dostupné prostředky pro napojení sociálních sítí a jiných služeb do aplikace MojeFavka a v kapitole Praktická Část si také představíme příklady konkrétních řešení pomocí dostupných pluginů.

V praktické části si představíme vlastní framework KiwiBlade, který byl rozšířen pro potřeby této práce. Z KiwiBlade vychází modul aplikačního programového rozhraní, díky kterému je možné doplnit dynamický obsah aplikace MojeFavka. Dále prozkoumáme aplikaci MojeFavka, porovnáme její architekturu před a po změnách této práce a představíme si metody provázání webové stránky s různými sociálními sítěmi. V neposlední řadě se také podíváme na administrační část Mojí Favky, skrz kterou lze spravovat hromadné odesílání emailů.

Na závěr se podíváme na metodiky testování jednotlivých částí aplikace MojeFavka. Přední část aplikace MojeFavka je testována pomocí frameworku Jasmine a administrační část a KiwiBlade jsou testovány pomocí frameworku PHPUnit.

2 Frontendové technologie

Webové aplikace nejsou vázané na běhové prostředí a pro jejich spuštění stačí webový prohlížeč, tedy aplikace, která dokáže interpretovat (zobrazit) HTML kód a s využitím CSS stylu zobrazit výslednou stránku. Přenos webových stránek od serveru ke klientovi je zajištěn pomocí HTTP protokolu, respektive pomocí šifrované varianty tohoto protokolu - HTTPS.

Vstupním bodem webové stránky je HTML dokument obsažený v odpovědi na HTTP požadavek. Tento dokument je po načtení prohlížečem interpretován neboli zobrazen. Pokud dokument vyžaduje pro zobrazení další prostředky, jako například styly, skripty, obrázky nebo jiné prostředky, prohlížeč pro ně standardně vytvoří navazující požadavky a po jejich načtení je také interpretuje.

2.1 HTML

HyperText Markup Language (HTML) je přední jazyk popisující obsah webových stránek. HTML používá tagy neboli značky k vytváření strukturovaných dokumentů pomocí textových prvků, kterými jsou například nadpisy nebo odstavce, seznamových prvků, odkazů nebo pomocí dalších elementů. HTML dále umožňuje autorům vkládat do stránky obrázky a jiné objekty a vytvářet interaktivní formuláře [19].

Základním prvkem HTML je element, kterým je popsán význam části stránky. Element je tvořen otevírací a uzavírací značkou, která v případě, že element nemá tělo, může být vynechána. Elementy mohou mít atributy, které upřesňují jejich vzhled, chování a případně i obsah. Obsahem elementu může být prostý text nebo další elementy.

```
<img src="rorschach/ink42.png" alt="Test␣image"/>
<label>
  Please, describe what you see on picture above
</label>
<textarea maxlength="1000" rows="4"></textarea>
```

Kód 2.1: Ukázka části formuláře

Na ukázce části formuláře znázorněné Kódem 2.1 se vyskytují značky:

- **img** je značka sloužící pro vložení obrázku do stránky, její atribut src určuje zdroj, odkud má být použit obrázek

- **label** je značka doprovázející uživatelský vstup, popisující požadovaný obsah

- **textarea** je značka uživatelského vstupu umožňující uživateli zadávat víceřádkový text. Její atribut maxlength určuje nejvyšší možný počet znaků atribut rows určuje výšku vstupu v řádcích textu

Webová stránka je běžný textový dokument strukturovaný podle specifikace jazyka HTML. Na začátku dokumentu je značka určující typ dokumentu, například `<!DOCTYPE html>` a dále následuje element *html*, který uvádí stromovou strukturu všech elementů stránky. V elementu *html* jsou zanořeny elementy *head*, neboli záhlaví, a *body*, neboli tělo či vlastní obsah stránky. Záhlaví určuje doprovodné informace o dokumentu jako je například jeho název nebo znaková sada použitá pro kódování obsahu. Tělo potom obsahuje elementy, které jsou zobrazovány prohlížečem.

V době vzniku této práce je nejnovější verze HTML5 široce podporována ve většině předních prohlížečů [1] HTML5 specifikuje množství technik, které se začaly objevovat v předchozích verzích, ale jejich implementace nebyly spolehlivé. Mezi tyto techniky patří například:

- **Canvas** Plátno, do kterého lze pomocí JavaScriptu vykreslovat text, tvary nebo obrazce

- **Web Sockets** Způsob udržování aktivního spojení mezi klientem a serverem, díky kterému lze přenášet data v reálném čase

- **Práce se souborem** Díky novým specifikacím pro nahrávání souborů lze například zobrazovat náhled nahrávaných souborů nebo nahrávat soubory jejich přetažením do určitého místa na stránce.

HTML5 dále definuje širokou množinu typů formulářových polí, mezi které patří například email, datum, URL nebo číslo. Díky těmto specifikacím může správnost zadaného vstupu ověřovat již prohlížeč [18].

2.2 CSS

Kaskádové styly (CSS z anglického Cascading Style Sheets) jsou pravidla, která popisují, jak má prohlížeč zobrazit dokumenty na obrazovce, při tisku nebo v různých dalších podobách. Kaskádové styly jsou významným nástrojem pro přizpůsobení vzhledu HTML dokumentů, bez kterého by webové

[1] http://html5readiness.com/

stránky byly méně robustní a poskytovaly znatelně méně příjemný vizuální zážitek [12].

Styly lze přímo definovat pro konkrétní element pomocí atributu style, ve kterém lze určovat pravidla. Dále lze styly definovat v souborech stylů pro množiny elementů podle jejich tagů nebo podle tříd či identifikátorů, které jsou elementům přiřazeny přímo v dokumentu.

Elementy se také mohou ocitat v různých stavech, které mohou dále upravovat vzhled. Příkladem tohoto může být například odkaz, který je zobrazen defaultně modrou barvou. Po navštívení odkaz nabude stav *visited* a začne být zobrazován barvou fialovou.

Pomocí stylů lze, mimo vzhledu, také definovat pozici elementů a animace doprovázející například jejich vznik nebo zánik.

Název Kaskádové Styly vychází z povahy aplikování stylů. V kaskádě mohou mít styly tři různé původy [21]:

- **Autor** Autor definuje styly dokumentu podle zavedených konvencí jazyku dokumentu. Například do HTML lze styly vkládat přímo do dokumentu nebo pomocí odkazů na externí soubory stylů.
- **Uživatel** Uživatel může definovat informace o stylu pro konkrétní dokument. Příkladem může být rozhraní poskytnuté uživatelským agentem, které generuje soubor uživatelských stylů.
- **Uźivatelský agent** Uživatelský agent, například prohlížeč, musí na dokument použít výchozí styly, pomocí kterých jsou elementy dokumentu prezentovány jednotně napříč uživatelskými agenty.

Kaskáda je potom tvořena pořadím, ve kterém jsou aplikovány styly na konkrétní elementy a které definuje prioritu stylu promítající se na výsledný vzhled elementu:

1. Deklarace uživatelským agentem
2. Uživatelovy standardní deklarace
3. Autorovy standardní deklarace
4. Autorovy důležité deklarace
5. Uživatelovy důležité deklarace

Pravidla stylů pro daný element jsou dále seřazena podle specifičnosti, kde styly více specifického selektoru mají přednost před obecnějšími selektory.

Výsledný styl, při shodě pořadí kaskády a při shodě specifičnosti selektorů, je zvolen na základě pořadí definování stylů, kde později definovaná pravidla mají přednost.

2.3 JavaScript

Webové prohlížeče zobrazují HTML dokumenty přizpůsobené pomocí kaskádových stylů. HTML definuje obsah, CSS doplňuje vzhled či prezentaci a JavaScript, při správném použití, přidává obsahu a prezentaci chování. Role JavaScriptu je vylepšení uživatelova požitku z prohlížení a usnadnění získání nebo přenosu informace. Javascript toho může docílit například následovně [4]:

- vytvářením vizuálních efektů, jako je například střídání obrázků, což uživatele jemně vede a pomáhá mu orientovat se ve stránce,
- řazením sloupců v tabulce pro usnadnění hledání toho, co potřebuje,
- skrýváním přebytečných detailů a jejich následným zobrazováním v průběhu uživatelova procházení,
- urychlením procházení pomocí načítání obsahu přímo od serveru, díky čemuž není třeba znovu načítat celou stránku.

JavaScriptový kód je v prohlížeči vykonáván v globálním kontextu, což znamená, že deklarované funkce a proměnné jsou viditelné ve všech následujících skriptech. Tomuto úkazu se říká "znečišťování globálního kontextu", což může mít nečekané následky[8].

Znečišťování globálního kontextu lze předcházet například použitím IIFE (Immediatelly Invoked Function Expression) výrazu, tedy okamžitě vykonaného funkčního výrazu, který je vykonán bezprostředně po jeho definici pro obalení funkčních modulů, viz Kód 2.2.

```
(function () {
var message = "Hello, World!";
console.log(message); //=> Hello, World!
})();

console.log(message); //=> undefined
```

Kód 2.2: Izolace kontextu pomocí okažitě vykonaného funkčního výrazu

JavaScript, pokud není zahrnut přímo v dokumentu, je třeba načítat pomocí dodatečných požadavků na server. Toto má negativní dopad na dobu potřebnou pro načtení dokumentu, obzvlášť pokud je načítáno větší množství souborů. Dobu načítání skriptů lze snížit optimalizací zdrojových kódů. Tento proces se nazývá minifikace a používá se po ustálení změn v kódu. Proces minifikace se může skládat z několika kroků, mezi které patří:

- Sloučení souborů zdrojového kódu do jednoho. Tímto se sníží počet požadavků, ale celkový objem přenesených dat zůstává.
- Vypuštění takzvaných bílých znaků. Za bílé znaky jsou například považovány mezery, tabulátory a znaky nového řádku. Tyto znaky nenesou v JavaScriptu žádný význam a neovlivňují výslednou funkčnost. Tento krok začíná snižovat objem přenesených dat.
- Nahrazením názvů proměnných. V JavaScriptu nezáleží na názvech proměnných, dokud mezi sebou nekolidují a proto je možné v rámci kontextu volně měnit názvy proměnných. Například místo názvu *productCategory* lze použít pouze jedno písmeno. Tento krok značně snižuje objem přenesených dat ale provádí nevratnou transformaci zdrojového kódu, která snižuje jeho čitelnost.

2.4 RequireJS

JavaScriptový modul je izolovaná část zdrojového kódu řešící vymezenou problematiku aplikace. Standardně jsou moduly načítány synchronně pomocí *script* elementů v dokumentu, což má za důsledek prodloužení načítání samotné stránky.

RequireJS řeší asynchronní definici modulů (AMD) a načítání těchto modulů nebo dalších JavaScriptových souborů. RequireJS je optimalizovaný pro použití v prohlížeči, ale lze jej používat i v dalších JavaScriptových prostředích [17].

Při použití ve webové stránce slouží RequireJS jako vstupní bod pro načítání dalších JS souborů. Po načtení RequireJS načte a spustí vstupní soubor aplikace, ve kterém jsou nakonfigurovány cesty k modulům nebo závislosti jednotlivých modulů, tedy výčet dalších modulů, které jsou nezbytně nutné pro správnou funkčnost daného modulu.

RequireJS uvádí pro práci s asynchronními moduly dvě funkce, které v zásadě poskytují stejnou funkčnost a liší se převážně významově:

- **require()** funkce slouží pro asynchronní volání funkcí, které nezbytně vyžadují konkrétní moduly

- **define()** funkce slouží pro deklaraci asynchronních modulů a předpokládají návratovou hodnotu

Použití těchto funkcí, představené v Kódu 2.3, znázorňuje deklaraci objektu Kolektiv, který slučuje množinu Osob, kde Osoba je objekt deklarovaný ve stejnojmenném modulu.

```
define(['osoba'], function (Osoba) {
  function Kolektiv() {
    var osoby = [];
    this.pridejOsobu = function(osoba) {
      if(!(osoba instanceof Osoba)) {
        throw new Error("Unexpected value");
      }
      osoby.push(osoba);
    }
  }
  return Kolektiv;
});
```

Kód 2.3: Použití RequireJS pro deklaraci modulu

Díky RequireJS lze docílit jasné specifikace stromu závislostí modulů, která je nezbytná pro dynamické načítání modulů bez potřeby zásahu do výchozího dokumentu.

2.5 AngularJS

AngularJS je JavaScriptový framework představený společností Google. AngularJS vznikl s myšlenkou zjednodušení starostí vývojáře frontendové aplikace. AngularJS výsledkem snažení o snížení náročnosti vývoje asynchronních JavaScriptových aplikací, takzvaných AJAX aplikací [7].

AngularJS je navržený tak, aby usnadňoval oddělení aplikační logiky od HTML dokumentu a aby tlačil vývojáře k modulárnímu návrhu. Vývoj pomocí Angularu předpokládá architekturu MVC.

Jednou z předností Angularu je jednoduchá rozšířitelnost pomocí dalších JavaScriptových knihoven. Každá část frameworku může být přizpůsobena nebo nahrazena, díky čemuž lze vývoj aplikace přizpůsobit jejím potřebám [2]. Příkladem nahrazení části frameworku je například výměna modulu *ngRouter* za *ui.router*, která bude blíže popsána v praktické části.

2.5.1 Správa závislostí

Závislosti je označení pro služby nebo jiné prostředky, které jsou nezbytně nutné pro vytvoření jiné služby nebo pro spuštění procedur. Závislost je základním prvkem kompozičního návrhu objektově orientované aplikace. S kompozičním návrhem je úzce spjatý "princip jedné odpovědnosti", který říká, že by každý prvek aplikace měl být zodpovědný pouze za jednu věc, nebo jinými slovy každý prvek by měl mít pouze jeden důvod pro změnu [5]. Příkladem odpovědnosti může být formátování dat do textové podoby nebo zápis textu do souboru.

Služba může typicky získat závislost třemi způsoby:

1. Služba závislost vytvoří, většinou pomocí konstruktoru.
2. Služba použije pro dohledání závislosti globální proměnnou.
3. Službě je závislost předána, když ji potřebuje.

První dva způsoby nejsou vhodné, protože je závislost úzce spjata se službou a nelze ji jednoduše nahradit. Toto je problematické především pro testování služby. Třetí způsob vychází z principu Inversion of Control a je vhodný protože služba již nemá odpovědnost za získávání závislostí a místo toho se spoléhá že jí budou odpovídající závislosti předány[1]

Značnou předností Angularu je mimo dalších i správa závislostí (Dependency Management) obsažená v jádře. Pro její správnou funkčnost je třeba anotovat funkce, čehož je možné docílit až třemi různými způsoby. Pomocí anotací funkcí může správce závislostí ($injector) zjistit, které závislosti jsou potřeba pro spuštění dané funkce, a tyto závislosti automaticky při jejím spuštění vložit (injektovat). Možnosti anotování parametrů funkce jsou následující:

- Implicitně
- Explicitní anotace polem v místě použití
- Explicitně atributem $inject

Implicitní anotace je realizována přímo pomocí názvů parametrů funkce a je nejjednodušší na zápis. Nevýhodou implicitní anotace je, že výsledný kód nelze minifikovat. Jednoduchý konfigurační blok využívající implicitní anotací parametrů je znázorněn Kódem 2.4.

```
var configureBlock = function ($stateProvider) {
  $stateProvider.state('home', {
    url: '/',
    templateUrl: 'app/templates/home.html'
  });
}
```

Kód 2.4: Implicitní anotace

Explicitní anotace parametrů v místě použití je realizována obalením funkce polem, na jehož začátku jsou uvedeny požadované závislosti a na konci pole je samotná funkce s parametry pro závislosti v žádaném pořadí. Explicitní anotace polem v místě použití je znázorněna Kódem 2.5. Výhoda tohoto kódu je, že zůstává funkční i po minifikaci, protože řetězce nejsou minifikací ovlivněny, ovšem jedná se o lehce složitější zápis.

```
var configureBlock = ['$stateProvider', function (
   $stateProvider) {
  $stateProvider.state('home', {
    url: '/',
    templateUrl: 'app/templates/home.html'
  });
}];
```

Kód 2.5: Explicitní anotace v místě použití

U explicitní anotace pomocí atributu $inject, obdobně jako in-place anotace, není ovlivněna funkčnost minifikací. Pole obsahující názvy závislostí v tomto případě neobaluje funkci, ale je jí vloženo jako atribut $inject, podle kterého správce závislostí pozná, které služby má dohledat a dosadit. Ukázka použití tohoto druhu anotace je znázorněna Kódem 2.6. Tento způsob anotace nelze použít pro přímo použité anonymní funkce, protože přiřazení atributu nelze vykonat ve stejném kroku, ve kterém je funkce definována.

```
configureBlock.$inject = ['$stateProvider'];
function configureBlock($stateProvider) {
  $stateProvider.state('home', {
    url: '/',
    templateUrl: 'app/templates/home.html'
  });
}];
```

Kód 2.6: Explicitní anotace atributem

Všechny tři výše znázorněné útržky kódu mají stejný význam a pokud se vyskytnou uvnitř IIFE bloku znázorněného Kódem 2.7, výsledná funkčnost bude stejná.

```
(function (angular){
  /* definice config bloku */
  angular.module('myApp', [])
    .config(configureBlock);
})(angular);
```

Kód 2.7: Použití anotovaných funkcí

2.5.2 Moduly v AngularJS

Základní celek v Angularu je modul - část aplikace řešící určitou problematiku. Moduly deklarují komponenty, kterými mohou být:

- kontroléry (controller),
- služby (service) či prostředky (value),
- továrny (factory) na služby či prostředky,
- direktivy (directive),
- filtry (filter).

Kontrolér slouží jako prostředník mezi šablonou a zbytkem aplikace. Do kontroléru patří logika starající se o aktualizaci dat v šabloně a zpracování dat a událostí. Služby a prostředky slouží jako jednotné umístění výkonného kódu nebo konfigurací, které lze získávat skrze správce závislostí. Deklarace služby nebo prostředku pomocí továrny přináší výhodu, že je objekt je vytvořen až v momentě, kdy je použit. Továrny lze také vytvářet pomocí poskytovatelů (provider), kteří slouží pro globální konfiguraci služeb či hodnot, které poskytují.

Direktivy slouží pro vytváření komponent a rozšíření funkčnosti a lze pomocí nic definovat zcela nové HTML elementy a atributy. Filtry slouží pro přizpůsobení hodnot před výpisem nebo pro výběr prvků z množiny.

Moduly mohou také obsahovat neomezené množství bloků typu config() a neomezené množství bloků typu run(). Bloky config() jsou určeny pro přípravu služeb pomocí jejich poskytovatelů před začátkem využívání dané služby. Využití konfiguračních bloků je znázorněno v předchozí kapitole Kódem 2.7.

Do bloků typu run() lze naopak vkládat pouze konkrétní služby. Tyto bloky slouží pro přizpůsobení vytvořené služby nebo pro vytvoření posluchačů událostí (tzv. watcher). Na ukázku konfiguračních bloků navazuje ukázka běhového bloku znázorněná Kódem 2.8, která zajišťuje, že se při pokusu o přechod do neexistujícího stavu aplikace vrátí do výchozího stavu.

```
run.$inject = ['$rootScope', '$state'];
function run($rootScope, $state) {
  $rootScope.$on('$stateNotFound', function (event,
     unfoundState) {
    console.error("State␣", unfoundState, "␣was␣not␣
       found");
    $state.go('home');
  });
}
```

Kód 2.8: Ukázka run() bloku

Aplikace vycházející z frameworku AngularJS nemají žádný vstupní bod - jedná se totiž o moduly, které definují funkčnost a které lze rozšířit pomocí dalších modulů. Po načtení stránky je v šabloně nebo ve vstupním skriptu provázán HTML dokument nebo jeho část s hlavním modulem a od té doby se AngularJS začne starat o veškerou běhovou logiku.

3 Protokol HTTP

Protokol HTTP slouží pro získávání například HTML dokumentů, CSS stylů, JS skriptů, obrázků či jiných souborů. Všeobecně se přenášenému obsahu říká resource, nebo česky prostředek, a HTTP se nestará o typ přenášeného prostředku.

Naopak velmi důležitá součást přenosu je identifikace prostředku, která je řešena pomocí URL (Uniform Resource Locator) neboli jednotného lokátoru prostředku. Ukázka URL struktury v úplné podobě je znázorněna Kódem 3.1(A). V běžném případě je při dotazování využito v URL jen několik částí viz Kód 3.1(B).

```
(A) schéma://uživatel:heslo@doménové_jméno:port/
   cesta_k_prostředku?params#návěští
(B) http://www.haystack.com/actions/search?term=
   needle&page=2
```

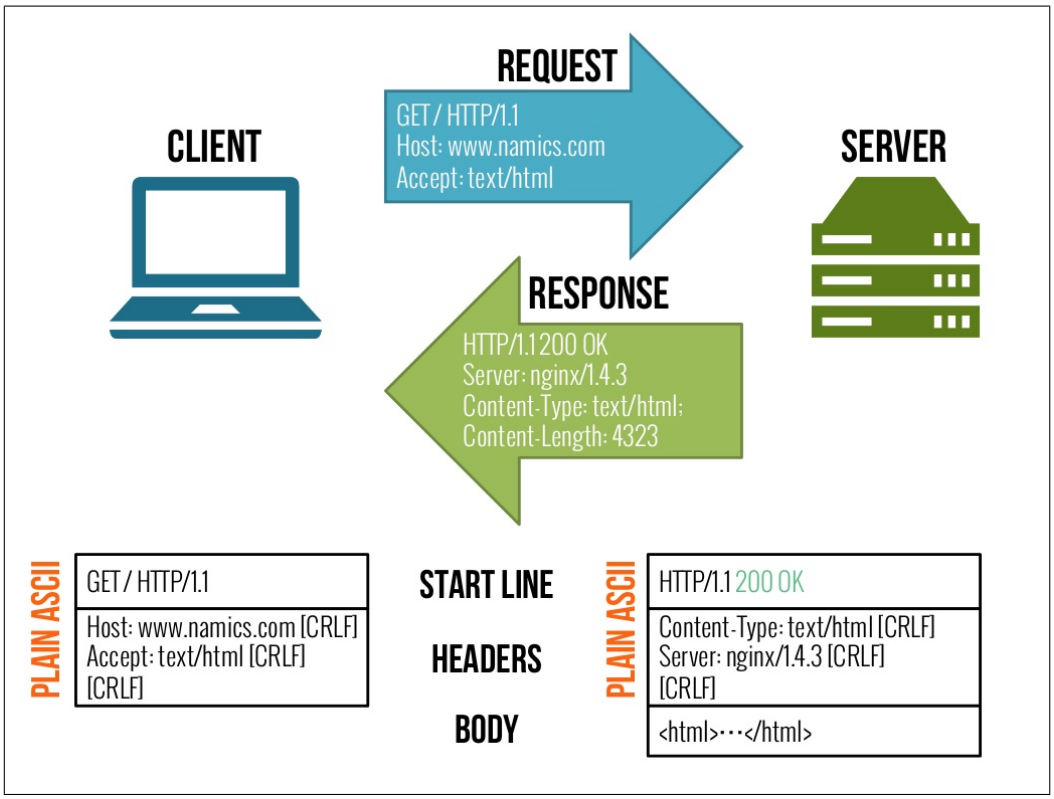
Kód 3.1: Struktura URL adresy

Jak je zřejmé z Kódu code.url-structure, URL obsahuje tyto části:

- **schéma** určuje protokol přenosu, například http nebo https,
- **uživatel + heslo**, slouží pro prokázání identity uživatele,
- **doménové jméno** je název počítače v internetu. Například www.google.com, může být nahrazeno adresou IP,
- **port** určuje aplikaci na kterou se má klient připojit,
- **cesta k prostředku** je cesta od kořenu webového serveru,
- **parametry** upřesňují požadavek, parametry jsou odděleny ampersandem,
- **návěští** například určuje umístění v dokumentu.

Protokol HTTP je navržený pro komunikaci mezi dvěma stranami, mezi klientem a serverem, ve formátu Request - Response, tedy Požadavek - Odpověď.

Server, nebo také webový server, je aplikace naslouchající na stanoveném portu. Standardně se pro HTTP protokol používá port 80 a pro HTTPS port 443. Klient je libovolná aplikace vznášející požadavek na server. Typickým



Obrázek 3.1: Struktura HTTP komunikace[15]

klientem je webový prohlížeč ale často se jako klient používají i různé konzolové aplikace. Komunikace mezi serverem a klientem je znázorněna na Obrázku 3.1.

HTTP požadavek začíná na prvním řádku záhlavím, které obsahuje metodou požadavku, požadovaný prostředek na serveru a verzi HTTP. Metoda požadavku definuje důvod požadavku a URL definuje rámec nebo oblast[22]. Běžně používané metody jsou:

- GET - slouží pro získání obsahu na dané URL
- HEAD - má stejný význam jako GET, odpověď by neměla obsahovat tělo
- POST - používá se pro odeslání dat na server
- PUT - vytváří prostředek na dané URL z obsahu těla požadavku
- DELETE - maže prostředek na dané URL
- OPTIONS - zjišťuje metody, které může klient vykonat na dané URL

Dále následují HTTP hlavičky, ve kterých jsou přenášeny různé informace, které specifikují očekávaný typ požadovaného prostředku, možnosti spojení, aktuální čas nebo uživatelský prohlížeč. V HTTP hlavičkách odpovědi jsou potom hlavičky popisující vlastnosti obsahu, jako například typ obsahu, velikost, použité kódování anebo nové umístění v případě jeho přesunutí z požadované lokace. Za hlavičkami následuje prázdný řádek značící konec HTTP záhlaví a dále může následovat samotný obsah[22].

Typ obsahu v HTTP je udáván ve formátu MIME, což je rozšiřitelný mechanismus pro jednoznačný popis obsahu. Díky MIME typu může prohlížeč zvolit odpovídající mechanismus prezentace obsahu [6].

Server při přijetí požadavku na základě metody a URL vykoná určitou akci a zpět odešle odpověď, která obsahuje odpovídající obsah a stavový kód.

Stavové kódy HTTP odpovědi slouží pro okamžité rozlišení stavu odpovědi a spadají do jednoho z následujících rozsahů[22]:

1xx **Informační zprávy**, Tento rozsah nebyl definován až do příchodu HTTP/1.1 a ve většině webových serverů je zcela nevyužit. Zprávy tohoto rozsahu slouží pro informace klientovi od serveru, například o změně protokolu.

2xx **Úspěšný klientský požadavek**, Zprávy tohoto rozsahu označují úspěšné vykonání klientova požadavku.

3xx **Přesměrování**, Zprávy tohoto stavového kódu jsou posílány, pokud byl například požadovaný dokument dočasně nebo trvale přesunut.

4xx **Chyba klientského požadavku**, Tyto zprávy oznamují chybu klientského požadavku, například požadavek na neexistující prostředek 404, nedostatečná oprávnění pro práci s prostředkem 401 nebo všeobecná chyba 400.

5xx **Chyba serveru**, Odpovědi obsahující stavový kód tohoto rozsahu značí chybu serveru a běžnými kódy jsou například 500 Vnitřní chyba serveru nebo 503 Služba není k dispozici.

Návrh protokolu HTTP pro jednoduchost nebere v potaz stav, jedná se tedy o bezestavový protokol, a tím pádem všechny HTTP zprávy jsou přenášeny a zpracovány nezávisle na ostatních. Díky tomuto lze předpokládat, že všechny požadavky na konkrétní prostředek budou v bezchybném stavu obslouženy odpovědí obsahující stejný prostředek.

Kvůli bezestavosti protokolu ale webový server nemá jednoznačný způsob, jak identifikovat protistranu. Pro "zavedení stavu" do aplikace jsou používány soubory Cookies, které mohou obsahovat informace právě o stavu

aplikace. V souboru Cookies se často vyskytuje informace session id, což je identifikátor relace. Pro každou otevřenou relaci si server uchovává unikátní data, ve kterých může být například identifikace přihlášeného uživatele, seznam posledních navštívených stránek nebo také aktuální obsah nákupního košíku.

V prvním kroku komunikace vyšle klient HTTP požadavek, který může obsahovat upřesňující parametry. Hlavním prvkem požadavku je URL žádaného prostředku. V tomto požadavku jsou hlavičky, které určují například přístup ke cachování[1], přístup k přesměrování nebo samotný cíl požadavku, soubor cookies a případně i tělo zprávy.

HTTP odpověď opět obsahuje hlavičky, mezi kterými je například stavový kód odpovědi nebo MIME typ obsahu těla odpovědi. Za hlavičkami následuje prázdný řádek, který značí konec HTTP záhlaví, a tělo odpovědi, kde je obsah odpovídající původnímu požadavku. Často tělo takovéhoto požadavku obsahuje HTML dokument.

Mimo webových aplikací jsou na protokolu HTTP provozovány také webové služby, což jsou oddělené části aplikací. Pomocí webových služeb lze rozdělit aplikaci do nesouvislých částí, neboli vrstev. Díky tomuto rozdělení je možné vytvořit i několik uživatelských rozhraní jedné aplikace, která využívají jednotné Aplikační Programové Rozhraní (API). Typ API je určen podle způsobu práce s prostředky. Příklady typů API jsou:

- SOAP (Simple Object Access Protocol)
- REST (Representational State Transfer)
- Graph[2]

Všeobecné API, u kterého nelze jednoduše určit typ, lze také nazývat Web API.

[1] Cachování je způsob ukládání odpovědí, které lze vracet aniž by musel server generovat novou odpověď.

[2] http://graphql.org

4 Webový server

Webový server je libovolná aplikace schopná přijímat HTTP požadavky a obsluhovat je odpověďmi. Webové servery poskytují prostředky klientům pomocí protokolu HTTP. Požadavek je obsloužen webovým serverem. Jako webový server se běžně používají servery Apache, IIS a nginx. Tato práce se zaměřuje na webový server Apache, pomocí kterého je provozována aplikace MojeFavka.

4.1 Apache

Webový server nabízí mnoho konfiguračních možností. Jedním z nejdůležitějších nastavení je kořenový adresář dokumentů, tedy složka v souborovém systému, která odpovídá URL adrese serveru a od které jsou hledány soubory dotazované pomocí URL.

Server Apache dále umožňuje podrobnější konfiguraci aplikace pomocí souborů *.htaccess* umístěných v adresáři obsahujícím aplikaci. Jedním z primárních účelů souboru *.htaccess* je nastavení přepisování URL pomocí regulárních výrazů a dalších podmínek. Díky tomu lze přizpůsobit například adresářovou strukturu aplikace, pokud webový server nepovoluje úpravy kořenového adresáře dokumentů. Kód 4.1 znázorňuje, jak lze přizpůsobit kořenový adresář do složky www.

```
RewriteEngine On
RewriteBase /
RewriteCond %{REQUEST_URI} !^/www
RewriteRule ^(.*)$ www/$1 [QSA]
```

Kód 4.1: Přizpůsobení kořenového adresáře

Na prvních dvou řádcích soubor *.htaccess* vyžaduje modul přepisování URL a nastavuje část URL, která nemá být ovlivněna přepisováním. Na třetím řádku je zapsána podmínka podmiňující použití následujícího přepisovacího pravidla. Tato podmínka vyžaduje, aby URL požadavku nezačínalo na `/www`. Pokud je tato podmínka splněna, pravidlo odchytí všechny požadavky a vloží před ně `/www`. Pravidlo má také nastaven příznak *QSA* (query string append), čímž je zaručeno, že všechny parametry požadavku typu GET budou přeneseny do přepsaného URL[3].

V takovéto konfiguraci bude požadavek http://localhost/index.php na

server s kořenovým adresářem dokumentů nasteveným na `/var/htdocs` vyřízen skriptem nacházejícím se v souboru `/var/htdocs/www/index.php`.

Dalším častým využitím je přesměrování požadavků. Přesměrování se používá, pokud do aplikace existují vnější odkazy a aplikace je například přenesena z HTTP protokolu na jeho zabezpečenou variantu HTTPS. V takovém případě je možné, že staré odkazy přestanou fungovat. Na Kódu 4.2 je znázorněno, jak využít přesměrování na úrovni Apache serveru.

```
RewriteEngine On
RewriteCond %{HTTPS} !=on
RewriteRule ^/?(.*) https://%{SERVER_NAME}/$1 [R,L]
```

Kód 4.2: Ukázka přesměrování se zachováním požadovaného zdroje

V ukázce znázorněné Kódem 4.2 je opět žádost na modul přepisování požadavku. Následuje podmínka vybírající požadavek, který nevyužívá protokol HTTPS. Pokud je tato podmínka splněna, následující přepisovací pravidlo odchytí všechny požadavky a přesměruje je na aktuálně dotazovaný server pomocí HTTPS protokolu, přičemž zanechá cestu k požadovanému souboru a parametry požadavku. Toto pravidlo má následující příznaky:

- Příznak R (redirect) značí, že má být požadavek přesměrován.
- Příznak L (last) značí, že nemají být použita žádná další pravidla aktuálního souboru .htaccess.

Takovýto server bez dodatečné konfigurace poskytuje statické stránky a další prostředky. Pomocí CGI (Common Gateway Interface) neboli pomocí sjednoceného rozhraní mohou webové servery spouštět programy v prostředí serveru. Toto rozhraní definuje způsob a předávání dat mezi serverem a dalším programem. Díky tomuto rozhraní má webový server například při zpracování požadavku na prostředek index.php možnost spustit PHP interpreta, který vykoná tento skript a předá webovému serveru výsledek, který je potom serverem vložen do odpovědi jako jakýkoliv jiný prostředek. Častěji se ale možnost interpretace PHP skriptů webového serveru dodává pomocí Apache modulu kvůli optimalizaci systémových volání. Výhoda PHP skriptů oproti prostým HTML dokumentům je, že v nich lze kdekoliv otevírat takzvané PHP tagy, které jsou interpretovány, a do HTTP odpovědi se jejich obsah nedostane, pokud v nich nejsou použity funkce pro výpis. V takovém případě, tedy když má dokument proměnlivý obsah, se jedná o HTML šablonu. Alternativně lze použít celý PHP skript pro aplikační logiku skládající se z definic funkcí nebo objektových tříd.

Vývoj a provoz PHP aplikací je vysoce rozšířený a díky tomu vznikají hotová webserverová řešení jako je například LAMP (Linux + Apache + MySQL + PHP) respektive multiplatformní alternativa XAMPP (Cross-Platform + Apache + MariaDB + PHP + Perl)[ibm-lamp].

4.2 Architektura webové aplikace

Webové aplikace lze ve většině případů rozdělit na dvě části: backend a frontend. S Frontendem jsme se již seznámili v rámci kapitoly Frontendové technologie. Pro Frontend platí že:

- obsahuje aplikační výstupy,
- tvoří uživatelské rozhraní, pomocí kterého uživatel aplikaci ovládá,
- základní strukturaf rontendu a jeho samotný obsah jsou definovány pomocí HTML dokumentu, vzhled tohoto dokumentu je určen kaskádovými styly a interaktivitu přinášejí skripty JavaScript.

Backend se stará o záležitosti "na pozadí", tedy například o zpracování dat z formulářů a jejich následné ukládání. K této části uživatel aplikace nemá přístup a je vhodné zde definovat tajné parametry, jako jsou například přístupové údaje k databázím nebo autorizační klíče k dalším službám.

O poměru částí Backend a Frontend značně rozhoduje použitá technologie a dále použitý návrh samotné aplikace. Ve většině případů jsou webové aplikace kombinací serverové a klientské části, kde serverová část obstarává například ukládání dat nebo neveřejné výpočty a klientská část mění povahu užívání aplikace.

Aplikace založené na technologii PHP, ASP.NET nebo Java Servlet jsou vykonávány na straně serveru, tedy na backendu aplikace, kde je zpracováván požadavek. Pro tento požadavek je následovně generována odpověď s tělem obsahujícím obsah tvořící frontend aplikace, respektive data pro frontend. Takovým aplikacím se říká server-side, z čehož ve většině případů vyplývá, že každá změna pohledu nebo akce znamená nové načtení stránky. Nové načtení stránky s sebou přináší aktuálnost informací, což je výhodné pro interaktivní aplikace.

Alternativně mohou být aplikace vykonávány převážně na straně klienta - v prohlížeči. Takovéto aplikace se nazývají client-side a běžně jsou realizovány pomocí skriptovacího jazyka JavaScript. Příkladem mohou být aplikace založené na frameworcích, jako je například AngularJS nebo React.

Aplikace vykonávané na straně klienta jsou vhodné pro jednostrannou prezentaci, kdy není vyžadováno zpracování vstupu návštěvníka - zpracování uživatelských vstupů není v rámci možností aplikace vykonávané na straně klienta. Předností takovéto aplikace je, že všechny části aplikace stačí načíst pouze jednou a navigace mezi nimi probíhá jen na straně klienta. Naopak v případě nutné interaktivity aplikace je potřeba navrhnout způsob pro aktualizaci zobrazovaných informací.

Dále v případě, že je potřeba zpracovat vstupy client-side aplikace, je nutné definovat rozhraní komunikace a implementovat je na straně serveru pomocí takzvaných end-pointů. Příklad end-pointu všeobecného web API je popsán Kódem 4.3.

```
Method: GET
URL: /numbers/pi
Parameters:
  [precision=4] :number,
Returns: float
```

Kód 4.3: Příklad endpointu

Odpověd na požadavek GET http://localhost/numbers/pi?precision=3, který je vyřízen tímto end-pointem, ve formátu JSON je znázorněna Kódem 4.4.

```
{"status":"success", "data": {"value": 3.141}}
```

Kód 4.4: Příklad zprávy ve formátu JSON

4.2.1 Backend

Tato část aplikace se nachází na webovém serveru a její práci lze popsat jednoduchým vzorem Request-Response, tj požadavek-odpověď, což vyplývá z formy komunikace pomocí protokolu HTTP. Ukázka aplikace *hello.php*, viz Kód 4.5, vyřizuje požadavek pozdrav, který přijímá parametr `name`, tedy jméno, které má pozdravit. Pokud žádné jméno není k dispozici, ukázka pozdraví svět.

```
<?php
$name = filter_input(INPUT_GET, 'name');
if(!$name) {
  $name = "World";
}
echo "Hello, $name";
```

Kód 4.5: Hello world v PHP

Tento kód vyřizuje požadavek zachycený webovým serverem v následujících krocích:

1. Server přijme požadavek, který pomocí URL specifikuje, že požaduje dokument hello.php. Příklad takovéhoto URL může vypadat následovně: http://localhost/hello.php

2. Server zjistí, že od kořenového adresáře lze nalézt složku examples a v ní soubor hello.php. Server připraví proměnné prostředí, serveru a požadavku a spustí interpretaci skriptu v tomto souboru. Pokud by URL vypadala následovně: http://localhost/hello.php?name=George, proměnná požadavku GET bude obsahovat pod klíčem 'name' hodnotu 'George'.

3. Výstupem skriptu je textová odpověď typu text/html začínající hlavičkami, za kterými následuje dokument. Dokud není PHP skriptem zobrazen žádný výstup, je možné posílat hlavičky pomocí funkce header(). Před prvním zápisem do těla odpovědi server doplní důležité hlavičky do záhlaví zprávy a po dokončení vykonávání skriptu server odešle odpověď.

Alternativně lze do těla odpovědi vložit JSON či XML dokument, které mají popořadě typy *application/json* a *application/xml*. Takovéto odpovědi jsou vhodné pro přenos dat protože je lze programově zpracovávat velmi jednoduše.

Rozdělení aplikační logiky do modulů přináší při návrhu aplikací řadu výhod. Správně izolované funkční celky jsou přehledné a snadno upravitelné bez rizika ovlivnění ostatních celků[10].

Typicky se při návrhu webových aplikací používá návrhový vzor MVC, tedy Model - View - Controller, ten definuje tři úrovně aplikace, které spolu komunikují:

- **Model** obsahuje aplikační logiku a stará se ukládání a načítání dat. Běžně se skládá například z komunikace s databází.
- **View** (pohled) se stará o prezentaci dat a zdrojů uživateli. Většinou je za tímto účelem využit šablonovací systém.
- **Controller** vyřizuje požadavky uživatele, kterými uživatel žádá o zobrazení nebo upravení prostředků.

Hlavní předností tohoto návrhového vzoru je oddělení tří na sobě relativně nezávislých částí a s tím spojené zjednodušení vývoje, údržby a testování aplikace [20, tip 337, p. 156]. Další z výhod tohoto rozdělení je, že je možné kteroukoliv z vrstev nezávisle nahradit, což může být zapotřebí, pokud se v některé z použitých technologií zjistí závažná chyba.

5 Sociální sítě

Sociální sítě pomáhají lidem zůstat v kontaktu s lidmi nebo s organizacemi, o které se zajímají. Sociální sítě nabízejí jednotné řešení komunikace mezi poskytovateli obsahu a uživateli.

V této práci budou zmíněny sociální sítě využívané Fakultou Aplikovaných Věd a jejími katedrami. V této práci budou fakulta a její katedry označovány jako sociální subjekty, protože se v kontextu práce jedná o celky, které se prezentují pomocí sociálních sítí. Při vzniku této práce byly nalezeny u sociálních subjektů následující sítě:

- Fakulta aplikovaných věd: Facebook
- Katedra informatiky a výpočetní techniky: Facebook
- Katedra kybernetiky: Facebook, YouTube
- Katedra mechaniky: -
- Katedra fyziky: Facebook, YouTube
- Katedra matematiky: Facebook
- Katedra geomatiky: Facebook, Google+, YouTube

Většina sociálních sítí staví na určité myšlence, využívá množiny médií a používá model interakce.

Například YouTube staví na sdílení libovolného obsahu ve video formátu, od výukových videí, přes hudební klipy a upoutávky k filmům či receptáře až po nahrávky z dovolené nebo reklamní kampaně. Model interakce YouTube lze popsat formátem "Vystavující - Sledující", kde každý uživatel může být vystavujícím i sledujícím.

Facebook staví na myšlence sdílení a jako média využívá různé textové formáty jako jsou například statusy či příběhy, obrázkové nebo video materiály, respektive alba z nich tvořená. Klíčovým prvkem Facebooku jsou potom reakce na obsah, díky kterým může uživatel vyjádřit, jak na něj obsah působí, tzv. likováním.

Dalšími médii Facebooku jsou zájmy a reakce na obsah, u kterého je nastaveno propojení s Facebookem, anebo třeba herní aktivita. Jeden z předních modelů interakce na Facebooku vychází z formátu "Známost" či "Spojení", kde je sdílený obsah zobrazen všem známostem.

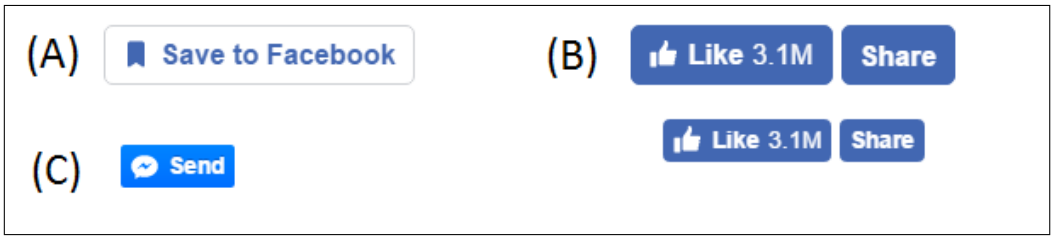
Google+ obdobně vychází z myšlenky sdílení a jako přední média používá textové příspěvky, obrázky a videa a případně alba tvořená z předchozích médií. V rámci sítě Google+ lze na obsah reagovat pomocí '+1', čímž uživatel vyjadřuje souhlas nebo pozitivní reakci. Klíčovým modelem interakce jsou Kruhy, pomocí kterých lze určovat cílové audience pro sdílení obsahu. Standardními kruhy jsou například "Známí", "Přátelé" nebo "Pouze Sledování". Díky kruhům lze zařadit uživatelovy kontakty až do několika skupin a při sdílení obsahu lze vybírat, které kruhy nebo osoby mají či nemají vidět sdílený obsah.

Spousta sociálních sítí usiluje o pokrytí širší oblasti působení mimo domovské stránky dané sociální sítě. Tohoto efektivně dosahují poskytováním prostředků vývojářům, kteří mohou jednoduše vytvořit propojení s vlastní aplikací. Takovýmito nástroji jsou například hotové pluginy, o jejichž vložení na stránku se stará knihovna poskytnutá danou sítí, nebo celá API, díky kterým může vývojář hlouběji integrovat služby dané sociální sítě.

5.1 Facebook

Facebook poskytuje několik hotových řešení připravených, pluginů, pro použití ve stránce. Tato řešení umožňují vkládání stylizovaných ovládacích prvků do stránky pomocí definované syntaxe. Použití jednotlivých pluginů je podmíněné načtením Facebookové vývojářské sady (FbSDK).

Po načtení vývojářské sady Většina těchto pluginů funguje na principu vložení podstránky do dokumentu, především pomocí HTML elementu <iframe>. Funkčnost většiny pluginů je podmíněna inicializováním vývojářské sady a provázáním aplikačního klíče. Pokud je vývojářská sada inicializována bez tohoto klíče, je vysoce pravděpodobné, že pluginy nebudou fungovat správně.



Obrázek 5.1: Ukázky facebookových pluginů

Mezi tyto pluginy patří například tlačítka pro uložení, Like a sdílení nebo přeposlání, které jsou znázorněny popořadě na Obrázku 5.1 (A), (B) a (C).

Klíčový parametr těchto pluginů je URL stránky, ke které se vztahují.

Tlačítko Uložit, znázorněné Obrázkem 5.1(A), vytvoří z nastaveného odkazu záložku v soukromé schránce uživatele[1] a automaticky jí umístí do rozpoznané kategorie.

Po uložení záložky Facebook uživatele později několikrát upozorní na nově přidanou záložku a dále jej upozorňuje na případné aktualizace týkající se uložené záložky. Například pokud má uživatel uloženou záložku na zájezd, na který následně vznikne akční nabídka, uživatel je na tuto nabídku upozorněn.

Dalšími pluginy jsou tlačítko Sdílet a tlačítko Like, které lze zkombinovat s tlačítkem Sdílet do jednoho pluginu. Kombinovaná varianta tlačítka Like a Sdílet je znázorněna na obrázku 5.1(B). Tato tlačítka slouží pro rychlejší Likování obsahu na internetu, respektive jeho sdílení na zeď dále upřesněné stránky.

Tlačítko Poslat, znázorněno obrázkem 5.1(C), slouží pro sdílení daného obsahu konkrétnímu kontaktu nebo skupině, do které má aktuálně přihlášený uživatel přístup.

Tyto pluginy slouží pro přenos aplikace, anebo její části, na Facebook a ve většině případů je na straně Facebooku pro tento obsah vytvářen náhled ve formě útržku textu a obrázku. Podobu náhledu lze přizpůsobit dle notace open graph[2], díky které lze stránku anotovat unikátními elementy, které poskytují informace používané nástrojem pro vytváření náhledu.

Pro konfiguraci pluginu Komentáře je, podobně jako u předchozích pluginů, potřeba URL adresa, ke které se Komentáře vztahují. Vložený plugin představuje vlákno komentářů, které se vztahuje k dané URL stejně, jako by se vztahovalo k příběhu na Facebooku. Tento plugin nabízí širokou škálu možností pro moderování, mezi které patří například schvalování komentářů, procházení nahlášených komentářů nebo přiřazování těchto komentářů jednotlivým moderátorům.

Poslední plugin Facebooku zmíněný v této práci je plugin Stránka, pomocí kterého je možné do webové stránky vložit náhled na facebookovou stránku a její příspěvky[3]. Ukázka tohoto pluginu bude znázorněna v kapitole Provázání se sociálními sítěmi Obrázkem 7.3.

Plugin Stránka má mnoho konfiguračních možností, mezi které patří například velikost úvodního obrázku, zobrazované záložky, celková velikost pluginu, fotky lidí, kterým se stránka líbí a další. Tyto možnosti jsou specifikovány pomocí atributů elementu, do kterého má být plugin vložen.

[1] https://www.facebook.com/saved

[2] https://developers.facebook.com/docs/sharing/webmasters

[3] https://developers.facebook.com/docs/plugins/page-plugin

Facebook dále poskytuje Graph API, díky kterému lze volat množství metod pro práci uzly (Node), tedy s entitamy na Facebooku. Uzlem může být například uživatel, odkaz, vlákno zpráv služby Messenger, status, notifikace, nebo další. Práce s Graph API probíhá pomocí požadavků, které určují pracovní uzel, a konkrétní vlastnosti tohoto uzlu, se kterými bude požadavek pracovat.

5.2 Google+

Google+ poskytuje vývojářskou platformu pro JavaScript, skrz kterou lze ovládat zobrazování dostupných pluginů[4]. Při načtení platformy lze zvolit jazyk, ve kterém mají být pluginy zobrazovány, a způsob zobrazování - zdali se mají pluginy zobrazit po načtení platformy anebo mají být zobrazovány při explicitním volání metod platformy.

Některé dostupné pluginy sítě Google+ jsou:

- +1 Tlačítko, pomocí kterého lze doporučit službu spjatou s tlačítkem,
- Odznáček, který poskytuje okamžitý náhled na stránku nebo profil uživatele,
- Tlačítko Sdílení, díky kterému lze snadno sdílet obsah se zvolenými kruhy.

Pro každý plugin je v platformě sada metod:

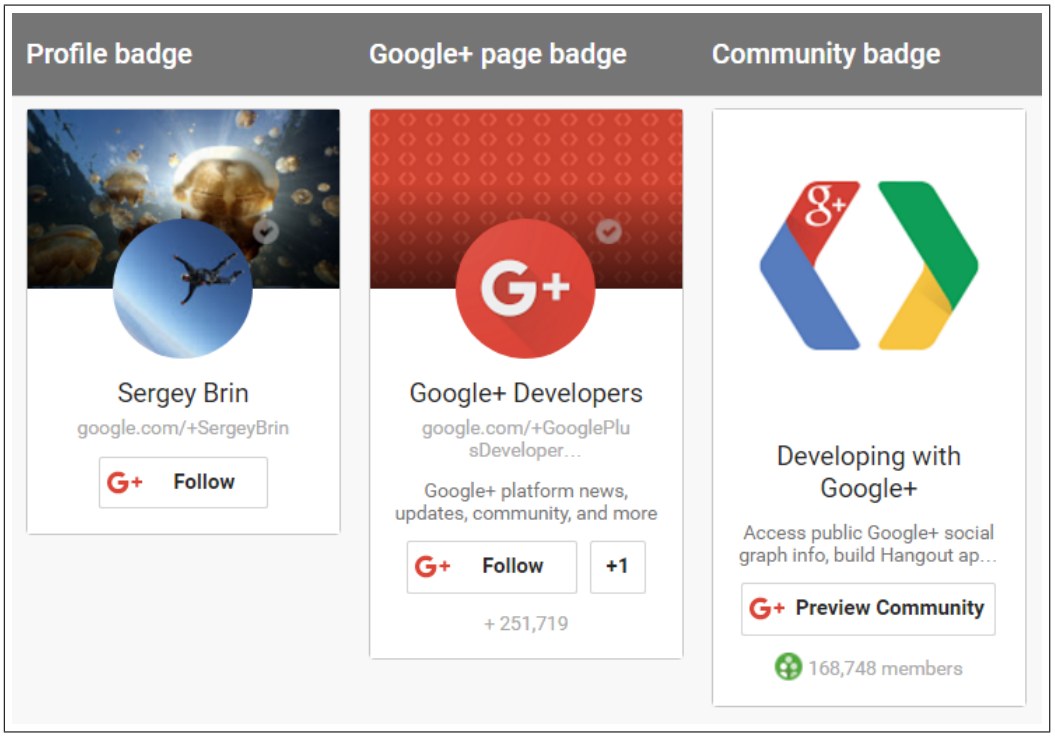
- .go(opt_container) : Spustí hledání elementů, které reprezentují daný plugin, a nalezené výskyty nahradí pluginem. Pokud parametr opt_container odkazuje na element, hledání je omezeno pouze na jeho obsah. V opačném případě je hledání spuštěno v rozsahu celého dokumentu.
- .render(container, parameters) : Vloží daný plugin s konfigurací určenou parametrem parameters do těla elementu obsaženém v parametru container.

Tlačítko +1, viz Obrázek 5.2 lze přidat na stránku v různých variantách lišících se například velikostí nebo způsobem zobrazení počtu doporučení. Přidáním +1 tlačítka na stránku je uživateli umožněno doporučení obsahu stránky kruhům uživatele a zvýšení návštěvnosti webové stránky i stránky na Google+.

[4]https://developers.google.com/+/web/api/javascript

Obrázek 5.2: Google+ tlačítko +1

Odznáček umožňuje vložit do stránky náhled na uživatelský profil, stránku nebo komunitu. Pomocí tohoto náhledu je, viz Obrázek 5.3, dále možné přímo ze stránky provádět relevantní činnosti dle typu odznáčku, který je automaticky zvolen podle zadané URL.



Obrázek 5.3: Ukázka variant odznáčků profilu, stránky a komunity

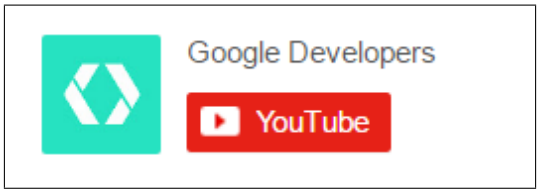
Odznáček je dále možné přizpůsobit pomocí několika možností, kterými například jsou:

- orientace : horizontální / vertikální,
- celková šířka,
- barevné rozvržení : světlé / tmavé.

Google+ dále nabízí široké REST API, pomocí kterého lze integrovat prostředky sítě do webové aplikace. Díky tomu mohou uživatelé maximálně využívat možností sítě Google+ přímo z aplikace[5].

5.3 YouTube

Platforma YouTube je spíše zaměřená na aspekty video obsahu a z pohledu sociální sítě nenabízí možnosti v míře srovnatelné s již zmíněnými sociálními sítěmi. V tomto ohledu YouTube nabízí pomocí JavaScriptové vývojářské platformy Google tlačítko odběru, viz Obrázek 5.4, které je vkládáno do stránky stejně jako pluginy sítě Google+.



Obrázek 5.4: Tlačítko odběru

Tlačítko odběru YouTube dále nabízí konfigurační možnosti zahrnující například velikost tlačítka nebo zobrazení počtu odběratelů. Pomocí YouTube lze také do stránky vložit existující video přímo pomocí elementu iframe se zvoleným videem.

YouTube naopak poskytuje širokou škálu možností pro vývoj aplikací ve formě rozsáhlých knihoven, pomocí kterých mohou vývojáři budovat YouTube integrace nebo celé aplikace založené na této platformě. Některé z těchto knihoven jsou[6]:

- API iframe přehrávače, pomocí kterého lze ovládat přehrávané video vložené ve stránce
- Data API, obsahující běžné možnosti YouTube, pomocí kterých lze například nahrávat videa, vytvářet a spravovat seznamy videí a další
- Streaming API, díky kterému lze plánovat a ovládat živé vysílání

[5] https://developers.google.com/+/web/api/rest/
[6] https://developers.google.com/youtube/documentation/

5.4 Elektronická pošta

Elektronická pošta (emaily) je jednou z nejranějších forem komunikace na internetu, která je využívána v nespočetném množství kontextů. V aktuální době jsou emailové schránky běžným způsobem pro rozpoznání uživatele internetu a díky nim lze realizovat například osobní nebo pracovní komunikaci (individuální pošta) nebo marketingové kampaně a předávání zpráv (hromadná pošta).

Většina poskytovatelů emailových schránek se snaží garantovat uživatelům pohodlí při využívání pošty, což zajišťuje například kontrolováním příchozí pošty na případně škodlivý či nevyžádaný obsah. Nevyžádaným obsahem, kterým jsou například reklamy nebo podvodné emaily, se emailová schránka nakazí velmi snadno a velmi těžce se jich zbavuje. Proto poskytovatelé emailových schránek při nálezu nevyžádanou poštu neumisťují mezi běžnou příchozí poštu ale do izolované složky, která se běžně nazývá "Spam".

Při návrhu automatizované emailové komunikace je proto třeba brát v potaz zásady hromadného odesílání emailů, mezi které patří[11]:

- Před odesílání emailů na schránku je třeba, aby uživatel souhlasil s odebíráním emailů.
- V každém emailu musí být obsažen odkaz, pomocí kterého lze odebírání jednoduše zrušit.
- Nepoužívat emailové schránky ze seznamů třetích stran. Toto vyplývá z prvního pravidla, tedy z nutnosti získání souhlasu uživatele.

6 Framework KiwiBlade

Framework KiwiBlade začal vznikat jako součást semestrální práce předmětu KIV/WEB a v rámci této práce byl rozšířen například o rozhraní pro práci s formuláři nebo o dynamickou rozšířitelnost pomocí modulů. KiwiBlade poskytuje programové rozhraní pro zpracování požadavku nebo pro hlášení chyb.

Framework KiwiBlade sjednocuje nejčastější operace prováděné v PHP aplikaci a poskytuje jednotný základ pro konfigurovatelné služby. KiwiBlade vychází z MVC návrhu a pomáhá sjednocovat akce oblasti do celků - do kontrolerů.

Pro generování těla HTTP odpovědi je standardně používán šablonovací systém Twig[1], pomocí kterého je implicitně hledána šablona. Twig je ve frameworku vytvářen pomocí rozšíření, které umožňuje jednotnou konfiguraci.

[1] http://twig.sensiolabs.org/

6.1 Struktura zdrojových souborů

Soubory obsahující zdrojové kódy jsou automaticky načítány v momentě jejich potřeby nástrojem, který je obsažený v balíkovacím nástroji Composer a který vychází ze standardu PSR-4. Toto opatření umožňuje vytvořit jednoznačné zobrazení jmenných prostorů na složky a díky tomu lze při hledání zdrojového souboru požadované třídy použít její plně kvalifikované jméno. Konfigurace automatického načítání souborů se nachází v souboru `composer.json`. Ukázka konfigurace automatického naćítání pro aplikaci MojeFavka je znázorněna Kódem code.composer-autoload:

```
{
  "autoload": {
    "psr-4": {
      "KiwiBlade\\": "libs/sestep/kiwi-blade/src/",
      "KiwiBladeTests\\": "libs/sestep/kiwi-blade/
         tests/"
    },
    "classmap": ["libs/"]
  }
}
```

Kód 6.1: Konfigurace automatického načítání

Pokud při takovéto konfiguraci vznikne dotaz na existenci třídy, jejíž plně kvalifikované jméno je `KiwiBlade\DI\Container`, nástroj zjistí že se jedná o jmenný prostor namapovaný na složku `libs/sestep/kiwi-blade/src/`. Ze jména třídy odebere název mapovaného jmenného prostoru a zbytek připojí k namapované složce: `/libs/sestep/kiwi-blade/src/DI/Container.php`. Ve zkratce to znamená, že struktura jmenných prostorů musí odpovídat adresářové struktuře, ve které jsou soubory uloženy.

Výhoda tohoto načítání se projevuje při vývoji, kdy nově přidané třídy do adresářové struktury s odpovídajícími jmennými prostory lze automaticky načítat bez potřeby vytváření nových struktur pro načítání souborů. Další výhodou je například zvýšená přehlednost, protože umístění každého souboru lze jednoduše zjistit i bez použití pokročilých vývojových nástrojů. Pro optimalizaci při přechodu na produkční prostředí lze použít Composer pro kompilaci cest k souborům do mapy tříd. Mapa tříd je asociační pole a funkce, která v něm dohledává požadované třídy. Tím pádem při načítání souboru pro požadovanou třídu lze přímo určit existenci a umístění souboru se zdrojovým kódem v konstantním čase.

6.2 Architektura frameworku

Framework je rozdělen do logických celků, které se zabývají určitou problematikou. Každý celek se nachází ve vlastním jmenném prostoru, jehož jméno odpovídá dané problematice. Problematiky, které KiwiBlade řeší, například jsou:

- DI (Dependency Injection) - správa a skládání závislostí,
- Http - práce s url a s požadavkem, generování odpovědi,
- Mail - odesílání mailů a statistiky,
- Storage - ukládání a načítání souborů.

Hlavní myšlenkou KiwiBlade je rozšiřitelnost a konfigurovatelnost a proto KiwiBlade není navržen, aby vycházel z jednoho konkrétního jádra - místo toho KiwiBlade motivuje k rozšiřování, nikoliv pomocí dědičnosti ale pomocí skládání. Dokonce i vlastní části KiwiBlade vznikají pomocí rozšíření.

Nejmenší částí aplikace je služba, což je třída starající se o konkrétní problematiku, například ukádání informací do relačního úložiště. Služby mohou být pojmenované, což slouží pro jejich identifikaci v konfiguraci a mohou být vytvořeny těmito způsoby:

Ideální způsob pro vytváření služeb je pomocí kontejnerových rozšíření, která dědí od abstrakce AContainerExtension. Takovéto rozšíření je nutno zaregistrovat v konfiguračním souboru vložením jeho FQN (Fully Qualified Name), neboli plně kvalifikovaného jména, do konfiguračního pole extensions pod klíč představující jméno tohoto rozšíření. Jméno rozšíření dále určuje sekci konfigurace, která tomuto rozšíření náleží.

Vstupním bodem aplikace založené na KiwiBlade je Konfigurátor, tedy třída načítající konfigurační soubory. Jejím cílem je vytvoření Kontejneru a zaregistrování požadovaných rozšíření. Použití Konfigurátoru je předvedeno Kódem 6.2.

```
$configurator = new \KiwiBlade\DI\Configurator(
   __DIR__ . '/config/');

// Application configuration
$configurator->addConfig('app');
// Server configuration
$configurator->addFirstExistingConfig(['local']);

$container = $configurator->createContainer();
```

Kód 6.2: Příklad vytvoření a použití konfigurátoru

Takto vytvořený Kontejner je klíčový prvek, který poskytuje závislosti, neboli jinak řečeno služby, které jsou nezbytně nutné pro funkčnost dalších služeb. Závislosti a argumenty jsou službám poskytovány pomocí konstruktoru zcela transparentně. Kontejner mimo služeb obsahuje i parametry služeb a parametry aplikace. Struktura konfigurace je popsána v následující kapitole.

Přístup ke službám registrovaným v kontejneru vychází z návrhu na standard kontejnerového rozhraní[2]. Toto rozhraní definuje dvě metody: `has($id)`, která určuje, zda-li kontejner obsahuje požadovanou službu, a `get($id)`, která se pokusí tuto službu získat. Rozhraní také definuje výjimky pro stavy, které mohou nastat a které ovlivňující tyto metody. Kontejner je navržený tak, že identifikátor služby je FQN třídy představující danou službu. Tímto opatřením je zajištěno, že nenastane kolize služeb protože které pro každou existující třídu musí být FQN unikátní.

Kontejner je zároveň navržený aby podporoval LazyLoading, což je přístup načítání prostředků až v momentě, kdy jsou potřeba. Toho je docíleno tím, že při inicializaci se do kontejneru dostane pouze postup pro získání žádané služby. Tento postup je vykonán poprvé, kdy je kontejner požádán o danou službu. V každé následující žádosti o službu, pokud není určeno jinak, kontejner vrací odkaz na službu vytvořenou při první žádosti.

Dalším klíčovým prvkem je Kontrolér (třída Controller). Kontrolér je třída starající se o zpracování uživatelských vstupů a o přípravu výstupů pro pohled. V kontroléru jsou funkce například pro přesměrování na jinou stránku nebo pro vytvoření flash zprávy a další metody pro běžné potřebných činností. Další metody kontroléru představují akce dosažitelné pomocí URL. Akce jsou v kontrolérech představovány metodami a mohou mít vý-

[2]https://github.com/container-interop/fig-standards/blob/container-configuration/proposed/container.md

konnou a/nebo zobrazovací povahu. Název metody pohledové akce musí být ve tvaru `action<jménoAkce>` a název metody zobrazovací musí být ve tvaru `render<jménoAkce>`. Použití zobrazovací akce předpokládá, že ve složce se šablonami aktuálního kontroléru bude šablona stejného jména jako je jméno akce.

6.3 Konfigurace KiwiBlade

Framework KiwiBlade lze konfigurovat pomocí strukturovaných souborů ve formátu PHP polí. Jednotlivé sekce, neboli služby, tohoto frameworku lze individuálně konfigurovat pomocí vyhrazených sekcí indentifikovaných unikátním názvem. Tyto konkrétní konfigurace lze přiřazovat podle nastaveného jména služby.

Ukázka konfigurace administrace aplikace Moje Favka je znázorněn v Kódu 6.3.

```
<?php
return [
  'parameters' => [
    'niceUrl' => false,
  ],
  'kb' => [
    'dispatcher' => [
      'controllerFormat' => "MojeFavka\\Controllers
        \\%cController",
      'errorController' => \MojeFavka\Controllers\
        ErrorController::class,
    ],
    'request' => [
      'niceUrl' => '%niceUrl%',
      'wwwSubfolder' => '/admin/',
      'defaultController' => 'dashboard',
    ],
  ],
  'configFiles' => [
    'social',
  ],
];
```

Kód 6.3: Příklad konfigurace KiwiBlade

V útržku kódu je vidět na začátku pole parametrů, kterými lze přizpůsobit aplikaci prostředí, ve kterém poběží. Zde je je parametr, který určuje, zda-li se mají použít pěkné URL.

Další sekce konfigurace je pro framework KiwiBlade, zde jsou nastaveny služby pro zpracování požadavku `Dispatcher` - tato služba se stará o nalezení kontroleru a jeho spuštení a případné zachycení, zpracování a zobrazení chyby. Následuje konfigurace pro službu `Request`, která má na starost předání HTTP požadavku do aplikace. Její vytváření je parametrizováno argumentem *niceUrl*, jehož hodnota je `%niceUrl%`. Při použití tohoto zápisu je hodnota nahrazena stejnojmenným parametrem aplikace. Poslední sekcí v konfiguračním souboru zvaná *configFiles* určuje jména dalších konfiguračních souborů, ze kterých má být načtena dodatečná konfigurace.

6.4 Vyřízení požadavku

Požadavek je v KiwiBlade reprezentován instancí třídy `Request`, která sjednocuje přístup do polí parametrů požadavku jako je *GET* nebo *POST* a přístup k tělu požadavku. Mimo standardních parametrů požadavku Request schraňuje například informace o URL serveru nebo aktuálním páru kontrolér-akce.

Tělo požadavku může být tvořeno textem v prosté podobě nebo zakódovanými daty například v JSON podobě a Request umožňuje číst tělo v těchto dvou podobách. Požadavek je následně předán dispečerovi (třída Dispatcher), který vytvoří instanci požadovaného kontroléru a pokusí se spustit požadovanou akci. Pokud není k nalezení požadovaný kontrolér nebo nalezený kontrolér neobsahuje požadovanou akci, dispečer požadavek zpracuje jako chybný pomocí chybového kontroléru. Tento kontrolér pro chybný požadavek zobrazí odpovídající chybové hlášení.

Dispečer spouští metody aktuálního kontroléru v tomto pořadí:

1. `startup()` metoda slouží pro připravení proměnných a dalších služeb použitých v kontroléru
2. [`action<jménoAkce>()`]slouží pro zpracování údajů požadavku
3. `beforeRender()` metoda slouží například pro dodatečné nastavení proměnných pro pohled, v momentě, kdy je téměř jisté že bude používán šablonovcí systém
4. [`render<jménoAkce>()`]by neměla obsahovat výkonnou logiku a místo toho by měla například jen získat potřebná data z databáze, respektive ověřit správnost požadavku

Metody v hranatých závorkách jsou volány pouze pokud jsou v aktuálním kontroléru definovány.

6.5 Práce s URL

KiwiBlade je připravený pro práci s klasickými URL adresami i s 'pěknými' adresami. Pěkné URL vyžadují přepisování požadavků na hlavní skript a obsahují parametry controller a action zakomponované do části URL *cesta k souboru*.

Zda-li má framework využívat pěkné URL je určeno parametrem *niceUrl* v konfiguračním souboru a na základě tohoto parametru je volen algoritmus zpracování URL požadavku a vytváření URL.

Programové určení lokace je ve formátu `[controller]:[action]`, kde pokud je některá z částí vynechána, je doplněna defaultní hodnotou. Lokace také může být ve formátu `action`, při čemž se k dané akci doplní aktuální kontrolér. Odkaz je možné vytvořit pomocí funkce `link` v kontroléru, funkce `link` služby `LinkGenerator` nebo případně pomocí funkce `link` v šabloně.

7 Moje Favka

Webová aplikace MojeFavka je určena pro uchazeče o studium na Fakultě Aplikovaných Věd. Prostřednictvím MojíFavky mají uchazeči možnost si nechat stručně představit jednotlivé obory, Fakultu Aplikovaných Věd nebo zajímavá místa Plzně.

Jednou z částí MojíFavky je "Šancomat", což je nástroj počítající uchazečovu šanci na přijetí z výsledků středoškolského studia podle aktuálních podmínek přijetí ke studiu pro daný akademický rok.

Jednou z částí MojíFavky je "Šancomat", což je nástroj počítající uchazečovu šanci na přijetí z výsledků středoškolského studia podle aktuálních podmínek přijetí ke studiu daného akademického roku. Mimo prezentace studia na fakultě nyní MojeFavka také slouží jako prostředník pro vytváření odběrů novinek týkajících se přijímacího řízení a jejich rušení.

Tento obsah se nachází v uživatelské části, která je postavena na frameworku AngularJS. V rámci této práce dále vznikla administrační část, postavená na frameworku KiwiBlade, díky které lze spravovat rozesílání emailů a další aspekty aplikace.

7.1 Architektura aplikace

Před začátkem návrhů úprav a jejich implementací existovala aplikace navržená tak, aby využívala dědičnost $scopů. Díky tomuto faktu bylo možné sdílet funkcionalitu, jako například detekce typu uživatelského prohlížeče nebo funkčnost pro navigování mezi stránkami, mezi hlavním kontrolérem a jednotlivými kontroléry pohledů. Tento návrh není optimální pro jednotkové testování protože kompozice kódu není zcela transparentní.

Dalším faktorem odlišným v původní aplikaci je struktura kódu - před začátkem úprav se zdrojový kód nacházel v globálním kontextu, což představovalo riziko nežádoucích kolizí jmen proměnných a funkcí. Aplikace je ve výsledném stavu strukturována do jednotlivých modulů vždy obsahujících pouze konkrétní část aplikace. Tyto zdrojové soubory jsou načítány pomocí správce modulů RequireJS a jejich struktura je k tomu patřičně upravena.

Každá část potom co nejlépe respektuje návrh AngularJS například tím, že je sdílená funkčnost přenášena pomocí vkládání závislostí namísto dědičnosti $scopů.

Před úpravami využívala aplikace pro routování modul *ngRoute*, který mapuje URL adresy přímo na pohledy. Místo něj je po aplikovaných změnách

použit modul *ui.router*, který mapuje URL adresy na stavy a pro jednotlivé stavy definuje pohledy.

Aplikace je potom vázaná na stavy a nikoliv na URL adresy a díky tomu je možné například měnit URL adresy na jednotném místě, bez potřeby zásahu do aplikační logiky nebo do šablon. Dále lze s pomocí ui.routeru aplikaci popsat jako stavový automat a jasněji popsat přechody mezi jednotlivými stavy.

Další předností *ui.routeru* a vázání aplikace na stavy jsou například direktivy, které usnadňují navigaci a určování aktuálního stavu, což je využito například v navigačním menu nebo v komponentě znázorňující aktuální stav.

V rámci této práce byly za účelem dekompozice šablon vytvořeny tyto direktivy:

- **Guidepost** je komponenta rozcestníku, která obaluje pozadí, jednotlivé ukazatele a klikací mapu. Dále se direktiva Guidepost stará o zvýraznění ukazatele, nad kterým je kurzor a o odkazy na stránky oborů.
- **PageControls** je komponenta zobrazující se chvíli po načtení stránky na 5 vteřin, která dává uživateli na vědomí možné způsoby navigace stránkou.
- **TopMenu** je komponenta navigačního menu, která se stará o zobrazení odkazů na jednotlivé části aplikace a o zvýraznění aktuálně zobrazené stránky.

Dále za účelem dekompozice funkční logiky vznikly atributové direktivy:

- **Title**: direktiva obnovující název stránky při změně pohledu,
- **Navigation**: direktiva obalující aplikační logiku týkající se navigace na stránce.

7.1.1 MojeFavka API

Aplikační programové rozhraní aplikace MojeFavka vzniklo v rámci této práce je využíváno MojíFavkou pro získávání informací o sociálních sítích představených kateder a fakulty a pro zpracování požadavků odběratelů.

Rozhraní aplikace MojeFavka je typu REST a definuje následující endpointy, které aplikace využívá pro zprostředkování dynamického obsahu:

```
GET /socials - získání informací o sociálních sítích
    subjektů fakulty
```

```
POST /subscribe/register - vytvoření odběru daného
   typu pro emailovou adresu
  parametry:
    email :string
    type  :string
```

```
POST /subscribe/cancel - zrušení odběru zadané
   adresy
  parametry:
    email :string
    code  :string - kód ověřující odhlášení odběru
```

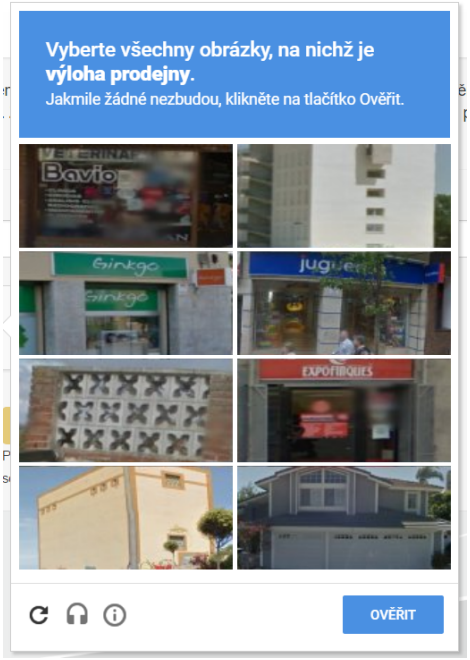
Implementace rozhraní vychází z frameworku KiwiBlade, z něhož používá například třídu Request pro práci s požadavkem. Rozhraní také využívá služeb z administrační část skrze kontejner závislostí. Všechny odpovědi tohoto API jsou formátované do JSON řetězců, což usnadňuje jejich zpracování.

Struktura je opět rozdělená do celků, do Api Kontrolérů, které se liší od standardních kontrolérů pojmenováním metod, které představují jednotlivé akce. Název metody se v těchto kontrolérech skládá z metody HTTP požadavku a ze jména akce. Například tedy endpoint `GET /socials` je implementovaný v kontroléru `SocialsApiController` metodou `getDefault()`.

7.2 Odebírání novinek

Novou sekcí, která byla vytvořena v rámci této práce, je odebírání novinek, pomocí které mohou uživatelé vytvářet odběry skrze požadavky na MojeFavka API.

Při vytváření odběru uživatel zadává emailovou adresu, na kterou chce získávat novinky, a zároveň vybírá skupinu odběratelů, do které spadá. Odběratelské skupiny v rámci Mojí Favky jsou *Uchazeč*, *Rodič* a *Vyučující*. Tyto skupiny slouží pro rozlišení odběratelů, kterým lze posílat cílenější zprávy. Uživatel před pokusem o vytvoření odběru musí složit test ověřující, že odběr vytváří skutečný člověk a ne robot. Tento test je založený na rozpoznávání obrázků pomocí technologie ReCaptcha. Ukázka testu ReCaptcha je znázorněna Obrázkem 7.1.



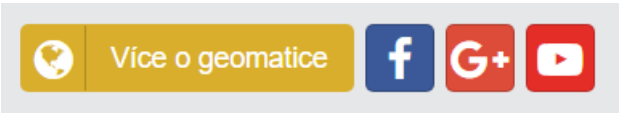
Obrázek 7.1: Ukázka testu ReCaptcha

Výsledek požadavku o vytvoření odběru, úspěšný i neúspěšný, je zobrazen uživateli okamžitě po zpracování. V případě úspěšného vytvoření odběru je zadaná emailová adresa vložena do databáze a při automaticky spouštěném skriptu rozesílání je odběr porovnáván s nastavenými podmínkami odeslání naplánovaných emailů.

V případě, že uživatel již nemá zájem dostávat oznámení z aplikace, má možnost se odhlásit pomocí odkazu, který je obsažen v každém emailu.

7.3 Provázání se sociálními sítěmi

Prvním krokem provázání aplikace se sociálními sítěmi je uvědomění, že sociální sítě existují. Za tímto účelem byla vytvořena direktiva *mfSocialWidget*, která řeší zobrazování tlačítek dostupných sítí jednotlivých subjektů.



Obrázek 7.2: Direktiva mfSocialWidget

Na Obrázku 7.2 je znázorněna direktiva zobrazující tlačítka sociálních sítí Facebook, Google+ a YouTube a tlačítko odkazující na webové stránky subjektu.

Tlačítka jsou realizována pomocí pluginu knihovny *Social Buttons for Bootstrap*[1], která definuje vzhled sociálních tlačítek. Ikony obsažené v jednotlivých tlačítkách jsou použity z knihovny ikon *Font Awesome*[2].

Jako další krok provázání bylo zvoleno vložení pluginů poskytnutých sociálními sítěmi. Sítě Facebook a Google+ poskytují JavaScriptové knihovny, které poskytují programové rozhraní pro úpravu obsahu dokumentu.

Tyto knihovny při spuštění prohledají dokument ve snaze nalézt elementy, které mají nastavenou relevantní třídu. Pro každý nalezený element knihona zpracuje atributy elementu a použije je pro načtení pluginu, který vloží na místo nalezeného elementu. Jako vložený element se běžně používá element iframe, který zaručuje jednotný vzhled a chování pluginu díky tomu, že je obsah elementu *iframe* izolovaný od stylů a skriptů hlavní stránky.

Pro demonstraci jsou integrované pluginy zobrazeny v modálním okně, což je komponenta frameworku *Bootstrap* představující dialogové okno v rámci HTML stránky. Díky tomu se předešlo výrazným změnám vzhledu stránek. Knihovny zobrazující pluginy jsou sjednoceny do jediné služby `socialParser`, která obaluje funkčnost zobrazování pluginů do asynchronního zpracování.

7.3.1 Facebook SDK

Knihovna Facebook SDK je standardně načtena asynchronně pomocí elementu *script* v dokumentu[3], při použití RequireJS je možné knihovnu načíst jako jakýkoliv jiný modul.

```
paths: {
  // ...
  FB: [
    '//connect.facebook.net/cs_CZ/sdk',
    '../fallback/fbSDK'
  ]
}
```

Kód 7.1: RequireJS konfigurace modulu Facebook SDK

[1] https://lipis.github.io/bootstrap-social/
[2] http://fontawesome.io/
[3] https://developers.facebook.com/docs/javascript/quickstart

RequireJS modul Facebook SDK je v aplikaci MojeFavka pojmenován FB a náleží mu, viz Kód 7.1, dvě cesty: první cesta je URL získávající skutečnou knihovnu a druhá cesta získává záložní soubor knihovny, který se načte pokud při načítání skriptu z první cesty nastane problém. Díky tomuto opatření může být aplikace spuštěna i pokud je knihovna Facebook SDK nedostupná.

Facebook SDK je třeba inicializovat spuštěním metody *init*, jejíž parametry slouží například pro určení verze API nebo aplikačního klíče.

U pluginů Facebooku se předpokládá, že DOM obsahuje elementy připravené k nahrazení pluginem a že postačí zpracovat DOM při načtení skriptu SDK. Povaha aplikací vycházejících z frameworku AngularJS tento předpoklad nesplňuje a elementy k nahrazení jsou do DOMu přidány při navigování napříč aplikací.

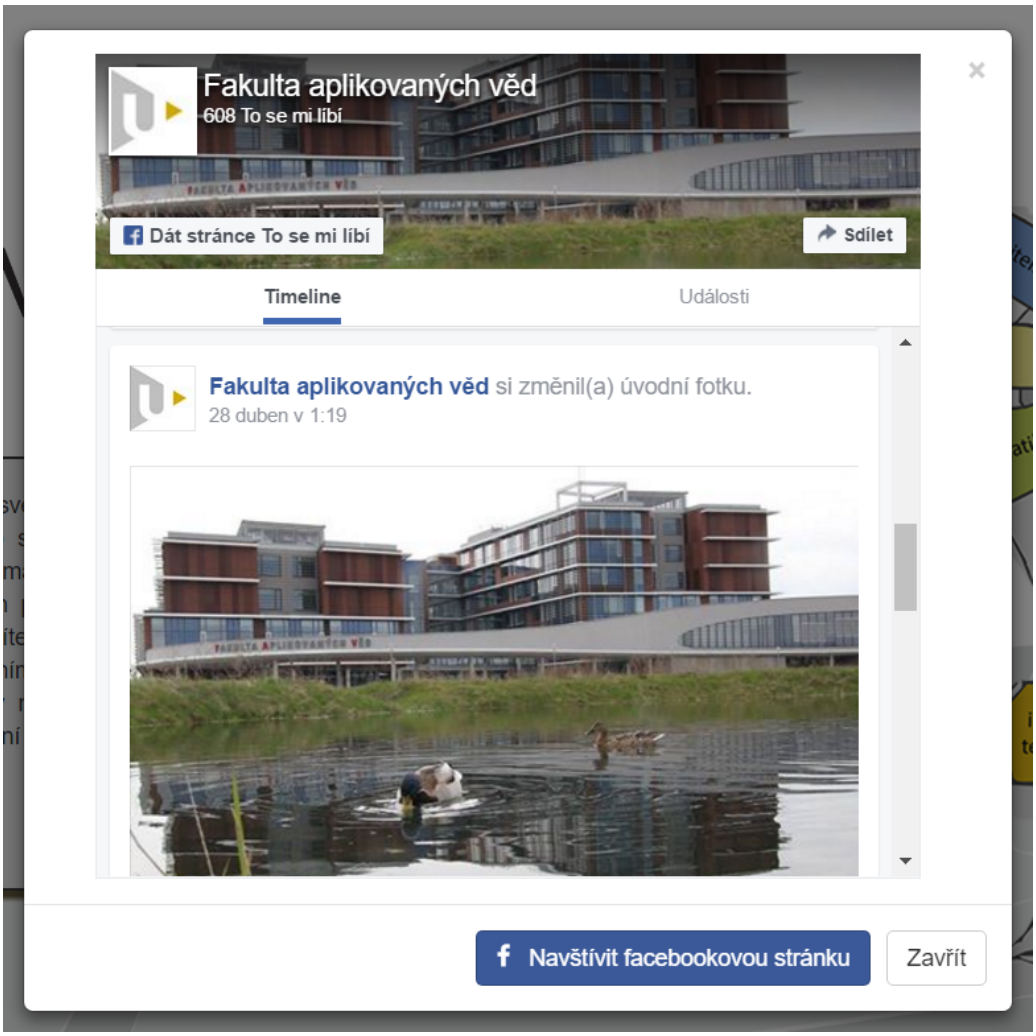
Facebook SDK však umožňuje ruční spuštění zpracování DOMu, případně nad konkrétním elementem, pomocí funkcí globálně dostupné knihovny FB. Ukázka ručního spuštění zpracování v AngularJS direktivně je znázorněna Kódem 7.2.

```
function init($scope, $element) {
  $timeout(function() {
    // retrieves javascript reference to DOM element
        from jQuery wrapper
    var domEl = $element[0];

    FB.XFBML.parse(domEl);
  });
}
```

Kód 7.2: Ruční spuštění zpracování DOMu

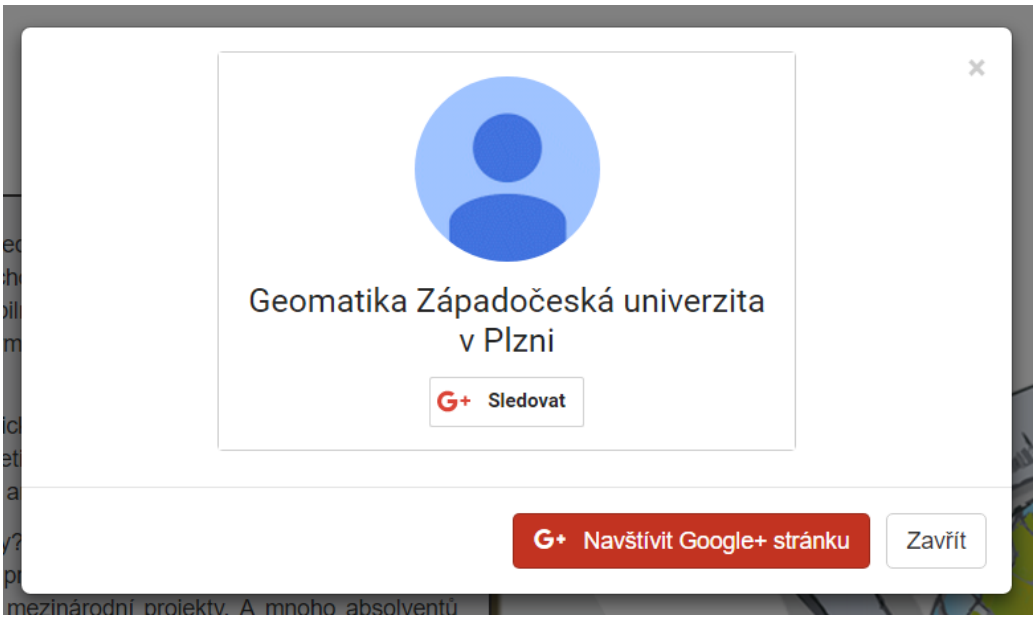
Plugin facebookové stránky je realizován pomocí elementu s nastavenou třídou *fb-page* a s atributem *data-href* obsahujícím odkaz na facebookovou stránku subjektu. Tento element je vložen do těla modálního okna, jehož obsah je následně zpracován knihovnou. Výsledná zobrazovaná komponenta je znázorněna Obrázkem 7.3.



Obrázek 7.3: Náhled na facebookovou stránku Fakulty Aplikovaných Věd

7.3.2 Google platforma

Načítání JavaScriptové platformy Google je realizováno stejně jako načítání Facebook SDK. Její konfigurace však probíhá okamžitě pro načtení skriptu a proto je nutné nastavit její parametry před načtením. V aplikaci MojeFavka je toto realizováno ve vstupním souboru *main.js*, kde je nastaven jazyk pluginů a způsob hledání pluginových elementů.



Obrázek 7.4: Ukázka odznáčku Google+

Google+ pluginy fungují na podobném principu - knihovna hledá elementy relevantních tříd a do každého nálezu vloží odpovídající plugin. Obsah modálního okna, který byl zpracován platformou, je znázorněn Obrázkem 7.4.

7.4 Administrace

Administrace aplikace MojeFavka slouží pro správu a konfiguraci různých aspektů aplikace MojeFavka. Skrze administraci lze vytvářet a spravovat typy emailů, sledovat statistiky typů emailů, odeslaných emailů a odběratelů nebo zobrazit přehled odběratelů či náhled na nakonfigurované sociální pluginy.

Architektura administrace vychází z frameworku KiwiBlade a řeší samotné vyřizování požadavků pomocí rozhraní definovaných frameworkem. Struktura administrace se dělí na sekce reprezentované kontroléry, které zpracovávají uživatelské požadavky a připravují výstupy. Tyto výstupy jsou následně sestavovány pomocí Twig šablon, čímž vznikne dokument zobrazovaný uživateli. Šablony administrace spadají do jmenných prostorů podle jejich významů:

- **@views** - šablony konkrétních pohledů,
- **@layouts** - aplikační šablony společné pro většinu pohledů,
- **@controls** - šablony komplexnějších prvků stránky, které lze díky oddělení používat vícenásobně, těmito prvky jsou například přihlašovací formulář nebo různé formuláře pro editaci datových struktur
- **@mails** - šablony spojené s odesíláním emailů.

Každý z těchto jmenných prostorů má přiřazenou právě jednu složku obsahující individuální šablony. Například jmenný prostor *@layouts* má přiřazenou složku `/admin/templates/_layouts/`. Když se aplikace pomocí Twigu pokusí získat šablonu *@layouts/body.twig*, Twig bude hledat soubor šablony `./admin/templates/_layouts/body.twig`.

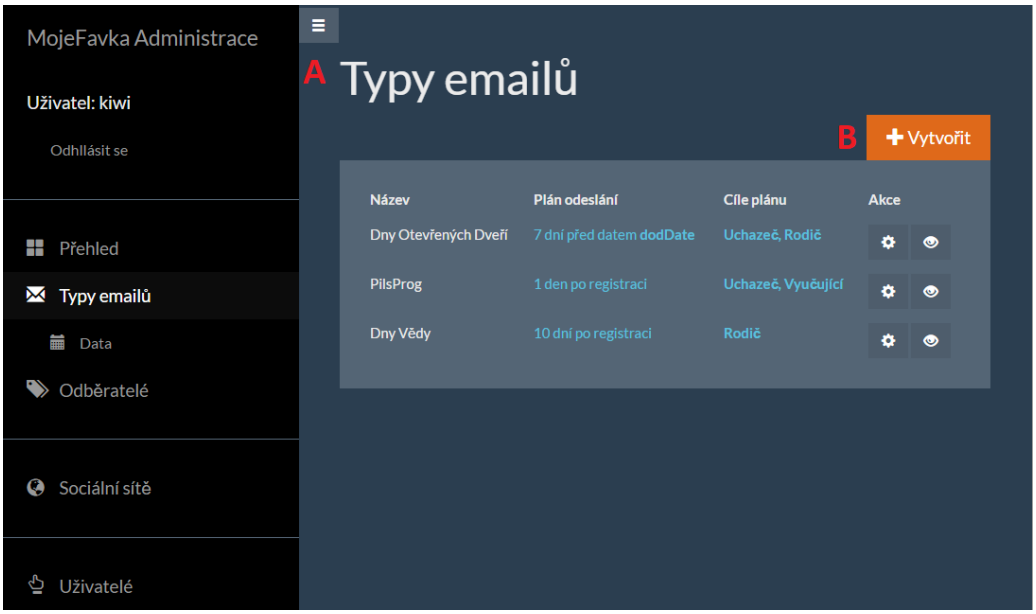
Uživatelské rozhraní administrace je tvořeno pomocí komponent nabízených frameworkem Bootstrap, jejichž vzhled je přizpůsobený stylem Superhero od Bootswatch[4].

Rozvržení stránky vychází ze šablony Simple Sidebar od Start Bootstrap[5], viz Obrázek 7.5 (A), které lze skrývat a které se automaticky skrývá v režimu úzkého zobrazení.

Obsah postranního menu je definován programaticky pomocí souboru tříd `SeStep\Navigation`. Součástí uživatelského rozhraní administrace jsou také takzvaná Call To Action tlačítka, neboli tlačítka vyzývající uživatele k nějaké akci nebo činnosti. Jedná se o tlačítka umístěna ve vrchní části stránky, kde na sebe strhávají pozornost, znázorněny na Obrázku 7.5 (B).

[4]https://bootswatch.com/superhero/

[5]https://startbootstrap.com



Obrázek 7.5: Rozvržení administrace

7.4.1 Autentizace

V administraci se mohou vyskytovat důvěrné údaje a lze v ní upravovat plány rozesílání mailů a proto je potřeba aby byl obsah administrace přístupný pouze pověřeným správcům. Před umožněním přístupu k těmto operacím aplikace vyžaduje po uživateli autentizaci formou přihlašovacího formuláře.

Přihlášení do administrace je možné pomocí typické kombinace jméno a heslo anebo pomocí nástroje WebAuth, který poskytuje jednotné přihlášení (Single Sign On) napříč aplikacemi provozovanými univerzitou.

Přihlášení pomocí jména a hesla je realizováno pomocí šifry Bcrypt nativně implementované v PHP od verze 5.5. Bcrypt je funkce pro jednosměrné šifrování hesel. Je postavena na šifrování Blowfish, což je symetrická bloková šifra proti které stále není známá žádná efektivní metoda prolomení. Bcrypt ovšem zašifruje heslo jednosměrně a proto se jedná o funkci hashovací.

```
bcrypt('password', 12) =
$2y$12$6VeyNWb2fjZWLBi176z3U.GB/V7i9nxHdv03ZYhqc8t4cLiwLFNGC
(a)

 2y (b)
(c) 12
  (d)  6VeyNWb2fjZWLBi176z3U.
                          (e)  GB/V7i9nxHdv03ZYhqc8t4cLiwLFNGC
```

Kód 7.3: Ukázka otisku generovaného šifrou Bcrypt

Ukázka výsledného otisku generovaného funkcí Bcrypt je znázorněna Kódem 7.3 (a), který se skládá ze tří částí oddělených znakem dolar.

Těmito částmi jsou identifikátor hashovacího algoritmu, parametr algoritmu a hodnota zakódovaná do soustavy base64, na Kódu 7.3 znázorněny popořadě (b), (c) a (d)+(e).

Zakódovaná hodnota je dále složena ze dvou částí: ze soli (d), která má vždy 22 znaků, a z otisku (e), kterému připadá zbývajících 31 znaků hodnoty[6].

Identifikátoru algoritmu Bcrypt připadají hodnoty 2a, 2x, 2y případně jiné varianty vždy začínající číslem 2. Důvod vzniku různých variant tohoto algoritmu spočívá v chybné implementaci v jazyce PHP. Při objevení chyby měly být vadné otisky označeny hodnotou 2x a nové otisky generované opraveným algoritmem 2y, díky čemuž je možné ověřovat i otisky nevyplývající z původní specifikace algoritmu [7] [8].

Parametr tohoto algoritmu je náročnost, nebo také cena, pro vypočítání otisku a ověření jeho správnosti. Parametr ceny, často také nazývaný cost, je celé číslo v rozsahu <4, 31>. Každá hodnota o jedna vyšší než předchozí exponenciálně zvyšuje dobu výpočtu otisku zhruba na dvojnásobek, což vychází z podstaty tohoto algoritmu: měl by zpomalovat útoky hrubou silou.

Říká se, že výkon počítačů zvyšuje na dvojnásobek každé dva roky[13] a tím pádem klesá doba na ověření hesla určité úrovně. V takovém případě je možné zvýšit požadovanou výpočetní náročnost. Jelikož ale aplikace z podstaty bezpečnosti nesmí znát uživatelovo heslo v jiném případě, než při uživatelově přihlášení, otisk nelze vygenerovat při úpravě parametru. Namísto toho je otisk generován právě při uživatelově příštím přihlášení, po ověření správnosti hesla pomocí otisku starého.

Pozn.: Při použití PHP implementace algoritmu Bcrypt je třeba dbát na fakt, že hashovaná hesla jsou oříznuta na 72 znaků[9].

Přihlášení pomocí univerzitního účtu Orion neřeší zabezpečení na straně aplikace a místo toho webový server před spuštěním skriptu na určené URL zajistí, že je uživatel přihlášen pomocí jednotného přihlášení. Po ověření přihlášeného uživatele začne server zpracovávat požadavek pomocí požadovaného skriptu a přidá do proměnné $_SERVER identifikaci aktuálního uživatele, které aplikace může důvěřovat.

[6] http://stackoverflow.com/questions/5881169/
[7] http://stackoverflow.com/questions/15733196
[8] http://seclists.org/oss-sec/2011/q2/632
[9] http://php.net/manual/en/function.password-hash.php

Jelikož Orion Loginem disponuje každý student nebo učitel, je třeba ověřit, že je uživatel oprávněn pracovat s administrací. V aplikaci je toto ověření realizováno hledáním přihlašovacího jména v seznamu jmen uživatelů, kteří jsou oprávněni vstupovat do aplikace, na úrovni databáze.

7.4.2 Správa emailů

Hlavní sekce administrace se zabývá plánováním emailů a což zahrnuje vytváření typů emailů a vytváření plánů odeslání. Typy emailů se od sebe rozlišují názvem a dále obsahují předmět zprávy a šablonu těla zprávy, která může být prostý text nebo html dokument. U typu emailu může být přiřazen plán odeslání, který se skládá z podmínky, při jejíž splnění je email odeslán, a z výčtu typů odběratelů, kterým má být daný typ emailu odeslán.

První pohled této sekce je stručný výpis typů emailů s jejich plány odesílání. V každém řádku je znázorněno, viz Obrázek 7.5, jméno typu emailu, podmínka pro odeslání emailu a cílená skupina, která má obdržet email při splnění podmínek pro odeslání. Pro každý email je odsud dále možné zahájit úpravy typu emailu nebo zobrazit jeho náhled.

Úprava typu emailu, znázorněna Obrázkem 7.6, se skládá z konfigurace vlastností typu emailu, jimiž jsou popisek typu, předmět zprávy emailu a tělo zprávy emailu. Tělo emailu může být prostý text nebo html a případně může obsahovat i element *style* pro dodatečnou definici vzhledu emailu. Tělo emailu může být použito jako šablona Twigu, který poskytuje široké šablonovací možnosti a zároveň neotevírá možnost vykonávat PHP kód.

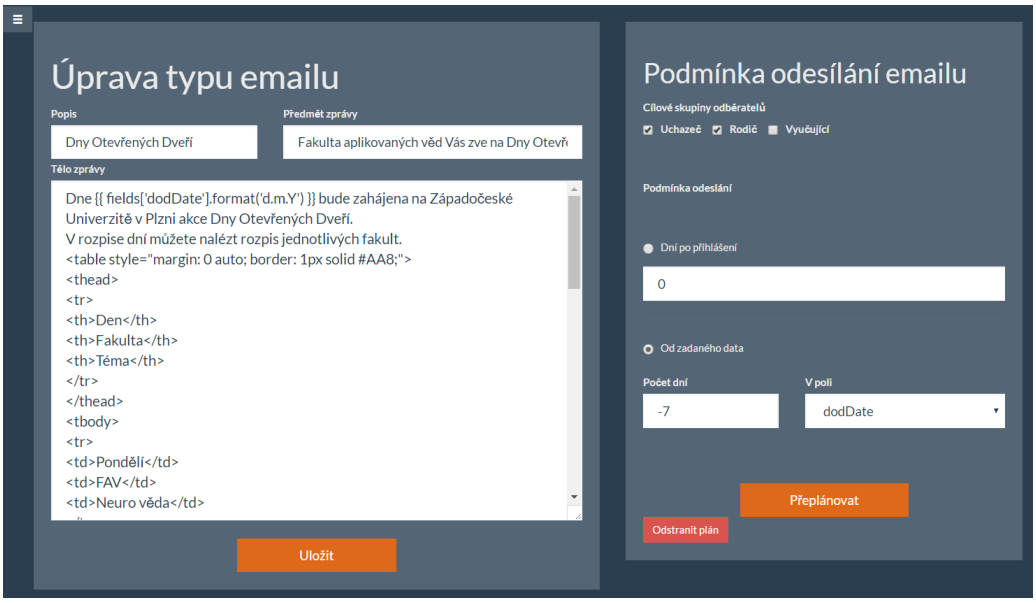
Další část úpravy typu emailu je naplánování jeho odesílání, respektive zrušení tohoto plánu. Plán odesílání emailu se skládá z výčtu cílových skupin odběratelů a konfigurace podmínky, při jejíž splnění má být email odeslán.

Každá podmínka odeslání emailů spadá do jednoho z typů podmínek a může mít parametry splnění. V rámci této práce byly vytvořeny dva podmínek:

- N dní po přihlášení se k odběru,
- N dní od data určeného v poli P.

Data pro podmínku druhého typu jsou specifikována pomocí souboru `./files/schedules/schedule_XXXX-YYYY.json`, kde XXXX a YYYY značí roky začátku a konce aktuálního ročníku.

Vytvoření nového emailového typu je možné pomocí skrze pohled téměř stejný, jako je úprava emailového typu, s tím rozdílem, že není k dispozici vytváření podmínky odeslání.



Obrázek 7.6: Úprava typu emailu

Datová struktura je navržená tak, aby bylo v případě potřeby možné rozšířit aplikaci o možnost vytvořit více podmínek odeslání jednoho emailu.

7.4.3 Další pohledy

Výchozí pohled administrace je nástěnka, ve které jsou zobrazeny statistiky aplikace týkající se například existujících typů emailů, emailů odeslaných a připravených k odeslání nebo počtů odběratelů.

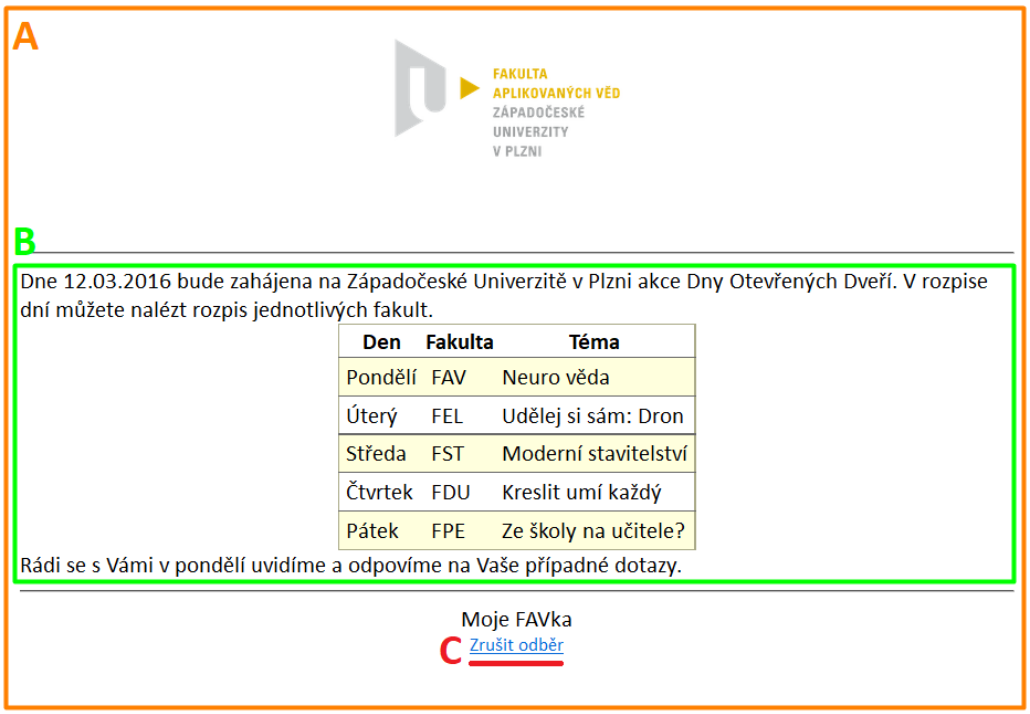
Administrace dále obsahuje pohled *Sociální Sítě*, který umožňuje okamžitou kontrolu konfigurace sociálních sítí jednotlivých subjektů. V tomto pohledu jsou panely představující jednotlivé subjekty. Hlavička každého panelu značí subjekt a tělo panelu obsahuje odkazy na nastavené sítě a spojení.

Další pohledy například jsou:

- Přehled dat, které lze použít v podmínce pro odesílání emailů,
- Přehled odběratelů, díky kterému lze kontrolovat přihlášené odběratele.

7.4.4 Rozesílání emailů

Rozesílání emailů probíhá při vytvoření požadavku na specifickou URL, čímž se spustí rozesílací skript. Tento skript je zabezpečený klíčem, který lze nastavit v konfiguračním souboru aplikace. Tento klíč je nutno poskytnout jako parametr požadavku při načítání stránky.



Obrázek 7.7: Příklad sestaveného emailu zobrazený emailovým klientem Mozilla Thunderbird

Při spuštění rozesílání jsou z databáze získány emaily k odeslání, tedy typy emailů spojené s odběry, pro které existují plány odeslání a podmínky těchto plánů jsou splněny. Z této množiny emailů jsou dále odebrány ty emaily, které již byly odeslány. Následně jsou tyto emaily zpracovány a odeslány tímto způsobem:

1. Pro aktuální email jsou připraveny proměnné, jako například odkaz pro zrušení odběru nebo definovaná data.

2. Tyto proměnné jsou použity pro sestavení obsahu emailu pomocí Twigu, pro jehož šablonu je použito tělo typu emailu.

3. Sestavený obsah emailu je, spolu s připravenými proměnnými, použit pro sestavení celého těla emailu, na což je opět použit Twig, nyní však se statickou šablonou *@mails/mailTemplate.twig*.

Na ukázce Obrázek 7.7 jsou znázorněny klíčové části emailu:

A Celistvá šablona emailu

B Sestavený obsah emailu

C Odkaz pro zrušení odběru

Služba starající se o samotné posílání emailů při vzniku získá z konfigurace počty určující limit emailů, které lze odeslat, a / nebo limit emailů, které lze odeslat na jednu doménu. Tyto limity omezují počty pro jedno spuštění rozesílání a při vykonávání služba počítá emaily odeslané, emaily přeskočené a emaily, jejichž odeslání se nezdařilo.

Na konci rozesílání emailů aplikace vrací odpověď ve formátu JSON obsahující tyto souhrnné počty.

Tento skript je vhodné spouštět například pomocí CRON úlohy, což je standardně linuxový skript starající se o pravidelné spouštění příkazů[10].

[10]https://kb.iu.edu/d/afiz

8 Testování

Všechny části aplikace jsou doprovázeny testovacími scénáři, které automaticky ověřují správnou funkčnost. Tyto testovací scénáře společně s programátorskou dokumentací popisují požadované funkčnosti modulů. Popis funkčnosti pomocí testů je tvořen z kategorií vstupů a z přesné definice očekávaných výstupů pro vstupy volené z kategorií. Tyto vstupy jsou často voleny jako hraniční případy, tedy vstupy, které jsou v nějakém smyslu extrémní.

8.1 Jasmine

Jasmine je JavaScriptový testovací framework založený na popisu chování. Nezávisí na žádném jiném frameworku a nepotřebuje DOM. Jasmine má čistou a jasně zřetelnou syntaxi, díky které lze jednoduše psát testy[9].

```
describe('String sorter', function() {
  it('sorts alphabetically in descending order', function() {
    var users = ['jack', 'igor', 'jeff'];
    var sorted = sortUsers(users);
    expect(sorted).toEqual(['jeff', 'jack', 'igor']);
  });
});
```

Kód 8.1: Příklad testu frameworku Jasmine

Příklad testu Jasmine, znázorněný Kódem 8.1, znázorňujě tři klíčové prvky testu představené funkcemi:

- **describe**: určuje jméno testovaného modulu
- **it**: určuje jméno konkrétní funkčnost, která je ověřována
- **expect**: přijímá výslednou hodnotu, která je porovnána zřetězenou funkcí **toEqual**

Při selhání testu je ve výpisu zobrazeno jméno neúspěšného testu a podrobný popis, jak se liší skutečná hodnota od očekávané. V uvedeném příkladu by toto jméno bylo *String sorter sorts alphabetically in descending order*.

Testy Jasmine je ale třeba spouštět a aplikace založené na frameworku AngularJS jsou běžně testovány pomocí spouštěče Karma, což je nástroj,

který dokáže například sledovat zdrojové soubory a automaticky spouštět testy. Pomocí Karmy lze také debugovat testy přímo v prohlížeči.

Kombinace Karma + Jasmine je běžně používaná pro testování Angularových aplikací, což je podloženo dokumentací [1] a ukázkovým projektem [2], ve kterém je připravena konfigurace aplikace a testů.

Při testování AngularJS aplikace jsou všechny HTTP požadavky zachyceny a odstíněny, což umožňuje snadnější testování asynchronních požadavků. Naopak se tímto znesnadňuje testování direktiv, které používají jako šablonu HTML, jelikož je šablona načtena pomocí HTTP požadavku, který v době testování není očekávaný.

Tento nedostatek je řešen načtením šablon před začátkem testování do šablonové cache pomocí pluginu *karma-ng-html2js-preprocessor* při spouštění Karmy.

Testování je spouštěno příkazem npm test, které spustí Karmu pomocí konfigurace v souboru `karma.conf.js` při každé změně sledovaného souboru jsou spuštěny všechny testy. Stručný výstup testů je znázorněný Kódem 8.2.

```
30 04 2017 23:04:49.989:WARN [karma]: No captured browser,
   open http://localhost:9876/
30 04 2017 23:04:50.009:INFO [karma]: Karma v1.4.1 server
   started at http://0.0.0.0:9876/
30 04 2017 23:04:50.009:INFO [launcher]: Launching browser
   Chrome with unlimited concurrency
30 04 2017 23:04:50.029:INFO [launcher]: Starting browser
   Chrome
30 04 2017 23:04:51.583:INFO [Chrome 57.0.2987 (Windows 10
   0.0.0)]: Connected on socket OtzwOs2C8tlq7j9LAAAA with id
   81442532
..........
Chrome 57.0.2987 (Windows 10 0.0.0): Executed 10 of 10
   SUCCESS (0.237 secs / 0.207 secs)
```

Kód 8.2: Výstup testovacího frameworku Jasmine

8.2 PHPUnit

PHPUnit testy jsou realizovány v balících nebo sadách testů. Tyto sady jsou v PHPUnit frameworku objekty dědící od třídy `TestCase`. Běžně se testy ukládají v adresářové struktuře odpovídající zdrojovým souborům [16], tedy

[1] https://docs.angularjs.org/guide/unit-testing

[2] https://github.com/angular/angular-seed

třída nacházející se v souboru src/Cash/Register.php by měla být pokryta sadou testů v souboru tests/Cash/RegisterTest.php.

Tato struktura je použita v případě frameworku KiwiBlade, který se nachází v adresáři `./libs/SeStep/kiwi-blade/src` a jeho testy v adresáři `./libs/SeStep/kiwi-blade/tests`. Adresářová struktura je také spjata se strukturou jmenných prostorů, které jsou využity pro načítání pomocných tříd některých testů.

Testy administrační části jsou v adresáři `./adminTests`, který strukturou odpovídá adresáři zdrojových souborů `./admin`.

```
class UserStorageTest extends TestCase
{

  public function testUserStorage() {
    $storage = new UserStorage();

    $this->assertEmpty($_SESSION, 'SESSION␣should␣be
       ␣empty␣before␣test');

    $storage->set("karel");
    $this->assertSame([UserStorage::class => 'karel'
       ], $_SESSION);

    $storage->clear();
    $this->assertEmpty($_SESSION, 'SESSION␣should␣be
       ␣empty␣after␣test');
  }
}
```

Kód 8.3: Ukázka testovacího scénáře práce s relací

Příklad jednotkového testu frameworku PHPUnit, který je znázorněn Kódem 8.3, ověřuje práci s proměnnou relace simulací procesu přihlášení a odhlášení uživatele.

Testování je z příkazové řádky spuštěno z kořenového adresáře příkazem znázorněným Kódem 8.4.

```
# windows
./vendor/bin/phpunit.bat --bootstrap ./adminTests/
   bootstrap.php <složka s testy>

#unix
./vendor/bin/phpunit --bootstrap ./adminTests/
   bootstrap.php <složka s testy>
```

Kód 8.4: Spouštění testů

Výsledek testů KiwiBlade frameworku je znázorněn Kódem 8.5, kde je k nalezení počet vykonaných testů a jejich úspěšnost v absolutním i relativním měřitku. Dále je ve výsledku vypsána časová a paměťová náročnost.

```
PHPUnit 5.7.13 by Sebastian Bergmann and
   contributors.

..............                                  14 / 14
   (100%)

Time: 917 ms, Memory: 4.00MB

OK (14 tests, 19 assertions)
```

Kód 8.5: Výsledek testů frameworku KiwiBlade

U testů je potom znázorněn znakem jeho výsledek, kde tečka znamená úspěšný a **F** neúspěšný test, **E** značí test, ve kterém nastala nečekaná chyba, a **S** značící test, který byl přeskočen z nějakého důvodu přeskočen. V případě chyby, selhání nebo přeskočení testu jsou za souhrny také vypsány detailní popisy jednotlivých chyb.

9 Závěr

Cílem práce bylo napojení aplikace MojeFavka na sociální sítě a vytvoření systému pro automatické odesílání emailových upozornění odběratelům.

V teoretické části práce byly představeny základní principy tvorby webových stránek a jejich provozu na webovém serveru Apache. V práci byly podrobněji představeny důležité frameworky, ze kterých vychází uživatelská část aplikace MojeFavka. Dále byly nalezeny sociální sítě využívané Fakultou Aplikovaných Věd a pod ní spadajícími katedrami a prozkoumány možnosti napojení těchto sítí do aplikace MojeFavka.

Pro potřeby práce byl rozšířen framework KiwiBlade, který vznikl v rámci studentovy semestrální práce na předmět KIV/WEB. Tento framework byl v praktické části náležitě představen v souvislosti s částmi API a administrace, které z něj vychází:

- Administrace, která slouží pro správu emailů a dalších aspektů aplikace,
- API, která umožňuje dynamičnost uživatelské části aplikace.

Jako kladnou stránku vzniku KiwiBlade považuji, že jsem měl možnost seznámit se s procesem návrhu a realizace generického základu pro PHP aplikace. Kladnou stránku také vidím v tom, že je tento základ použit pro více částí. Celkově ale vidím tvorbu nového frameworku v negativním světle, protože ve srovnání s déle existujícími frameworky, které jsou často open-sourcové, bude vlastní framework rozvíjen pomaleji a často bude řešit problematiky, které už jsou vyřešeny.

Z důvodu možnosti rozšiřování a snadnějšího testování byla změněna architektura aplikace MojeFavka v oblastech kompozice aplikační logiky a šablon. Následně byly do aplikace přidány prostředky pro její napojení na sociální sítě formou odkazů na relevantní stránky sociálních sítí a pluginů poskytovanými skrze vývojářské nástroje.

Dekompozici šablon a modulů považuji za významné pozitivum, protože tím byla značně zvýšena přehlednost kódu i přes malé organizační náklady. Tato změna může negativně ovlivnit výkon aplikace z důvodu navýšení počtu požadavků pro úplné načtení aplikace. Tento dopad ale může být odstraněn zmíněným procesem minifikace.

V rámci práce byly taky nakonfigurovány testovací prostředí PHPUnit, ve kterém byly vytvořeny testy ověřující funkčnost frameworku KiwiBlade a částí z něj vycházejících, a Jasmine, ve kterém byly vytvořeny testy uživatelské části vycházející z frameworku AngularJS. Kapitola Testování obsahuje podrobnosti ohledně konfigurace a spouštění testů v těchto prostředích.

Použité zkratky a pojmy

API	Application Programming Interface = Programové rozhraní aplikace
CGI	Common Gateway Interface = Způsob volání programů webovými servery
HTML	HyperText Markup Language = Jayzk popisu dokumentu
HTTP	HyperText Transfer Protokol = Protokol pro přenos dokumentů a dalších dat
JSON	JavaScript Object Notation = JavaScriptový popis objektu v textové podobě
MVC	Model View Controller = Návrhový vzor používaný při vývoji webových aplikací
npm	Node Package Manager = Balíkovací nástroj Node
SDK	Software Development Kit = Vývojářská sada
URL	Uniform Resource Locator = Jednotná adresa zdroje

- backend = Serverová část aplikace
- endpoint = Koncový bod API
- frontend = Klientská část aplikace

Literatura

[1] AngularJS. *AngularJS: Developer Guide: Dependency Injection* [online]. . [cit. 2017/03/05]. Dostupné z: `https://docs.angularjs.org/guide/di`.

[2] AngularJS. *AngularJS* [online]. . [cit. 2017/03/05]. Dostupné z: `https://angularjs.org/`.

[3] Apache. *RewriteRule Flags* [online]. [cit. 2017/03/05]. Dostupné z: `https://httpd.apache.org/docs/current/rewrite/flags.html`.

[4] Flanagan, D. *JavaScript: The Definitive Guide*. O'Reilly Media, 2011. ISBN 978-0-596-80552-4.

[5] Gervasio, A. *The Single Responsibility Principle* [online]. 2012. [cit. 2017/03/05]. Dostupné z: `https://www.sitepoint.com/the-single-responsibility-principle/`.

[6] Grand, M. *MIME Overview* [online]. [cit. 2017/03/05]. Dostupné z: `http://www.nada.kth.se/kurser/kth/2D1351/mime.pdf`.

[7] Green, B. – Seshadri, S. *AngularJS*. O'Reilly Media, 2013. ISBN 979-1-449-34485-6.

[8] hallettj. *How and why to avoid global variables in JavaScript* [online]. 2009. [cit. 2017/03/05]. Dostupné z: `https://gist.github.com/hallettj/64478`.

[9] Jasmine. *Jasmine: Behavior-Driven Javascript* [online]. [cit. 2016/03/09]. Dostupné z: `https://jasmine.github.io/`.

[10] Krasner, G. E. – Pope, S. T. A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System. *Journal of Object-Oriented Programming*. January 2000, 3, 3. ISSN 0896-8438. Dostupné z: `http://heaveneverywhere.com/stp/PostScript/mvc.pdf`.

[11] MailChimp. *Requirements and Best Practices for Lists* [online]. [cit. 2016/03/09]. Dostupné z: `http://kb.mailchimp.com/lists/growth/requirements-and-best-practices-for-lists`.

[12] Mason, B. *The Web Professional's Handbook*. Apress, 2003. ISBN 978-0-596-80552-4.

[13] Moore, G. E. *Moore's Law* [online]. [cit. 2016/03/09]. Dostupné z: `http://www.mooreslaw.org/`.

[14] Neti, S. SOCIAL MEDIA AND ITS ROLE IN MARKETING. *International Journal of Enterprise Computing and Business System*. July 2011, 1, 2. ISSN 2230-8849. Dostupné z: http://www.ijecbs.com/July2011/13.pdf.

[15] Origamiaddict. *HTTP* [online]. [cit. 2017/03/05]. Dostupné z: https://www.slideshare.net/origamiaddict/http-40249449.

[16] PHPUnit. *PHPUnit: Organizing Tests* [online]. [cit. 2016/03/09]. Dostupné z: https://phpunit.de/manual/current/en/organizing-tests.html.

[17] RequireJS. *RequireJS* [online]. [cit. 2017/03/05]. Dostupné z: http://requirejs.org/.

[18] Robbins, J. N. *HTML5 Pocket Reference: Quick, Comprehensive, Indispensable*. O'Reilly Media, 2013. ISBN 978-1-4493-6335-2.

[19] Vaughan-Nichols, S. J. Will HTML 5 Restandardize the Web? *Computer*. April 2010, 43, 4, s. 13–15. ISSN 0018-9162. doi: 10.1109/MC.2010.119. Dostupné z: http://ieeexplore.ieee.org/document/5445161/.

[20] Vrána, J. *1001 tipů a triků pro PHP*. Computer Press, 2011.

[21] W3. *The cascade* [online]. W3. [cit. 2017/03/05]. Dostupné z: https://www.w3.org/TR/CSS22/cascade.html.

[22] Wong, C. *Automating Tasks on the Web* [online]. 1997. [cit. 2017/03/05]. Dostupné z: http://www.oreilly.com/openbook/webclient/ch03.html.