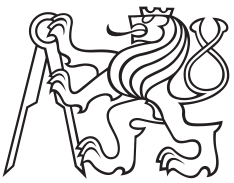
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Dokončení klientské části Integračního portálu

Jiří Blažek

Květen 2015
Vedoucí práce: Ing. Ondřej Macek, Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Jiří Blažek**

Studijní program: Otevřená informatika
Obor: Softwarové systémy

Název tématu: **Dokončení klientské části integračního portálu**

Pokyny pro vypracování:

Seznamte se s cíli projektu "Platforma pro sdílení souborů vědecko-výukové skupiny" [1] a s diplomovými pracemi P.Strnada [3] a K. Hašlarové [2]. Zaměřte se především na dokončení klientské části, která vznikla v rámci [2]. Následně proveďte přípravu integrace výstupů obou diplomových prací. Za úspěšnou přípravu na integraci se považuje buď úplná integrace proti backendu z [1] nebo úplná integrace proti simulaci rozhraní backendu. Výstupy své práce otestujte.

Seznam odborné literatury:

1. Kubr et al.: Platforma pro sdílení souborů vědecko-výukové skupiny, ČVUT FEL, 2014, technický report

2. K. Hašlarová: Front-end pro portál pro sdílení souborů, ČVUT FEL, 2015, diplomová práce

3. P. Strnad: Integrační server pro sdílení a zálohování dat, ČVUT FEL, 2015, diplomová práce

Vedoucí: Ing. Ondřej Macek, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Ing. Filip Železný, Ph.D.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 1. 4. 2015

Poděkování / Prohlášení

Tímto bych chtěl poděkovat svému vedoucímu práce Ing. Ondřejovi Mackovi, Ph.D. za odezvu poskytnutou během mojí práce a za jeho rady. Dále bych chtěl poděkovat celé svoji rodině za finanční podporu v době mého studia a mojí přítelkyni za psychickou podporu.

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

V Praze dne 22. května 2015

.....

Abstrakt / Abstract

Cílem této práce je implementace všech požadovaných funkcionalit a dokončení klientské části integračního portálu. Práce navazuje na předchozí vývoj, ve kterém bylo vytvořeno uživatelské rozhraní. Klientská část je implementována jako single-page application (SPA) za pomocí skriptovacího jazyka JavaScript a frameworku AngularJS. Testování je provedeno pomocí Protractor testovacího frameworku. Podařilo se mi v termínu implementovat a otestovat požadované části aplikace, včetně některých nových požadavků, které vyvstali během vývoje. Aplikace je částečně integrována s testovací serverovou částí, chybějící funkcionality jsou nasimulovány na straně klienta.

Klíčová slova: bakalářská práce; angularjs; klientská aplikace; webová aplikace; javascript; protractor; integrační portál; cesnet, testování

Primary goal of this work is to implement required basic features and finalization of client component for integration portal. The thesis follows the previous development in which user interface was created. Client application is implemented as a single-page application (SPA) by using JavaScript scripting language and framework AngularJS. Testing is done by using Protractor end-to-end test framework. I managed to implement and test all required functionalities of the application, including some of the new requirements that have emerged during development. Application is partially integrated with test server, missing functionalities are simulated on the client side.

Keywords: bachelor thesis; angularjs; client application; web application; javascript; protractor; integration portal; cesnet, testing

Title translation: Finalization of the Integration Portal's Client Component

Obsah /

Obrázky

Kapitola 1
Úvod

Projekt Integrační portál vznikl na základě již probíhajícího projektu CESNET [1] č. 493R1/2013 - Konsolidace zálohování, archivace a sdílení dat. Původní projekt se zaměřuje na způsob využití datových úložišť CESNET pro zálohování a archivaci dat. V rámci požadavku zálohování a archivace poskytuje CESNET možnost manuální nebo automatické archivace uložených dat na magnetické pásky.

Primárním cílem výzkumně-výukové skupiny je realizace samotného výzkumu a výuky. Její členové nemohou trávit čas studiem manuálů a dokumentace ke každému integrovanému softwaru zvlášť. Cílem integračního portálu je tedy poskytnout uživatelsky přívětivou možnost konfigurace služeb nebo získání návodů k provedení konfigurace cílových softwarů. Integrační portál bude odstiňovat uživatele od přímého použití služeb CESNET a zároveň bude portál integrován do infrastruktury zhotovitele (identity management, propojení s dalšími IS).[2]

Mezi požadavky na realizaci projektu patří sdílení dat mezi uživateli systému, rychlý přístup k datům přes uživatelsky přívětivé rozhraní, šifrování dat a centralizace služeb do jednoho systému. Spousta komerčních softwarových systémů některé tyto funkcionality nabízí, nicméně žádný z těchto softwarů nesplňuje všechny tyto požadavky. Proto vzniklo toto rozšíření projektu, jehož vyřešení pokryje všechny požadavky cílových uživatelů.

Projekt se nachází ve stavu vývoje. Dosud byla implementována část serverové části Integračního portálu v rámci diplomové práce Petra Strnada [3] a je vytvořeno grafické uživatelské rozhraní v rámci diplomové práce Kateřiny Hašlarové [4]. S probíhajícím vývojem vyvstávají další požadavky na funkčnost systému a vzhledem k jeho robustnosti, nebyl systém v těchto diplomových pracích dokončen.

1.1 Cíl práce

Tato práce navazuje na předchozí realizaci grafického uživatelského rozhraní, které bylo vypracováno Kateřinou Hašlarovou v rámci její diplomové práce [4] a realizace serverové části, která byla vypracována Petrem Strnadem v jeho diplomové práci [3].

Cílem této práce je dokončení požadovaných funkcionalit klientské části (front-end) integračního portálu a jejich testování. Výstup práce bude tedy funkční klientská část, která bude připravena pro integraci. Pro klientskou část bylo vytvořeno uživatelské rozhraní na které je nyní potřeba implementovat a poté navázat veškeré funkcionality, které systém obnáší a připravit klientskou část k integraci.

Nejprve musím opravit několik chyb a provést několik změn, které obsahuje předešlý návrh aplikace, tyto změny jsou popsány v kapitole 5. Dále budu pokračovat v předchozí práci a aplikaci budu implementovat jako single-page application (SPA) a použiji framework AngularJS [5]. Tento framework je velice populární a hojně používaný ve velkých projektech, mezi jeho hlavní výhody patří jeho aktivní vývoj, výborná dokumentace celého frameworku s doplněnými názornými příklady použití a online kurz pro

začátečníky *Shaping up with Angular* [6]. Vzhledem k tomu, že mám s tímto frameworkem zkušenosti z předchozích projektů a pro potřeby projektu integračního portálu bude velmi dostačující, nebudu tento výběr měnit.

Testování aplikace budu provádět pomocí Protractor [7] testovacího frameworku, který slouží pro koncové testování (end-to-end) AngularJS aplikací. Aplikace bude komunikovat s testovacím serverem, který poskytuje omezené množství funkcionalit, ale pro testovací účely se bude velice hodit. Další data nezbytná pro běh aplikace budou nasimulovaná přímo v aplikaci, nicméně aplikace bude připravena po malých změnách v kódu na nasazení.

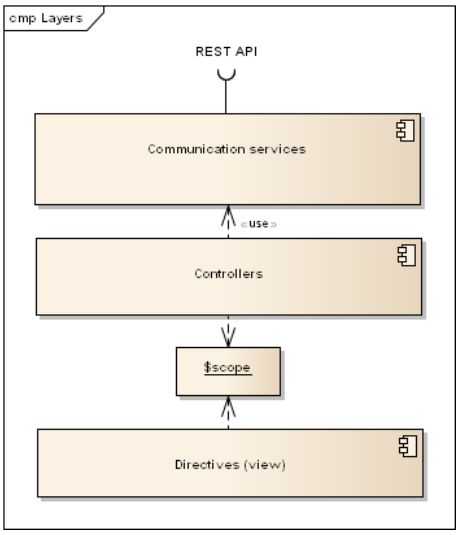
Kapitola 2
Současný stav klientské aplikace

Malá část funkcionalit již byla implementována v předchozím vývoji, nicméně komplexnost problémů z nově vzniklých požadavků na celý systém si vyžádala velké změny. Z předchozího vývoje byla tedy použita jen velmi malá část kódu řešícího funkcionality aplikace.

Při tvorbě uživatelského rozhraní byl použit framework SemanticUI [8], který slouží k tvorbě uživatelsky přívětivých grafických uživatelských rozhraní. Důraz byl také kladen na responzivní design aplikace, tzn., že zobrazení aplikace je optimalizováno pro velkou škálu zařízení. Jelikož v rámci předchozí práce proběhlo několik testů s potencionálními uživateli z cílové skupiny, design integračního portálu by měl odpovídat nejnovějším trendům webových aplikací a zároveň být uživatelsky přívětiví. Implementace funkcionalit byla provedena jen u některých částí aplikace a bude potřeba tyto nedostatky opravit a doplnit. Implementace aplikace integračního portálu je založena na frameworku AngularJS [5].

2.1 Architektura klientské části

Architektura Integračního portálu již byla vytvořena v předešlé práci Kateřiny Hašlarové [4], protože architektura celé aplikace je z velké části ovlivněna použitým frameworkem, v tomto případě AngularJS [5]. Na diagramu komponent Obr. 2.1 je znázorněna architektura klientské části celé aplikace. Každá skupina funkcí je rozdělena do komponent, které mezi sebou vzájemně komunikují a komponenty jsou dále rozděleny do vrstev. Tyto vrstvy oddělují komunikaci se serverem, aplikační logiku aplikace a prezenční logiku (view). Z diagramu je zřejmé, že rozvrstvení odpovídá vzoru MVC. Toto rozdělení vrstev je z důvodu udržitelnosti systému, který je pak mnohem jednodušší a flexibilnější. Je tak možné bez zásadních změn v kódu software dále rozšiřovat a modifikovat,

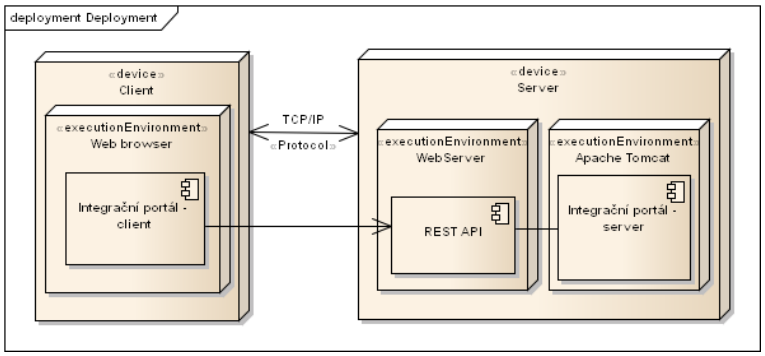


Obrázek 2.1. Diagram komponent front-end aplikace Integračního portálu

- **Communication services** jsou objekty, které zprostředkovávají komunikaci s REST API
- **Controllers** je komponenta obsahující kontroley pro manipulaci s modelem aplikaci
- **$scope** je objekt jehož prostřednictvím controllery ovládají view
- **Directives (view)** jsou HTML šablony, které slouží k zobrazení modelu aplikace

2.2 Diagram nasazení

Na diagramu nasazení Obr. 2.2 je popsána komunikace mezi hardwarovými prvky, na kterých budou spuštěny aplikace integračního portálu, tedy klientská a serverová část. Komunikace mezi klientem a serverem probíhá pomocí protokolu TCP/IP a vnitřní komunikace mezi klientskou a serverovou částí je řešena architekturou REST API. Serverová část v případě tohoto projektu bude fungovat jako zprostředkovatel, jelikož Integračním portálu bude integrovat další řadu služeb, vedle služeb **ČVUT** a CESNET [1] úložiště se tak do budoucna v projektu plánuje integrace služeb Dropbox [9], Google Drive [10] a další. Server tedy bude dále komunikovat pomocí API s dalším službami a jelikož se tato další komunikace netýká klientské části nebude v diagramu uvedena.



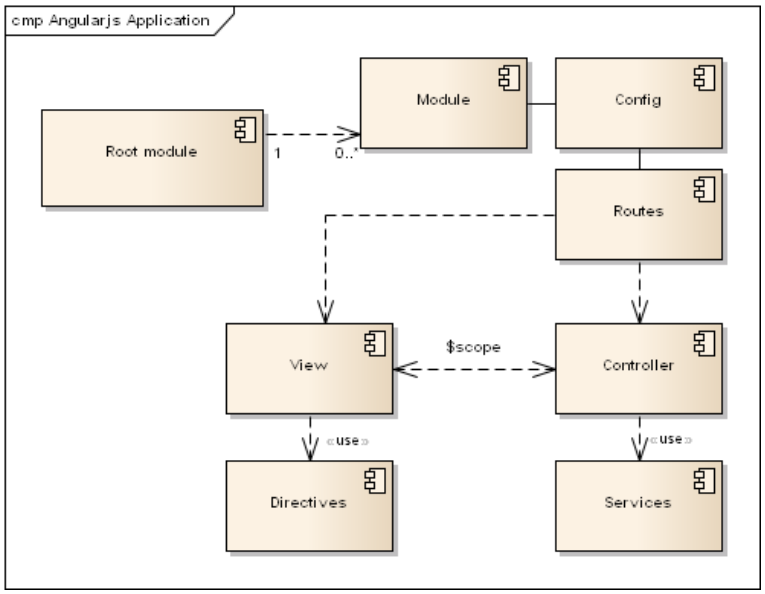
Obrázek 2.2. Diagram nasazení front-end aplikace integračního portálu

Kapitola 3
Framework AngularJS

AngularJS je open-source framework, který slouží pro vývoj webových aplikací a je zaštitován společností **Google**. Na jeho vývoji se podílí, jak tato společnost, tak komunita nezávislých programátorů. Hlavním cílem tohoto frameworku je zjednodušení a zpřehlednění vývoje SPA webových aplikací a jejich testování. AngularJS rozšiřuje HTML o řadu dalších elementů a atributů, které slouží k lepšímu oddělení uživatelského rozhraní a aplikační logiky.

3.1 Architektura AngularJS aplikace

V následujícím diagramu jsou zobrazeny všechny komponenty AngularJS [11] aplikace.



Obrázek 3.1. Komponenty AngularJS aplikace

3.2 Modules (Moduly)

Moduly AngularJS aplikací slouží k zaobalení jedné části aplikace do samostatného kontejneru. V podstatě se jedná o větší komponentu aplikace a pomocí modulů můžeme sjednotit veškeré funkcionality této komponenty na jedno místo. AngularJS obsahuje vždy hlavní (root) modul, do kterého se pak injektují další moduly.

3.3 Directives (Direktivy)

Direktivy umožňují rozšiřovat HTML o spoustu dalších atributů a elementů. AngularJS nabízí i možnost vytvoření vlastních direktiv, jak elementů, tak atributů. AngularJS

sám o sobě již obsahuje poměrně velké množství direktiv a jsou dobře dokumentované, takže jejich použití určitě usnadní práci, zpřehlední celou aplikaci a zjednoduší její testování.

V AngularJS aplikacích jsou direktivy jediné místo, kde by se mělo přistupovat k DOM modelu. Hlavním důvodem je testování aplikace, které by bylo z tohoto důvodu velice komplikované a dalším důvodem je přehlednost psaného kódu.

3.3.1 Two-way data binding (Obousměrné provázání dat)

Ve většině klasických šablonovacích systémů je provázání dat mezi aplikačním modelem a uživatelským rozhraním (view) pouze jednosměrné. V AngularJS šablonovacím systému je toto provázání dat řešeno obousměrně, tzn. změny provedené v modelu se ihned projeví na uživatelském rozhraní a naopak, když uživatel interaguje s uživatelským rozhraním, změny se ihned projeví i v modelu. Model je tedy tzv. single-source-of-truth, což zjednodušeně znamená, že view je projekcí samotného modelu. Z Two-way data binding plyne výhoda pro testování controllerů, jelikož se takto kompletně oddělí logika controlleru a samotného view.

3.4 Controllers (Kontrolery)

Kontrolery slouží v AngularJS aplikaci k poskytnutí proměnných a funkcí pro direktivy a výrazy použité ve view. Kontroler je přiřazen k DOM modelu pomocí direktivy a je mu přiřazen operační prostor ($scope). V tomto prostoru může programátor pomocí kontroleru pracovat s proměnnými a funkcemi, které budou poskytnuty direktivám nacházejícím se ve stejném operačním prostoru ($scope).

Kontrolery se používají k:

- inicializaci $scope objektu
- přidávání funkcionalit nebo chování $scope objektu

Kontrolery by se neměli používat k:

- manipulaci s DOM modelem
- uchovávání prezentační logiky (vzhledem k testovatelnosti)
- sdílení kódu nebo stavů mezi ostatními kontrolery
- spravování životních cyklů ostatních komponent

3.5 Services (Služby)

Služby slouží k organizaci a sdílení kódu napříč celou AngularJS aplikací. Jedná se o singleton objekty, které poskytují funkcionality ostatním částem aplikace. Objekt služby se vytváří pouze pokud na něm závisí ostatní komponenty aplikace. AngularJS obsahuje několik předdefinovaných služeb např.: $http sloužící ke komunikaci se vzdálenými HTTP servery pomocí XMLHttpRequest objektů nebo pomocí JSONP.

3.6 Další použité moduly

Během vývoje vyvstalo několik problémů, které si vyžádaly použití dalších modulů, jelikož jejich implementace by byla časově náročná a zpomalilo by to tedy vývoj celého projektu.

Další použité moduly:

- AngularUI Router v0.2.13[1])
- MacGyver v0.6.1[2])
- FileSaver.js[3])

AngularUI Router [12] je modul pro realizaci routování (přesměrování). Tento modul byl již použit v předešlé práci a stará se o routování dynamického obsahu stránek podle URL adres. Každá URL adresa je přesměrovávána na příslušnou direktivu.

Pro potřeby implementace některých klíčových funkcionalit aplikace jsem vybral knihovnu **MacGyver** [13], která je založena na frameworku AngularJS. Jedná se tedy o AngularJS modul, který obsahuje velkou škálu funkcionalit, direktiv a služeb. Direktivy obsahují svůj vlastní CSS soubor pro stylizaci MacGyver elementů, nicméně tato skutečnost nijak nenarušuje přívětivost nebo kvalitu již vytvořeného uživatelského rozhraní. Hlavními výhodami tohoto modulu je jeho stále probíhající vývoj a dobrá dokumentace API.

Dalším použitým modulem je **FileSaver.js** [14], který implementuje funkci *saveAs* sloužící k uložení generovaného souboru do počítače klienta. Tento modul umožňuje uložení široké škály typů souborů. Implementace FileSaver.js poskytuje tuto funkci i pro prohlížeče, které ji nativně nepodporují.

[1]) `https://github.com/angular-ui/ui-router`
[2]) `https://github.com/angular-macgyver/MacGyver`
[3]) `https://github.com/eligrey/FileSaver.js`

Kapitola 4
Analýza

Tato kapitola obsahuje detailní popis požadavků na systém, které je potřeba implementovat v rámci této práce. Každá základní funkcionalita je popsána ve vlastní podsekci. Některé z funkčních požadavků jsou již zmíněni v rámci diplomové práce Kateřiny Hašlarové [4], nicméně během vývoje došlo k několika drobným změnám. Tyto změny jsou popsány v příslušné kapitole 5.

4.1 Základní funkcionality

Popis základních funkcionalit:

- **Uživatelské skupiny** – skupiny kontaktů uživatele, detailně viz 7.3.3
- **Štítky** – štítky pro označení souborů/složek, detailně viz 7.3.4
- **Organizační jednotky** – detailně viz 7.3.1
- **Uživatelské účty externistů** – detailně viz 7.3.2
- **Správce souborů** – souborový manager, detailně viz 7.2

4.2 Funkční požadavky

Funkční požadavky představují veškeré požadavky na funkcionalitu klientské aplikace Integračního portálu. Pro pokrytí všech požadavků bylo potřeba implementovat případy užití pro každý požadavek. Seznam případů užití pro funkční požadavky dané skupiny požadavků je vždy v příslušné sekci v kapitole 7.

1. **Uživatelské skupiny** – systém umožní uživateli
   - vytvářet skupiny
   - editovat skupiny
   - smazat skupiny
   - přidat další uživatele jako členy do skupiny
   - smazat členy skupiny
2. **Štítky** – systém umožní uživateli
   - vytvářet vlastní štítky
   - editovat štítky
   - smazat štítky
   - přidávat štítky k jednotlivým souborům/složkám
   - odebrat štítek z jednollivých souborů/složek
   - vybrat jeden nebo více štítků, podle kterých se bude filtrovat obsah na příslušném úložišti
3. **Organizační jednotky** – systém umožní správci organizačních jednotek
   - vytvářet nové organizační jednotky

- smazat organizační jednotky
- editovat jednotlivé atributy organizačních jednotek

4. **Uživatelské účty externistů** – systém umožní správci externistů

- vytvářet uživatelské účty externistů
- editovat uživatelské účty externistů
- smazat uživatelské účty externistů

5. **Správce souborů** – systém umožní uživateli

- vytvořit novou složku
- smazat libovolnou složku a celý její obsah
- editovat složky (pokud je složka ve stavu *offline*, má uživatel pouze možnost nastavit ji do stavu *online*).
- přesouvat složky/soubory do jiných složek
- prohlížet veškerý obsah složek v kořenovém adresáři
- nahrát soubor
- smazat soubor
- sdílet složky/soubory s ostatními uživateli a skupinami
- filtrování obsahu adresáře podle aktivních štítků
- filtrování obsahu adresáře pomocí pole *Filter...*
- připnout složku do oblíbených (rychlých odkazů)

4.3 Obecné požadavky

Obecné požadavky jsou požadavky, které kladou omezení na design a provedení (např.: požadavky na výkonnost, standardy kvality, nebo designové omezení) [15]. Většina obecných požadavků na klientskou aplikaci již byla specifikována v diplomové práci Kateřiny Hašlarové [4], proto uvedu jen obecné požadavky, které se přímo týkají mé práce.

- Systém bude realizován jako webová aplikace.
- Přihlášení bude implementováno přímo v aplikaci, tedy uživatel nebude přesměrován na jinou stránku.
- Systém bude testován proti testovacímu serveru, ostatní funkcionality budou pracovat se simulovanými daty.

Kapitola 5
Změny v původní implemantaci

V této kapitole jsou popsány změny, které bylo potřeba provést v původním návrhu projektu během implementace nových částí aplikace.

5.1 Použité frameworky

V aplikaci integračního portálu byl v předešlé práci použit framework SemanticUI [8] pro tvorbu grafického uživatelského rozhraní a AngularJS framework pro tvorbu samotné aplikace. Jelikož jsou oba frameworky pořád ve vývoji, bylo potřeba zaktualizovat jejich knihovny na poslední vydanou stabilní verzi z důvodu aktuálnosti aplikace. Nové verze frameworků obsahují nové funkcionality, opravy chyb apod.

5.1.1 AngularJS

U frameworku AngularJS [5] byla použita již zastaralá verze souborů pro tento framework. V nové práci jsem tedy použil aktuální a stabilní verzi 1.3.15[1]).

5.2 Problémy z předchozího vývoje

Mezi hlavní problémy patřily chyby ve frameworku SemanticUI, které ovlivňovaly funkčnost celé aplikace. Jednalo se především o chyby v implementaci modálních dialogů, které jsem v aplikaci použil velice často vzhledem k jejich přehlednosti a zřejmému sdělení uživateli, co od něj aplikace právě očekává. Tento problém v implementaci jsem vyřešil použitím modulu MacGyver, který obsahuje direktivy pro tvorbu modálních dialogových oken.

Velkým nedostatkem předchozí práce byla absence veškeré dokumentace zdrojového kódu a testů funkčnosti aplikace. Vzhledem k tomu, že serverová strana byla dokončena jen z části a nebyla tak možná úplná integrace, tak ostatní data měla být simulovaná pomocí mock objektů. Nicméně většina simulovaných dat byla napsaná přímo do kódu. Tato skutečnost téměř znemožnila testování v rané části vývoje a přinesla mnoho komplikací v moji implementaci. Některé z funkcionalit naimplentované v předchozí práci byly na těchto datech závislé a po simulaci dat správným způsobem se tyto implementované části staly nefunkční a nepoužitelné.

Kód jsem během vývoje pečlivě dokumentoval a v závěru mé práce také otestoval. Nedostatky týkající se simulovaných dat jsem odstranil vytvořením správné simulace. Data tedy nejsou „zadrátovaná“ přímo do zdrojového kódu, ale jsou simulovaná mimo aplikaci, kde se pro ně aplikace dotazuje téměř totožným způsobem, jako kdyby se dotazovala serverové strany.

[1]) `https://github.com/angular/angular.js`

5.3 Refactoring

Při předchozím vývoji nebyl kladen žádný důraz na čitelnost zdrojového kódu aplikace. Čitelnost, přehlednost a čistota kódu je důležitým aspektem při vývoji robustních softwarových systémů a tak byl tento krok nezbytný kvůli urychlení procesu vývoje, zmenšení rizika chyb, testovatelnosti a čitelnosti kódu vzhledem k jeho rozšiřitelnosti.

Z tohoto důvodu jsem úplně změnil celou strukturu projektu. Pro porovnání jsou viditelné změny na fragmentech 5.3.1 a 5.3.2, kde je zřejmá změna celé struktury kořenového adresáře projektu a seskupení zdrojových kódu podle funkcionality.

Další částí refactoringu byla oprava a přejmenování některých názvů proměnných, funkcí a objektů, které neodpovídali programátorským konvencím. Veškeré provedené změny jsou dokumentovány v repositáři projektu CESNET-Integracni-portal/integracni-portal-ui[1]) na GitHubu.

```
---
integracni-portal-ui/
    css/
        portal.css
        semantic.css
    js/
        angular.js
        angular-ui-router.js
        app.js
        data.js
        ...
    partials/
        subpartials/
            add.html
        archived.html
        archived_detail.html
        archived_show.html
        folder.html
        set-change-pass.html
        set-groups.html
        ...
    index.html
    login.html
---
```

Fragment 5.3.1. Struktura projektu z předchozího vývoje

Strukturu celého projektu jsem upravil tak, aby další programátoři, co budou spolupracovat na tomto projektu měli ihned přehled o tom, co se děje v které části aplikace. Strukturu projektu jsem vytvořil podle článku vydaného na blogu AngularJS *Best Practice Recommendations for Angular App Structure* [16]. Každá komponenta aplikace má svůj modul, který obsahuje pouze funkcionality této komponenty, zjednodušší se tak testovaní komponent a zvýší přehlednost kódu.

[1]) `https://github.com/CESNET-Integracni-portal/integracni-portal-ui`

```
---
integracni-portal-ui/
    css/
         macgyver.css
        portal.css
        semantic.css
    js/
        externists/
            externists.module.js
        groups/
             groups.module.js
        home/
        ...
        app.controllers.js
        app.module.js
        services.module.js
        ...
    /libs
        angular.min.js
        angular-ui-router.min.js
        FileSaver.js
        ...
    partials/
        externists/
            externist_modal.html
            set-externists.html
        groups/
         group_members_list.html
         group_modal.html
         set-groups.html
        home/
        file-folder-modals/
            file_edit_modal.html
            file_upload_modal.html
        ...
    tests/
        e2e/
            admin/
                admin.po.js
                admin.spec.js
            home/
            settings/
            conf.js
    index.html
---
```

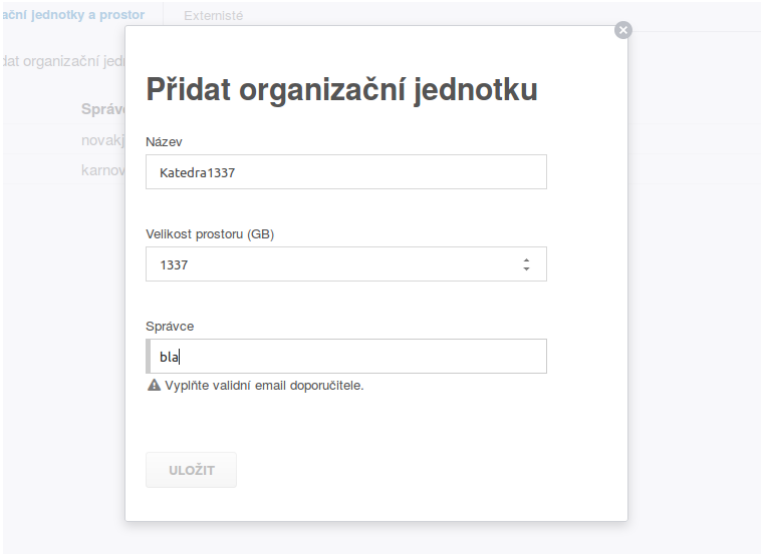
Fragment 5.3.2. Aktuální struktura projektu

Kapitola 6
Úpravy GUI

V této kapitole jsou popsané změny, které bylo nezbytné provést k zlepšení funkce celého systému a odstranění chyb v předešlém návrhu. Každá sekce této kapitoly obsahuje popis každé změny v systému. Při implementaci některých částí aplikace se ukázalo výhodnější a uživatelsky přívětivější provést malé změny v uživatelském rozhraní. Některé funkcionality si tyto změny dokonce vyžádali, jelikož se při návrhu uživatelského rozhraní nepočítalo s komplikacemi, které nastaly až při samotné implementaci.

6.1 Sjednocení dialogových oken

Dialogová okna byla v předchozí práci použita pouze pro vytvoření složky a nahrání souboru. Veškeré další formuláře pro tvorbu skupin, organizačních jednotek apod. byly přímo na příslušné stránce společně se seznamem skupin nebo seznamem organizačních jednotek. V praxi by to znamenalo, že uživatel má zobrazený formulář i přesto, že ho nechce použít a chce pouze pracovat se samotnými skupinami nebo organizačními jednotkami. Veškeré formuláře, které aplikace obsahuje jsem tedy sjednotil a pro každý formulář udělal příslušné modální dialogové okno. Na Obr. 6.1 je ukázka modálního dialogu pro vytvoření organizační jednotky.



Obrázek 6.1. Modální dialog pro tvorbu organizační jednotky

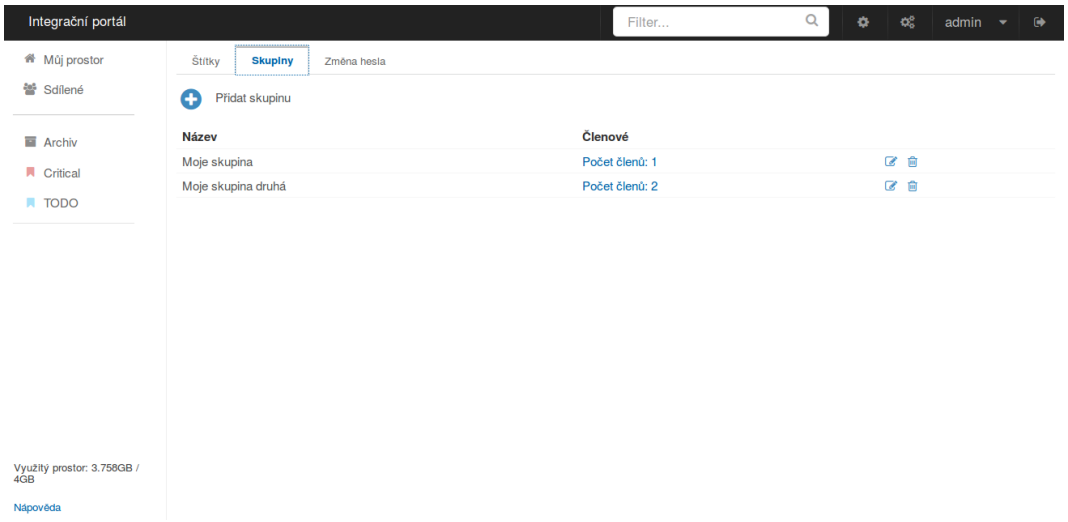
V předešlé práci byl použit framework SemanticUI pro tvorbu a úpravu modálních dialogových oken. Jak se ukázalo v navazující implementaci, tento framework obsahoval několik implementačních chyb, které se týkaly výhradně modálů a znemožňovali tak práci s nimi (např.: po zavření modálního okna zůstal aktivní dimmer). Proto jsem na práci s modály v implementaci použil AngularJS modul MacGyver.

6.2 Členové skupiny

V předchozí práci byl počet členů zobrazen již v seznamu skupin, tento způsob nebyl dobrý, protože pokud by měla skupina více členů, zabrala by tato informace celou

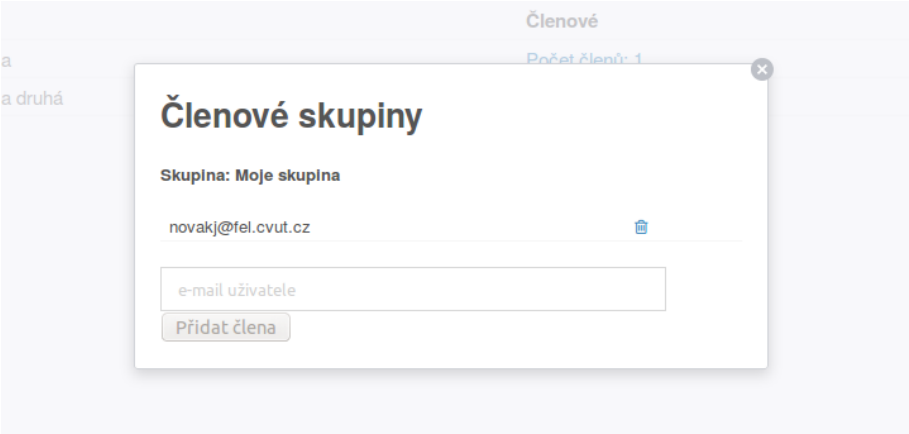
stránku. Nyní je přidávání členů a zobrazení členů provedeno pomocí modálního dialogového okna. Při zobrazení všech skupin se uživatel dozví počet členů každé skupiny.

Na Obr. 6.2 je vidět rozložení stránky v případě kliknutí na záložku *Skupiny*. Zde přihlášený uživatel vidí pouze jím vytvořené skupiny a může s nimi interagovat.



Obrázek 6.2. Seznam skupin uživatele

Po kliknutí na odkaz ve sloupci členové v příslušném řádku skupiny, se uživateli zobrazí seznam všech členů viz Obr. 6.3 a zároveň se mu otevře i možnost členy přidávat a odebírat.



Obrázek 6.3. Seznam členů skupiny

6.3 Rozdělení nastavení

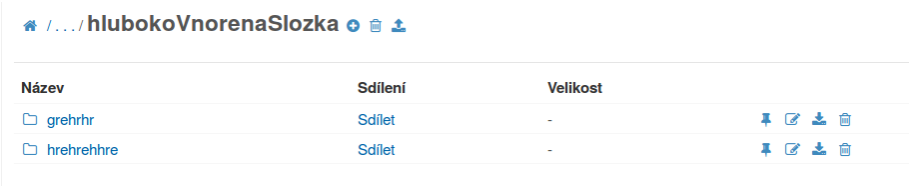
Každý uživatel má přístup k osobnímu nastavení. Osobní nastavení obsahuje správu štítků, správu uživatelských skupin a pro uživatelské účty externích uživatelů i změnu hesla. Další nastavení je pouze pro administrátory systému. Administrátorské nastavení obsahuje správu organizačních jednotek a správu uživatelských účtu externistů.

V původním návrhu byla tato nastavení seskupena pod jedním tlačítkem. Pro přehlednost jsem tato nastavení rozdělil a každé dal pod samostatné tlačítko. Toto řešení je uživatelsky přívětivější a přehlednější.

6.3.1 Sjednocení ikon

V částech aplikace s podobnou funkcionalitou nebylo sjednocené pořadí ikon pro editaci, tvorbu, mazání a přidání do oblíbených, tyto nedostatky bylo potřeba opravit, aby měl uživatel lepší přehled o aplikaci a dokázal si lépe osvojit uživatelské rozhraní. Sjednotil jsem tedy ikony pro práci se složkami a soubory, aby byly ve stejném pořadí v každé části aplikace.

Dalším problémem byly např.: breadcrumbs, což jsou stopy navigace, které poskytují uživateli informaci o aktuální lokaci ve složkách. Bylo potřeba opravit situace, kdy bylo několik vnořených podsložek a došlo tak k přetečení breadcrumbs přes okraj a následně k deformaci uživatelského rozhraní. Tuto chybu jsem opravil kontrolou délky breadcrumbs viz Obr. 6.4, které se při konkrétní délce nahradí tečkami, které odkazují na předchozí složku a zároveň je viditelný název poslední složky, uživatel tak neztratí přehled o lokaci ve správci souborů a navíc nedojde k nechtěným deformacím uživatelského rozhraní.



Obrázek 6.4. Breadcrumbs u vnořených složek

6.3.2 REST API v0.2

REST neboli Representational State Transfer je architektonický styl pro komunikaci. Tento styl komunikace je orientován přímo na data nikoliv na volání procedur. RESTful aplikační rozhraní je takové rozhraní, které dodržuje všechny zásady a principy, které definuje REST.[17]

Během implementace mojí práce došlo ke změnám v REST API. Konkrétně bylo vytvořeno REST API v0.2, kde proběhlo několik změn, týkajících se dotazů na server. Hlavními změnami bylo rozdělení jednotlivých dotazů na editaci dat. Toto rozdělení mělo za následek vznik několika dalších dotazů, pomocí kterých se bude na serveru zjišťovat k jakým konkrétním změnám v datech došlo. Změna API souvisí i se změnou architektonického vzoru serveru. Veškeré informace a dokumentace k API v0.2 jsou k dispozici na Apiary CESNET Integračního portálu[1]).

[1]) `http://docs.cesnetintegracniportal.apiary.io/`

Kapitola 7
Implementace nových částí

V této kapitole popíši hlavní části implementace, které jsem provedl v rámci dokončování klientské části integračního portálu. V průběhu vývoje projektu vyvstala řada problémů kvůli stále probíhajícímu vývoji serverové strany a neustálé pokládání nových požadavků na celý Integrační portál. Veškeré zdrojové kódy ke klientské části aplikace jsou k dispozici ve veřejně přístupném repositáři CESNET-Integracni-portal/integracni-portal-ui[1]) na GitHubu.

7.1 Přihlašování

V této sekci je popsán proces přihlášení a řešení autentizace přihlášených uživatelů. Dále jsou zde popsány změny týkající se přihlašovacího dialogu. Vysvětlení procesu zobrazení stránky můžete najít na Obr. 7.1.

7.1.1 Autentizace uživatele

Autentizace uživatele je řešena pomocí **OAuth 2.0** na bázi tokenů. Uživatel při přihlášení obdrží tzv. access token, který je uložen pomocí cookies ve webovém prohlížeči, ve kterém je aplikace spuštěna. Tento token posílá webový prohlížeč s každým požadavkem na server a server tak pozná, jestli má uživatel oprávnění zobrazit požadovaný obsah.[18]

7.1.2 Přihlášení pomocí FELid

Jedním z požadavků integračního portálu je přihlašování pomocí **FELid** [19]. Jedná se o požadavek pro budoucí implementaci, jelikož tato funkcionalita není vyřešena na serverové straně, nicméně implementace klientské části je na tento případ užití již připravena a bude potřeba jen málo změn k zprovoznění její funkčnosti.

Jedná se o globální autentizační systém pro studenty a pracovníky **ČVUT** fakulty elektrotechnické. Uživatel zadává přihlašovací údaje pouze na centrální přihlašovací stránce FELid. Zde uživatel zadá svoje přihlašovací údaje a po úspěšné autentizaci je přesměrován zpět do aplikace, která uživatele označí jako přihlášeného. Aplikace, která vyžaduje přihlášení tak dostane pouze informaci o úspěšném přihlášení a vůbec se nedostane k samotným přihlašovacím údajům uživatele. Toto přihlášení je tak nejen bezpečné, ale pro uživatele i pohodlné, jelikož se nemusí do různých aplikací vyžadujících FELid přihlašovat zvlášť.[19]

7.1.3 Přihlašovací dialog

Dialog pro přihlášení byl v předešlé práci řešen pomocí přesměrování na jinou stránku. Na této stránce se nacházelo tlačítko k přihlášení už přednastaveného uživatele. Pro potřeby testování byl tento způsob přihlášení dostačující, ale pro reálné nasazení systému bylo potřeba vytvořit přihlašovací formulář pro přihlášení libovolného uživatele.

[1]) `https://github.com/CESNET-Integracni-portal/integracni-portal-ui`

Přesměrování přihlášení na jinou stránku se neztotožňuje s architektou SPA, proto bylo potřeba přihlášení implementovat uvnitř aplikace. Pro tyto potřeby jsem vytvořil přihlašovací modální dialog s formulářem, který je zobrazen při vyžádání si jakékoliv části aplikace, která vyžaduje přihlášení.[20]

Ověření toho, jestli část aplikace nebo stránka vyžaduje přihlášení je řešeno na klientské straně. Pro každou URL adresu neboli stav aplikace je nastaven parametr *requireLogin*, který obsahuje informaci o tom, zda je vyžádováno přihlášení k zobrazení obsahu na této adrese viz fragment 7.1.3.

```
        $stateProvider
                .state("index", {
                    url: "/",
                    templateUrl: "./partials/home/home.html",
                    controller: 'indexCtrl',
                    data: {
                        requireLogin: true
                    }
                })
```

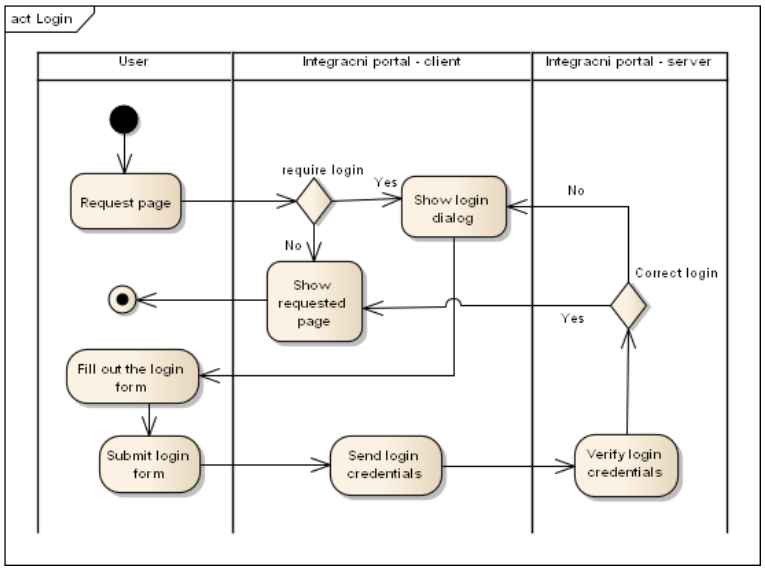
Fragment 7.1.3. Nastavení pro stav aplikace

Kontrola toho, jestli je potřeba přihlášení uživatele ke zobrazení obsahu stránky je prováděna v hlavním modulu viz fragment 7.1.4 aplikace pomocí služby *oauthService*, která obsahuje všechny funkce potřebné k přihlášení a autentizaci aplikace. Realizaci této funkcionality jsem provedl pomocí AngularJS funkce *$on('$stateChangeStart', function(...))*, která se zavolá při každé změně stavu aplikace viz 7.1.4.

```
$rootScope.$on('$stateChangeStart', function(event, toState, toParams) {
                    var requireLogin = toState.data.requireLogin;
                    // $rootScope.currentUser -> currently logged
                    if(requireLogin && $rootScope.currentUser === null){
                        event.preventDefault();
                        that.login();
                    }
                });
```

Fragment 7.1.4. Kontrola potřeby přihlášení při změně stavu aplikace

Na diagramu viz Obr. 7.1 je znázorněn proces, kterým musí celý systém projít, pokud chce uživatel zobrazit nějaký obsah aplikace.



Obrázek 7.1. Diagram aktivit zobrazení stránky

7.2 Správce souborů

V této sekci popíši řešení dílčích části aplikace, které se týkají správce souborů, tedy práce se složkami/soubory, jejich sdílení, přidávání do oblíbených atd.

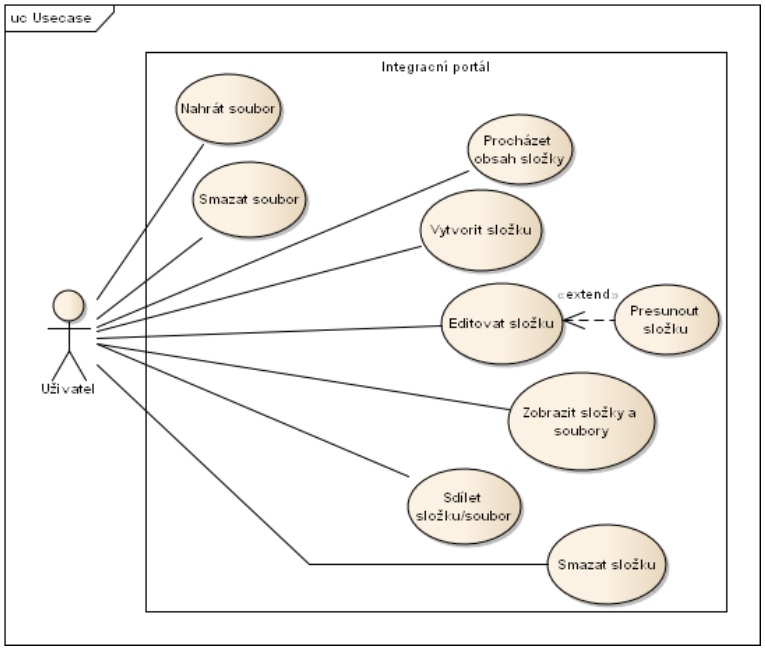
7.2.1 Prostor pro práci s vlastními daty

Každý uživatel má standardně nastavenou velikost svého přiřazeného prostoru, se kterým může libovolně pracovat. K tomuto prostoru má uživatel přístup pomocí tlačítka *Můj prostor*. Uživatelské akce týkající se složek nebo souborů v části *Můj prostor* jsou znázorněné na diagramu užití viz Obr. 7.2. V kořenovém adresáři svého prostoru může uživatel libovolně vytvářet, mazat, editovat, stahovat, nahrávat složky/soubory a sdílet je s ostatními uživateli nebo celými skupinami.

Případy užití implementované v kontextu práce se složkami/soubory v *Můj prostor*:

- **Zobrazit všechny složky/soubory** – umožní uživateli zobrazit seznam všech složek a souborů
- **Vytvořit složku** – umožní uživateli vytvořit novou složku
  - zadat název složky
  - označit stav složky (online/offline)
  - přiřadit štítky ke složce
- **Smazat složku** – umožní uživateli smazat libovolnou složku a celý její obsah
- **Editovat složku** – umožní uživateli editovat složky (pokud je složka ve stavu *offline*, má uživatel pouze možnost nastavit ji do stavu *online*).
- **Přesunout složku/soubor** – umožní uživateli přesouvat složky/soubory do jiných složek
- **Procházet obsah složky** – umožní uživateli prohlížet obsah složek v kořenovém adresáři
- **Nahrát soubor** – umožní uživateli nahrát soubor
- **Smazat soubor** – umožní uživateli smazat soubor
- **Sdílet složku/soubor** – umožní uživateli sdílet složky/soubory s ostatními uživateli a skupinami

- **Filtrovat obsah pomocí štítků** – umožní uživateli filtrování obsahu adresáře podle aktivních štítků
- **Filtrovat obsah adresáře** – umožní uživateli filtrování obsahu adresáře pomocí pole *Filter...*
- **Přidat/odebrat složku do/z oblíbených** – umožní uživateli připnout/odepnout složku do/z oblíbených (rychlých odkazů)



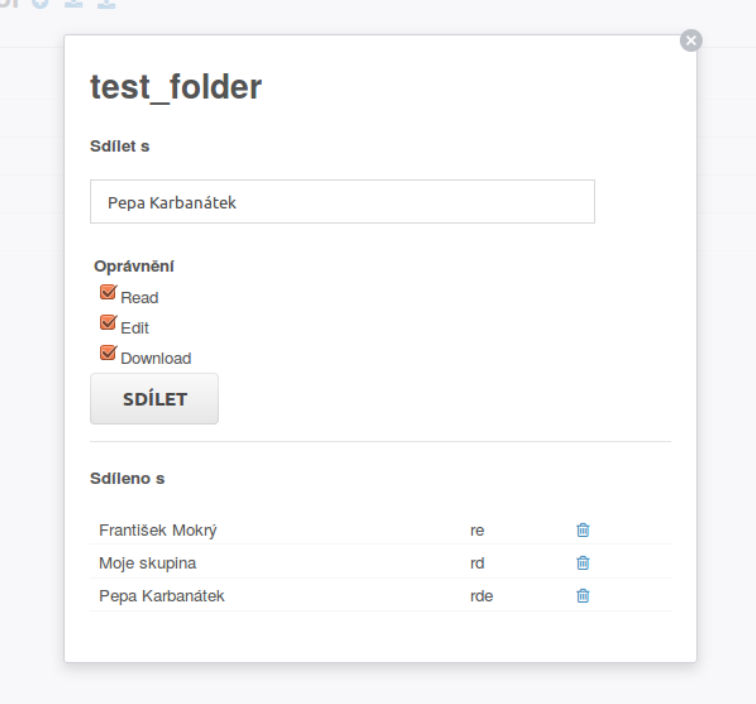
Obrázek 7.2. Diagram případů užití pro správce souborů

Implementací těchto případů užití jsem pokryl veškeré funkční požadavky zadané na správce souborů.

7.2.2 Sdílení

Vlastník souboru/složky má možnost tyto sdílet s ostatními uživateli používající aplikaci integračního portálu. Uživatel má možnost sdílet složku/soubor s jednotlivými uživateli nebo s celými skupinami uživatelů, které si sám vytvořil. Dále mu aplikace také umožní přiřadit jednotlivá oprávnění k práci se souborem/složkou pro uživatele nebo skupinu, se kterými tato data sdílí.

Pro sdílení složky/souboru musí uživatel přejít k modálnímu dialogu prostřednictvím tlačítka *Sdílet* v příslušném řádku dané složky nebo souboru. Pokud již uživatel tuto složku/soubor sdílí bude tlačítko *Sdílet* nahrazeno tlačítkem *Sdíleno s: [číslo]*, kde číslo znamená počet uživatelů nebo skupin, se kterými uživatel data již sdílí viz 7.3.



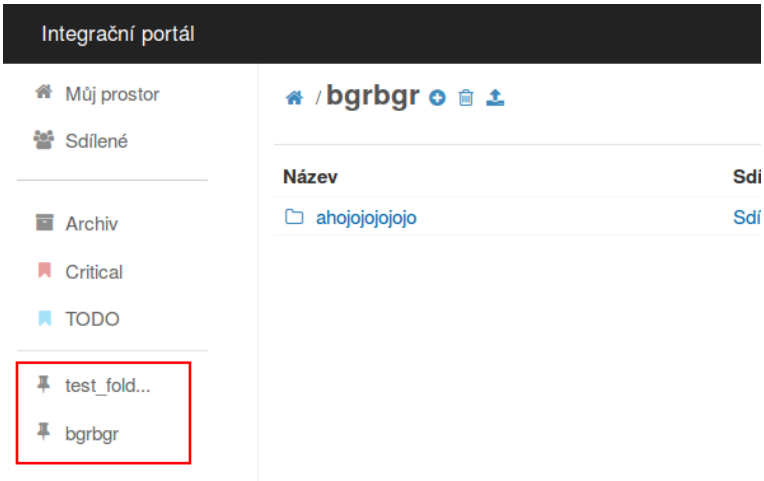
Obrázek 7.3. Modální dialogové okno pro sdílení

Oprávnění, které může uživatel přidělit uživatelům nebo skupinám uživatelů:

- **Read** – uživatel může libovolně procházet obsah složky
- **Edit** – uživatel může složku editovat, měnit její název, přesunout do jiné složky
- **Download** – uživatel má oprávnění ke stažení složky/souboru

7.2.3 Oblíbené

Oblíbené složky fungují na principu připínání rychlých odkazů na oblíbené složky, pomocí kterých se uživatel může snáze orientovat v jeho prostoru. Uživatel si tak může často používanou složku připnout do levého menu a nemusí ji tak vždy hledat. Připnuté složky jsou označeny červeným rámečkem viz Obr. 7.4. Pro přidání složky do oblíbených musí uživatel kliknout na ikonu připínáčku v příslušném řádku dané složky/souboru. Po kliknutí na odkaz v tomto menu bude uživatel v souborovém manageru přesunut do lokace složky příslušné tomuto odkazu.

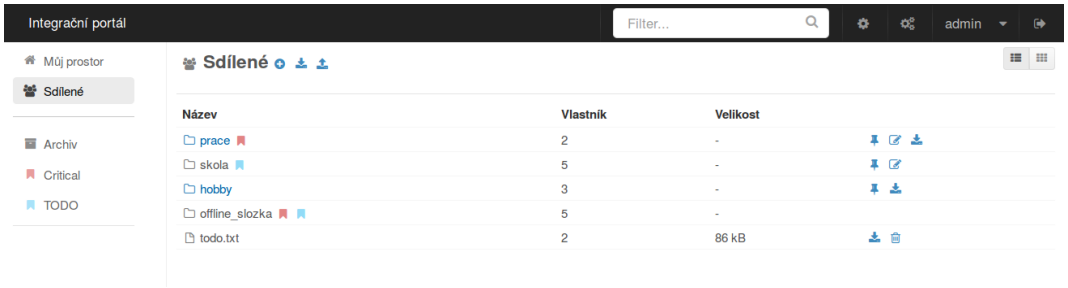


Obrázek 7.4. Ukázka oblíbených složek

7.2.4 Prostor sdílených složek/souborů

Prostor *Sdílené* viz Obr. 7.5 je místo v aplikaci, kde může uživatel pracovat se složkami a soubory, které s ním sdílejí další uživatelé. Akce, které může uživatel se složkami/soubory provádět jsou dány oprávněními, které nastavil uživatel, jenž složku/soubor sdílel. Oprávnění nastavené pro složku se rekurzivně vztahuje i na veškerý obsah této složky. Veškeré funkce pro práci se složkami/soubory v sekci *Sdílené* mají stejnou funkci jako v sekci *Můj prostor* až na vyjímku mazání složek a souborů.

Uživatel, který není vlastníkem složky/souboru nemá oprávnění k její/jeho smazání. Funkce *delete* s ikonou malého koše je uživateli dostupná, nicméně funguje jako smazání sdílení tzn.: složka/soubor zůstane na úložišti, ale nebude již dále sdílena s tímto uživatelem.



Obrázek 7.5. Ukázka prostoru sdílených složek/souborů

7.3 Nastavení

V sekci nastavení popíši, jak fungují jednotlivé části aplikace týkající se osobního a administračního nastavení. V původní práci byla tato nastavení sjednocená pod jedním tlačítkem, ale ukázalo se jako lepší varianta tato nastavení oddělit.

- **Administrační část** – obsahuje správu organizačních jednotek a tvorbu uživatelských účtů pro externí pracovníky
- **Osobní část** – obsahuje osobní nastavení uživatele, tedy správu štítků, správu uživatelských skupin a změnu hesla k uživatelskému účtu (pro externisty)

Tlačítka pro navigaci mezi administrační a osobní částí nastavení jsou umístěna v horní části aplikace v menu vedle uživatelského jména přihlášeného uživatele. Uživatelské jméno funguje zároveň jako rozbalovací menu přes které je taktéž možné navigovat uživatele k nastavení. Přítomnost rozbalovacího menu je hlavně z důvodu responzivního designu, tedy především pro zařízení s menším displejem.

Při realizaci této práce byla, pro účely testování, veškerá data z nastavení simulována pomocí mock objektů, z důvodu nekompletnosti serverové strany Integračního portálu.

7.3.1 Organizační jednotky

Organizační jednotky představují skupiny uživatelů v rámci FEL, které slouží k organizaci těchto uživatelů. Členy těchto jednotek mohou být studenti a akademičtí pracovníci. V rámci organizace může správce organizačních jednotek např.: přiřazovat prostor na příslušném úložišti. Pro správu organizačních jednotek musí mít uživatel přiřazené oprávnění pomocí atributu *units* v poli *permissions* viz Obr. 7.3.5.

Případy užití implementované v kontextu organizačních jednotek:

- **Zobrazit všechny organizační jednotky** – umožní správci organizačních zobrazit seznam všech organizačních jednotek

```
...
permissions: {
    units: true,
    ...
}
...
```

Fragment 7.3.5. Oprávnění pro správu org. jednotek

- **Vytvořit organizační jednotku** - umožní správci organizačních jednotek vytvořit novou organizační jednotku
  - zadání názvu
  - přiřazení prostoru
  - zvolení správce
- **Smazat organizační jednotku** - umožní správci organizačních jednotek smazat libovolnou organizační jednotku
- **Editovat organizační jednotku** - umožní správci organizačních jednotek editovat jednotlivé atributy organizační jednotky

V rámci této práce nebyla provedena implementace funkce přidání uživatele do organizační jednotky, jelikož při vytváření uživatelského účtu je možné každému uživateli přiřadit libovolnou organizační jednotku a tento atribut lze při editaci daného uživatelského účtu upravit.

Pro práci s organizačními jednotkami v rámci celé fakulty bude použitá technologie **KOSapi** [21]. KOSapi poskytuje aplikační rozhraní v podobě RESTful webových služeb, které zprostředkovává přístup k vybrané části dat v databázi **KOS**. Z těchto dat je pak možné vyfiltrovat data související s uživatelem přihlášeným v aplikaci pomocí FELid. Tato funkcionalita zatím není na serveru implementována nicméně její realizací se zabývá bakalářská práce Vlastimila Fengla [22].

Serverová strana zatím neposkytuje většinu metod, které jsou potřeba pro kompletní funkcionalitu organizačních jednotek. Z tohoto důvodu byla data pro testování aplikace nasimulovaná přímo na straně klienta. Nicméně celá aplikace je, po drobných úpravách v kódu, připravena na integraci s funkční serverovou stranou. Veškeré potřebné úpravy pro následující integraci jsou dokumentovány v kódu a nebude tak s tímto procesem žádný problém.

7.3.2 Externisté

Externisté představují uživatelské účty uživatelů, kteří nejsou součástí FEL nebo ČVUT, jinými slovy nemohou se přihlásit pomocí FELid. Jelikož se na akademických výzkumech a projektech v rámci ČVUT podílí část externích pracovníků, byla implementace této funkcionality nutností. Externí pracovníci budou mít po přihlášení do aplikace přístup stejně jako uživatelé, kteří se přihlásí před FELid. Pro správu uživatelských účtů pro externí pracovníky musí mít uživatel přiřazené oprávnění pomocí atributu *externists* v poli *permissions* viz fragment 7.3.6.

Případy užití implementované v kontextu externích uživatelů:

- **Zobrazit všechny uživatelské účty externistů** – umožní správci externistů zobrazit seznam všech uživatelských účtů externistů
- **Vytvořit uživatelský účet externisty** – umožní správci externistů vytvářet uživatelské účty

```
...
permissions: {
    externists: true,
    ...
}
...
```

Fragment 7.3.6. Oprávnění pro správu org. jednotek

- zadat jméno a příjmení externisty
- zadat uživatelské jméno externisty
- zadat email externisty
- přiřadit prostor
- přiřadit externistu do organizační jednotky
- uvést uživatele na čí doporučení je registrace externisty provedena

- **Editovat uživatelský účet externisty** – umožní správci externistů editovat uživatelské účty
- **Smazat uživatelský účet externisty** – umožní správci externistů mazat uživatelské účty

Každý externí pracovník bude mít možnost změnit heslo ke svému účtu. Tato možnost nebude povolená pro uživatele přihlášené přes FELid, jelikož hesla k účtům FELid nebude spravovat aplikace integračního portálu.

7.3.3 Skupiny

Skupiny slouží uživateli pro seskupení jeho kontaktů na ostatní uživatele a usnadňuje tak sdílení s nimi. Uživatel si tedy může udělat libovolné množství skupin s libovolným počtem uživatelů. S těmito skupinami může uživatel dále sdílet data a nastavit oprávnění ke sdílení pro celou skupinu.

Tato funkcionalita najde uplatnění především při sdílení složek a souborů, kdy uživatel nemusí složku nebo soubor sdílet zvlášť s každým uživatele, ale může si vytvořit vlastní skupinu (seznam uživatelů), pomocí které může sdílet složku nebo soubor se všemi uživateli najednou.

Případy užití implementované v kontextu skupin:

- **Zobrazit všechny uživatelské skupiny** – umožní uživateli zobrazit seznam všech uživatelských skupin
- **Vytvořit skupinu** – umožní uživateli vytvářet skupiny
- **Editovat skupinu** – umožní uživateli editovat skupiny
- **Smazat skupinu** – umožní uživateli smazat skupiny
- **Přidat člena skupiny** – umožní uživateli přidat další uživatele jako členy do skupiny
- **Smazat člena skupiny** – umožní uživateli smazat členy skupiny

7.3.4 Štítky

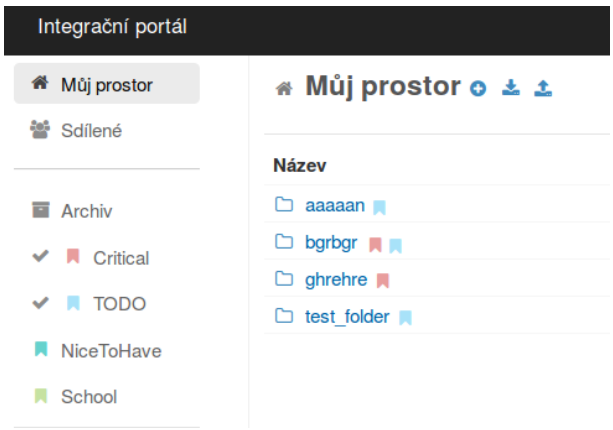
Štítky slouží k uspořádávání složek a souborů do kategorií. Uživatel má tak přehled, do jaké kategorie složka nebo soubor patří a navíc může přidat k souborům i složkám více štítků. Ty se pak zobrazí ve výsledcích vyhledávání všech štítků přiřazených příslušnému souboru či složce.

Případy užití implementované v kontextu štítků:

- **Zobrazit všechny štítky** – umožní uživateli zobrazit seznam všech štítků

- **Vytvořit štítek** – umožní uživateli vytvářet vlastní štítky
- **Editovat štítek** – umožní uživateli editovat štítky
- **Smazat štítek** – umožní uživateli smazat štítky
- **Přidat štítek k souboru/složce** – umožní uživateli přidávat štítky k jednotlivým souborům/složkám
- **Odebrat štítek ze souboru/složky** – umožní uživateli odebrat štítek z jednollivých souborů/složek
- **Vybrat štítky podle kterých filtrovat** – umožní uživateli vybrat jeden nebo více štítků, podle kterých se bude filtrovat obsah na příslušném úložišti

Štítky, podle kterých může uživatel filtrovat svoje soubory a složky fungují na principu zaškrtávacích tlačítek viz Obr. 7.6. Uživatel si může zvolit (zaškrtnout) jeden nebo více štítků, podle kterých se poté vyfiltruje obsah aktuálního adresáře nebo úložiště, ve kterém se právě uživatel v aplikaci nachází.



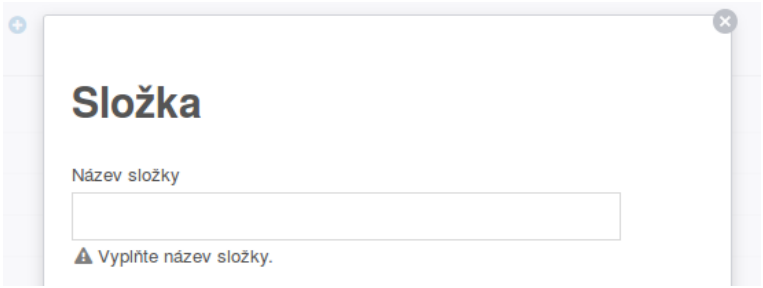
Obrázek 7.6. Ukázka filtrace složek/souborů podle přiřazených štítků

7.4 Formuláře

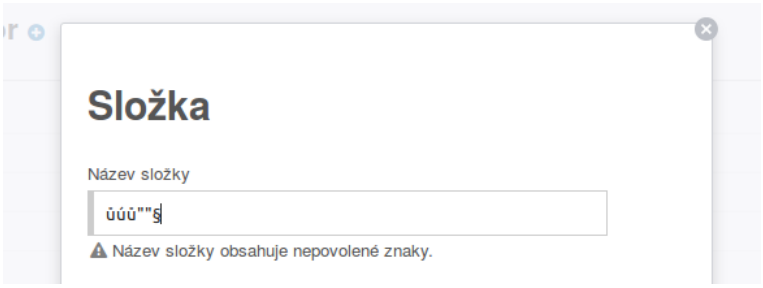
Klasický HTML formulář jsem použil v každém modálním dialogovém okně, které bylo v aplikaci vytvořeno. Pro stylování formulářů jsem použil CSS framework SemanticUI. Každé modální dialogové okno s formulářem obsahuje tlačítko pro jeho potvrzení, nicméně formulář se předá ke zpracování pomocí formulářové atribut direktivy *ng-submit*, která má jako parametr funkci. Tímto řešením se předejde klasickému odeslání formuláře a pomocí funkce v atributu *ng-submit* můžu zpracovat model napojený na tento formulář v kontroleru příslušného modálního dialogového okna.

7.4.1 Validace formulářů

Kontrolu validity formulářů jsem realizoval pomocí direktiv poskytnutých frameworkem AngularJS. AngularJS obsahuje spoustu direktiv, které slouží k validaci formulářů na straně klienta. Hlavní výhodou validace formuláře na klientské straně je to, že uživatel může být notifikován o chybách ještě před odesláním formuláře. Z tohoto důvodu se tyto direktivy jeví jako přívětivější pro uživatele, protože uživatel dostane ihned odezvu o chybě. Narozdíl od serverové validace, kdy je potřeba nejprve odeslat formulář a až poté se uživatel dozví, jestli došlo k nějaké chybě.[11] V praxi je samozřejmě serverová validace nezbytná a nelze spoléhat pouze na validaci na straně klienta.



Obrázek 7.7. 1. Ukázka notifikace ve formuláři



Obrázek 7.8. 2. Ukázka notifikace ve formuláři

7.4.2 Našeptávač

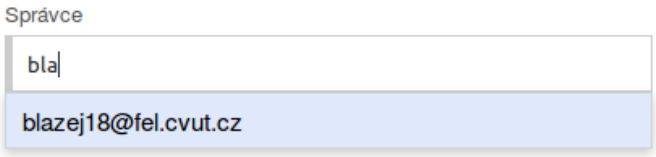
Během vývoje vznikl požadavek, aby u vybraných polí některých formulářů byla možnost vybrat data z nabídky. Jelikož obsah těchto dat je většinou poměrně rozsáhlý pro výběr pomocí klasických tlačítek výběru (tzv. radiobutton), zvolil jsem způsob výběru dat pomocí našeptávače viz Obr. 7.9. Uživatel tedy začne psát text do příslušného textového pole u kterého se zobrazí rozbalovací lišta s nabídkou dat, která odpovídá textu napsanému v příslušném textovém poli.

Pro implementaci této funkčnosti jsem využil modul MacGyver [13], který obsahuje direktivu pro autocomplete (našeptávač). Direktiva *mac-autocomplete* obsahuje atributy pomocí kterých lze upravit chování a prezentaci dat v rozbalovací liště a také textového vstupu. Pro ilustraci je přiložený fragment kódu 7.4.7.

```
<mac-autocomplete
     ...
     autocomplete="off"
     mac-autocomplete-source="users"
     mac-autocomplete-label="email"
     mac-placeholder='"e-mail uživatele"'
     ...>
</mac-autocomplete>
```

Fragment 7.4.7. ukázka použití direktivy mac-autocomplete

- **autocomplete** – atribut pro vypnutí nativního našeptávače prohlížeče
- **mac-autocomplete-source** – zdroj našeptávaných dat
- **mac-autocomplete-label** – popisek, který se uživateli zobrazí v rozbalovací liště
- **mac-placeholder** – zástupný text našeptávače



Obrázek 7.9. Našeptání dat ze psaného textu

7.5 Vyhledávání pomocí filtru

Vyhledávání v aktuálně zobrazených složkách jsem vyřešil pomocí AngularJS funkce filter. Díky two-way data binding se změny ve vyhledávacím poli ihned projeví v modelu aplikace, a tak se okamžitě vyfiltrují hledané složky. Uživatel se tak nemusí starat o žádná tlačítka, jen napíše text, podle kterého se mají složky a soubory vyfiltrovat. Filter je vložen přímo do HTML šablony a o samotné filtrování se stará AngularJS viz fragment 7.5.8.

```
...
<tr ng-repeat="folder in home.folders | filter:search">
...
```

Fragment 7.5.8. Ukázka použití filtru

- **ng-repeat** – direktiva, která vytváří nový HTML element pro každý element z pole
- **folder** – jeden element představující jednu složku
- **home.folders** – pole obsahující složky
- **filter** – AngularJS funkce pro filtrování obsahu
- **search** – proměnná obsahující text, podle kterého se data filtrují

7.6 Služby pro komunikaci se serverem

Modul *services.module* obsahuje několik služeb, které slouží pro komunikaci se serverem. Celý modul by se dal rozdělit na dvě hlavní části podle funkčnosti. Modelovou komunikační část a serverovou komunikační část. Modelová komunikační část zpracovává data z modelu aplikace a předává je dále na server pomocí dotazů na REST API. Pro každou skupinu funkcí týkajících se jedné části aplikace je samostatná služba (např.: userService, groupService atd.). Tyto dotazy zprostředkovává právě serverová komunikační část. Ta se stará o tvorbu HTTP požadavků, správu cookies webového prohlížeče, přesměrování stránek a autentizaci. Veškerý kód v tomto modulu je řádně dokumentován viz Obr. 7.6.9 podle konvencí psaní dokumentace ke zdrojovému kódu.

Na fragmentu 7.6.9 je ukázka kódu ze služby *spaceService*, konkrétně metody *getAll* pro získání všech složek a souborů z kořenového adresáře příslušného úložiště *spaceId* a vyfiltrované podle štítků z parametru *labels*.

```
/**
* Retrive all files and folders in a Space Root
*
* List all files and folders in root folder of a Space. Optionally
* use labels query parameter to list all the files and folders
* with the given label assigned.
*
* @param {int} spaceId - space identifier
* @param {array} labels - labels to filter by
* @returns {promise}
*/
getAll: function (spaceId, labels) {
        // Ready for API v0.2
        if (labels.length === 0) {
            return httpService.createRequest("GET", baseUrl + 'space/'
                    + spaceId, {}, "application/json");
        } else {
            var lbls = "" + labels[0];
            for (i = 1; i < labels.length; i++) {
                lbls = lbls + "," + labels[i];
            }
            return httpService.createRequest("GET", baseUrl + 'space/'
                    + spaceId + '?labels='
                    + lbls, {}, "application/json");
        }
    }
```

Fragment 7.6.9. Ukázka funkce *getAll* s dokumentací

Tato služba je součástí modelové komunikační části a k předání dat na server využívá službu *httpService* ze serverové komunikační části, pomocí které vytvoří HTTP dotaz na server.

Kapitola 8
Testování

Pro testování aplikace integračního portálu jsem využil framework Protractor, detailněji popsaný v sekci 8.1. Jelikož při vývoji nebyl plně funkční server integračního portálu, aplikaci jsem testoval pomocí test serveru na API v0.1 vytvořený v rámci práce P.Strnada [3], který poskytuje omezené množství funkcí. Ostatní data byla simulována pomocí mock objektů přímo na klientské straně.

8.1 Protractor test framework

Protractor je framework pro koncové testování speciálně vytvořený pro AngularJS aplikace. Jedná se o automatizované testy aplikace v realném webovém prohlížeči a interaguje tak s aplikací stejným způsobem, jako by to dělal uživatel. Protractor je postaven na Selenium WebDriveru (API pro práci s webovými prohlížeči), podporuje AngularJS lokátory a tak umožňuje testovat specifické prvky AngularJS frameworku.[7]

Příkazy protractoru jsou založeny na Jasmine frameworku [23], který obsahuje velmi intuitivní API a psaní testů je tak pro programátory velmi pohodlné. Při psaní protractor testů programátor píše reálné scénáře, na které uživatel při používání aplikace narazí. Při testování tak programátor vidí, co se v prohlížeči přímo děje a nemusí tak aplikaci po každé změně v kódu „proklikávat“. Protractor obsahuje poměrně dobrý logovací systém, kterým popisuje chyby ke kterým při testování došlo.

8.1.1 Průběh testování

K testování jsou potřeba 2 základní soubory:

- **spec.js** – testovací soubor obsahující popis scénářů samotných testů
- **conf.js** – konfigurační soubor k nastavení cesty k testovacím (spec) souborům, označení sad testů, adresy k Selenium serveru atd.

V souboru *conf.js* může uživatel nastavit webové prohlížeče, ve kterých bude testování aplikace probíhat. Protractor pak testy spouští postupně v obou prohlížečích a programátor tak může odhalit, jak chyby ve funkčnosti, tak chyby v kompatibilitě s webovými prohlížeči. Protractor umožňuje vytvořit sady testů, které lze poté spouštět zvlášť. Programátor tak nemusí testovat pokaždé celou aplikaci, ale vybere jen tu sadu testů, které se týkají provedených změn v kódu.

Pro spuštění jednotlivých sad testů je potřeba mít zapnutý Selenium server, pomocí kterého Protractor při testování spouští a komunikuje s jednotlivými webovými prohlížeči. Selenium server se spouští příkazem:

```
webdriver-manager start
```

Po úspěšném spuštění Selenium serveru může programátor spouštět jednotlivé sady testů pomocí příkazu:

```
protractor conf.js --suite nazev_sady_testů
```

Po dokončení testů je uživatel informován o výsledcích testů pomocí logu v příkazovém řádku. Pro každou sadu testů a pro každý webový prohlížeč je pro přehlednost log oddělen.

```
[chrome #2]
[chrome #2] Finished in 16.626 seconds
[chrome #2] 13 tests, 7 assertions, 0 failures
[chrome #2]

[launcher] 0 instance(s) of WebDriver still running
[launcher] firefox #1 passed
[launcher] chrome #2 passed
```

Obrázek 8.1. Výpis po dokončení sady testů

8.1.2 Testované případy užití

Jednotlivé testy jsem pro přehlednost rozdělil podle případů užití. Následuje seznam testovaných případů užití v rámci každé sady testů.

- **home**
  - Přihlásit se do aplikace
  - Zobrazit všechby složky/soubory
  - Filtrovat obsah adresáře
  - Vytvořit složku
  - Smazat složku
  - Editovat složku
  - Přidat složku do oblíbených
  - Odebrat složku z oblíbených
  - Odhlásit se z aplikace
- **admin**
  - Zobrazit administrátorské nastavení
  - Zobrazit všechny organizační jednotky
  - Vytvořit organizační jednotku
  - Editovat organizační jednotku
  - Smazat organizační jednotku
  - Zobrazit seznam uživatelských účtů externistů
  - Vytvořite uživatelský účet externisty
  - Editovat uživatelský účet externisty
  - Smazat uživatelský účet externisty
- **settings**
  - Zobrazit osobní nastavení
  - Zobrazit všechny štítky
  - Vytvořit štítek
  - Editovat štítek
  - Smazat štítek
  - Zobrazit všechny uživatelské skupiny
  - Vytvořit uživatelskou skupinu
  - Editovat uživatelskou skupinu
  - Přidat členy do skupiny
  - Odebrat člena ze skupiny
  - Smazat uživatelskou skupinu

8.1.3 Struktura testů

Pro aplikaci integračního portálu jsem vytvořil testy pro hlavní části aplikace, které bude uživatel nejčastěji používat a kde by se mohlo potenciálně vyskytovat nejvíce chyb.

```
...
tests/
    e2e/
        /home
            home.po.js
            home.spec.js
        /settings
             settings.po.js
             ...
        /admin
            ...
        conf.js
...
```

Soubory s koncovkou **.po.js* jsou tzv. *page objects*. Tyto soubory slouží k zapouzdření informací o webových elementech testované aplikace a umožňují tak psát čistší a přehlednější testy. *Page object* může být použit napříč různými testy a pokud se změní šablona webové aplikace, stačí upravit pouze *page object*.[24]

8.2 Zhodnocení testů

Během testování jsem objevil několik chyb ve funkčnosti aplikace (např.: po smazání složky zůstala složka v oblíbených, po editaci složky zůstal v oblíbených starý název složky). Pro opravu těchto chyb bylo potřeba jen pár úprav v kódu, tímpádem opravit je nebyl problém. I přesto, že jsem testy nevytvářel hned v rané části vývoje, ale až ke konci implementaci, testování mi usnadnilo hodně práce a do budoucna usnadní práci i dalším programátorům podílejícím se na vývoji klientské části integračního portálu.

V rámci testování aplikace jsem vytvořil 3 hlavní sady testů, které jsem dále rozdělil na jednotlivé scénáře interakce s aplikací. Testování jsem prováděl na prohlížeči *Firefox WebBrowser* v. 38.0 a *Chromium WebBrowser* v. 41.0.2272.76 pro Ubuntu 14.04 (64-bit). Počet testů v rámci každé sady testů:

- **home** sada testů – 10 testů z toho 10 úspěšných (100% úspěšnost)
- **admin** sada testů – 12 testů z toho 12 úspěšných (100% úspěšnost)
- **settings** sada testů – 13 testů z toho 13 úspěšných (100% úspěšnost)

Kapitola 9
Závěr

Při realizaci tohoto projektu jsem pokračoval v předchozí práci na projektu v rámci diplomové práce Kateřiny Hašlarové [4]. V moji implementaci se mi podařilo úspěšně navázat na již vytvořené grafické uživatelské rozhraní a implementovat požadované chybějící funkcionality. Provedl jsem i několik změn v grafickém uživatelském rozhraní, tyto změny jsou popsány v kapitole 6. Pro realizaci klientské části Integračního portálu jsem použil framework AngularJS [5] a několik dalších modulů viz 3.6.

Implementované funkcionality aplikace byly úspěšně otestovány pomocí Protractor [7] koncových testů. Samotný proces testování a výsledky testů jsou popsány v kapitole 8. Při testech jsem objevil několik chyb v implementaci, které nebylo problém opravit. Díky použitému frameworku pro koncové testování webových aplikací je pro budoucí vývoj možné testy snadno modifikovat nebo rozšiřovat.

Celkově bych svojí práci zhodnotil jako velice přínosnou pro realizaci projektu Integračního portálu. Podařilo se mi implementovat veškeré požadované funkcionality aplikace, která je tak připravena k integraci. Ačkoli během mé práce došlo k několika změnám, konkrétně velké změny v REST API a dále chování některých částí aplikace (např.: archiv) které rychlost vývoje dost ovlivnily, stihl jsem práci vypracovat v termínu.

Pro budoucí vývoj jsem vytvořil dobrou a robustní webovou aplikaci Integračního portálu, která je připravena na integraci do zkušebního provozu. Tato aplikace je díky robustnímu základu snadno udržovatelná a rozšiřitelná o další funkcionality. Další programátoři, kteří budou pracovat na tomto projektu tak budou moci snadno navázat na moji práci. Již teď jsou známy některé další požadavky a změny, které proběhnou v rámci celého projektu, takže moje aplikace nebude úplně finální verze. Na projektu budou pracovat další studenti, kteří hotovou aplikaci budou upravovat podle nových požadavků. V rámci projektu Integračního portálu se bude v budoucnu implementovat integrace dalších služeb, kterými jsou např.: Google Drive [10], Dropbox [9] a další.

Literatura

[1] *CESNET*.
http://www.cesnet.cz/sdruzeni.

[2] Ondřej Macek. *CESNET Zálohování – wiki*.
https://gitlab.fel.cvut.cz/cesnet-zalohovani/dokumentace/wikis/intro/o-projektu.

[3] Petr Strnad. *Integrační portál pro sdílení a zálohování dat*. 2014.

[4] Kateřina Hašlarová. *Front-end pro portál pro sdílení souborů*. 2014.

[5] *AngularJS – JavaScript Framework*.
https://angularjs.org/.

[6] *Code School AngularJS*.
http://campus.codeschool.com/courses/shaping-up-with-angular-js/intro.

[7] *Protractor – end to end testing for AngularJS*.
http://angular.github.io/protractor/.

[8] *Semantic UI – CSS Framework*.
http://semantic-ui.com/.

[9] *Dropbox*.
https://www.dropbox.com/.

[10] *Drive Google*.
https://drive.google.com.

[11] *AngularJS documentation*.
https://docs.angularjs.org.

[12] *AngularUI – UI-Router*.
http://angular-ui.github.io/ui-router/site/#/api/ui.router.

[13] Adrian Lee. *MacGyver module for AngularJS*.
http://angular-macgyver.github.io/MacGyver/0.6.1/.

[14] Eli Grey. *Saving generated files on the client-side*. 2011.
eligrey.com/blog/post/saving-generated-files-on-the-client-side.

[15] *Analýza požadavků*.
http://cs.wikipedia.org/wiki/Anal%C3%BDza_po%C5%BEadavk%C5%AF.

[16] Google. *Best Practice Recommendations for Angular App Structure*. 2014.
https://docs.google.com/document/d/1XXMvReO8-Awi1EZXAXS4PzDzdNvV6pGcuaF4Q9821Es/mobilebasic?pli=1.

[17] *Representational State Transfer*.
http://cs.wikipedia.org/wiki/Representational_State_Transfer.

[18] *OAuth 2.0*, 2006.
http://oauth.net/2/.

[19] *O systému FELid*.
https://wiki.fel.cvut.cz/net/felid/about.

[20] Gabe Scholz. *Authentication made simple in Single Page AngularJS Applications*. 2004.
http://brewhouse.io/blog/2014/12/09/authentication-made-simple-in-single-page-angularjs-applications.html.

[21] Jakub Jirůtka. *KOSapi*.
https://kosapi.fit.cvut.cz/projects/kosapi/wiki.

[22] Vlastimil Fengl. *Back-end pro integrační portál*. 2015.

[23] *Jasmine: Behavior-Driven JavaScript*.
http://jasmine.github.io/.

[24] *Protractor documentation*.
https://github.com/angular/protractor/blob/master/docs/page-objects.md.

Příloha A
Zkratky

API	Application programming interface
ČVUT	České vysoké učení technické
DOM	Document Object Model
FEL	Fakulta elektrotechnická
KOS	Studijní informační systém ČVUT
MVC	Model-view-controller
REST	Representational state transfer
SPA	Single-page application

Příloha B
Obsah přiloženého CD

```
bp_text/
    /img/          - obrázky použité v textu práce
    /text/         - zdrojové kódy k textu práce
src/               - veškeré zdrojové kódy aplikace
blazej18_bp.pdf    - výsledný dokument textu práce
README.txt         - informace o projektu
```