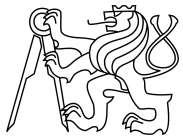
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Aplikace pro správu tematických okruhů k státní závěrečné zkoušce
Student:	Bc. Vojtěch Petrus
Vedoucí:	Ing. Vojtěch Jirkovský
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2016/17

Pokyny pro vypracování

1) Analyzujte požadavky na sestavování tematických okruhů (TO) k státní závěrečné zkoušce (SZZ).
2) Prostudujte dostupné datové zdroje (bílá kniha, KOSapi apod.).
3) Navrhněte a implementujte backend aplikace pro správu TO ve formě API. Aplikace bude umožňovat zejména:
 - podle studijních plánů a uživatelské konfigurace sestavit seznam oborů a předmětů k SZZ,
 - vkládání a aktualizaci textů TO pro každý semestr, ve kterém byl předmět vyučovaný,
 - kopírování textu TO z předchozích běhů předmětu.
4) Navrhněte a implementujte frontend aplikace využívající API vhodný pro integrování do systému EDUX.
5) Implementaci řádně otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 11. června 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Aplikace pro správu tematických okruhů k státní závěrečné zkoušce

Bc. Vojtěch Petrus

Vedoucí práce: Ing. Vojtěch Jirkovský

9. května 2016

Poděkování

Děkuji vedoucímu Ing. Vojtěchu Jirkovskému za ochotu a součinnost při řešení četných komplikací a rodině a přítelkyni za neustálou podporu při psaní této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 9. května 2016 .....

Odkaz na tuto práci

Petrus, Vojtěch. *Aplikace pro správu tematických okruhů k státní závěrečné zkoušce*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato práce se zabývá správou tematických okruhů k státní závěrečné zkoušce a jejich zobrazení studentům. Zkoumá rozšíření běžící na výukové aplikaci EDUX a snaží se odhalit jejich nedostatky a možnosti vylepšení. Cílem práce je navrhnout a implementovat nástroje, které bude možné snadno integrovat do stávajících systémů a které maximálně usnadní správu okruhů.

Klíčová slova tematické okruhy, závěrečné zkoušky, EDUX, webové technologie

Abstract

This work focuses on the administration of topics of final state examinations and how the topics are displayed to students. It examines plugins currently running on the educational application EDUX and tries to find their drawbacks and enhancement possibilities. The aim is to design and implement tools which could be easily integrated into current systems and which would make the administration of topics much easier.

Keywords topics, final examinations, EDUX, web technologies

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Struktura této práce

Tato práce je rozdělená do šesti hlavních kapitol. V první z nich se podívám na současný stav, zjistím, jak funguje správa tematických okruhů pro státní závěrečné zkoušky, a pokusím se odhalit nedostatky. Těchto poznatků využiji v druhé kapitole, kde si stanovím cíle a požadavky na výslednou aplikaci. V sekci *Analýza* provedu návrh uživatelského rozhraní a vyjmenuji použité technologie. V páté kapitole rozeberu realizaci aplikačního rozhraní backendu. Kapitola *Realizace frontend části* popíše strukturu a některé technické detaily frontend části aplikace. Závěrečnou kapitolu pak věnuji testování.

Kapitola 1

Současný stav

Správa státnicových okruhů na fakultě informačních technologií se točí výhradně kolem **EDUX**u[1]. *EDUX* je zavedený školní systém sloužící k vytváření a uchování informací, které poskytují vyučující studentům, a naopak, sdílení vypracovaných prací studentů učitelům. Přes *EDUX* vyučující oznamují aktuality o vyučovaných předmětech, nahrávají na něj studijní materiály, návody a odkazy na zajímavé externí zdroje. Studenti zase nahrávají své semestrální práce, vytvářejí k nim dokumentace a další pomocné stránky ve svém namespacu[2].

EDUX běží na open source[3] systému DokuWiki (byť za roky používání značně upraveném a rozšířeném). DokuWiki je víceúčelový wiki[4] software, který je jednoduchý na používání a nepotřebuje ke svému chodu databázi[1].

1.1 Možné zdroje dat

Jakákoliv aplikace zabývající se tematickými okruhy by měla mít přístup k datům popisujícím hierarchii školních struktur z důvodu usnadnění práce administrátorům. Týká se to hlavně struktury programů, oborů a studijních plánů, které pod ně spadají, a v nesposlední řadě i předmětů zařazených pod studijní plány. V případě, že by tato data nebyla k dispozici, musel by je správce aplikace vyplňovat a udržovat aktuální ručně, což by bylo velmi náročné.

V této práci jsem se zaměřil na dva potenciální zdroje dat a to *Bílou knihu* a *KOSapi*.

[1] `http://edux.fit.cvut.cz`

[2] namespace (jmenný prostor) - vyhrazený prostor pro uživatele EDUXu v rámci jednoho předmětu, kam může nahrávat osobní informace a dokumenty

[3] `http://en.wikipedia.org/wiki/Open-source_software`

[4] `http://en.wikipedia.org/wiki/wiki`

1.1.1 KOSapi

KOSapi poskytuje aplikační rozhraní v podobě RESTful[5] webových služeb, které zprostředkovávají přístup k vybrané části dat v databázi studijního informačního systému KOS[2]. K základním datům je po zaregistrování aplikace přístup autentifikovaný pomocí OAuth 2.0[6] protokolu. Nyní se podíváme na některé zdroje, které by mohly být pro naši aplikaci relevantní.

1.1.1.1 Studijní programy

- **URI**: `https://kosapi.fit.cvut.cz/api/3/programmes/{code}`
- **Příklad:** `https://kosapi.fit.cvut.cz/api/3/programmes/MI`

Obsahuje základní informace o programech. Pro nás jsou zajímavé programy bakalářský, magisterský a jejich anglické protějšky s kódy končícími na E (BIE, MIE).

1.1.1.2 Obory

- **URI**: `https://kosapi.fit.cvut.cz/api/3/branches/{id}`
- **Příklad:** `https://kosapi.fit.cvut.cz/api/3/branches/4779`

Programy se dělí do jednotlivých oborů. Obor může být například *Webové a softwarové inženýrství*, který se dál dělí na konkrétní **zaměření** (např. *zaměření webové inženýrství*). Některé obory se však už dále nedělí, typicky to jsou bakalářské obory.

1.1.1.3 Studijní plány

- **URI**: `https://kosapi.fit.cvut.cz/api/3/studyPlans/{code}`
- **Příklad:** `kosapi.fit.cvut.cz/api/3/studyPlans/MI-WSI-WI.2013`

Studijní plán je základní strukturou, jež jednak určuje, které předměty musí student během studia absolvovat, ale také nepřímo ze kterých předmětů se bude **státní závěrečná zkouška** (*SZZ*) skládat. Ovšem existují i výjimky, kdy například povinný předmět studijního plánu není součástí SZZ. Informace o tom, jestli je či není předmět zařazen, bohužel (alespoň podle informací, které jsem získal) nelze z KOSu, ani žádného jiného zdroje dat získat.

Je běžné, že obor (případně zaměření) má vícero studijních plánů, ty pak mají názvy podle roku nástupu, například *Zaměření Webové inženýrství - verze pro ty, kteří se zapsali v roce 2013*.

[5] `https://en.wikipedia.org/wiki/Representational_state_transfer#Applied_to_web_services`

[6] `http://oauth.net/2/`

1.1.1.4 Skupiny předmětů

- **URI**: `https://kosapi.fit.cvut.cz/api/3/studyPlans/{code}/coursesGroups`
- **Příklad:** `kosapi.fit.cvut.cz/api/3/studyPlans/MI-WSI-WI.2013/coursesGroups`

Studijní plány dále obsahují skupiny předmětů, které rozdělují předměty do oddělených kategorií. Jelikož se z volitelných předmětů neskádají závěrečné zkoušky, zajímají nás v podstatě pouze tyto tři skupiny předmětů:

- **PP** - povinné předměty programu
- **PO** - povinně oborové předměty
- **PZ** - povinné předměty zaměření

Ovšem, jak jsem již uvedl v sekci 1.1.1.3, po získání předmětů z těchto sekcí pravděpodobně dostaneme i předměty, které součástí SZZ nebudou.

1.1.2 Bílá kniha

Bílá kniha[7] je mimo jiné název aplikace, která v jednoduché formě zobrazuje seznam studijních plánů, předměty, které plány obsahují, či doporučené průchody jednotlivými semestry. Zdrojem dat pro bílou knihu je XML soubor exportovaný z KOSu, nacházející se na adrese `https://bk.fit.cvut.cz/data/bk.xml`. Tento soubor může být poměrně obsáhlý, v řádu jednotek MB.

1.2 Skript Ing. Vojtěcha Jirkovského

Než mi vedoucí mé diplomové práce, Ing. Vojtěch Jirkovský, nabídl toto téma, používali se na *EDUX*u jeho skripty pro generování státnicových okruhů. V této sekci se pokusím stručně popsat jejich fungování.

Data pro aplikaci se berou z XML[8] exportu ze školního studijního informačního systému KOS[9], který je zdrojem dat i pro *Bílou knihu*.

Celý proces začíná spuštěním shellového skriptu[10] `zpracuj-studijni-plan.sh`, který stáhne výše zmíněný XML soubor. Tato data jsou pomocí XSLT transformace[11] převedena do CSV[12] formátu, se kterým se dále pracuje.

Následně data zpracují tyto skupiny skriptů:

- Skupiny souborů `roztrid-soubory*.sh` slouží ke kopírování souborů okruhů do adresářů pro semestr, kdy byl předmět vyučován.

[7] `https://bk.fit.cvut.cz`
[8] Extensible Markup Language
[9] `http://kos.is.cvut.cz`
[10] `https://cs.wikipedia.org/wiki/Shellový_skript`
[11] `http://www.w3.org/TR/xslt`
[12] comma-separated values

- Soubory `vypis-*.sh` umí vypsat různá data z CSV studijních plánů.
- Skripty `vytvor-strukturu-*.sh` volají hlavní skript `vytvor-strukturu.sh` s přednastavenými parametry pro daný program.
- Skript `vytvor-strukturu.sh` na základě několika parametrů (kód programu, délka studia, název konfiguračního souboru se studijními plány...) vygeneruje všechny *DokuWiki* stránky s odkazy na jednotlivé programy a studijní plány. Vytvoří se kostra pro jednotlivé předměty a existuje-li soubor s tematickými okruhy z minulých semestrů, zkopíruje se bez ohledu na změny, které mezi tím mohly nastat.

1.3 Skript Ing. Miroslava Balíka, Ph.D

Během psaní této práce se na EDUXu objevil nový formát zobrazení tematických okruhů. Seznam okruhů byl vygenerován skriptem, jehož autorem je Ing. Miroslava Balík, Ph.D. Pan Balík nám poskytl následující informace:

Zdrojem dat pro tento skript je Google Doc[13] distribuovaný mezi jednotlivé vyučující. Ti vyplní do připravené tabulky potřebná data, která se následně zpracují jednoduchým skriptem.

Skript nejprve transponuje tabulku a pro každý řádek vyrobí jednu položku formátovanou pro DokuWiki, aby se na EDUXu zobrazil jako jeden řádek. Příklad výstupu:

```
  | 1. | TO-BI-SPOL-1 | Výroková logika - pravdivostní tabulky,
disjunktivní a konjunktivní normální tvary, universální systém logických
spojek, logický důsledek a jeho ověření. Booleova algebra (jazyk,
vlastnosti, uspořádání, příklady, věta o isomorfismu). | BI-MLO |
```

1.4 Nevýhody současného řešení

Nevýhody skriptu Ing. Miroslava Balíka, Ph.D jsou poměrně zřejmé. Jedná se o rychlé řešení, které ušetří ruční přepisování okruhů na EDUX, ale kromě toho nepřináší velké výhody oproti naivnímu řešení, kdy by například vyučující odesílali finální verze okruhů administrátorovi, který by z nich ručně vytvářel EDUXové stránky.

I sofistikovanější balíček skriptů Ing. Vojtěcha Jirkovského s sebou nese určité nevýhody:

- Je náročný na konfiguraci a spouštění, což musí administrátor provádět před každým státnicovým obdobím.
- Je náchylný k chybám při konfiguaraci, které mohou způsobit nekonzistentní či neaktuální data.

[13] `https://www.google.com/docs/about/`

- Neřeší uživatelská práva a oddělení zodpovědnosti. Není možné delegovat vybrané okruhy pod správu garantů předmětů.
- Není možné přehledně verzovat jednotlivé okruhy.

Kvůli zmíněným nedostatkům vznikla potřeba pro novou aplikaci, která by tyto problémy řešila. Požadavky na takovou aplikaci rozeberu v následující kapitole.

Kapitola 2

Cíle a požadavky

Při sestavování cílů této práce a požadavků na výslednou aplikaci jsem vycházel z konzultací s vyučujícími a vedoucími pracovníky školy. Ing. Ivan Halaška nám (mně a vedoucímu práce Ing. Vojtěchu Jirkovskému) objasnil jaká data, v jaké podobě a jakým způsobem lze získat z *KOS*u. Ing. Miroslav Balík, Ph.D. nám nastínil svou vizi státnicových okruhů v následujících letech a sdělil své požadavky a návrhy k budoucí aplikaci. S Ing. Tomášem Kadlecem jsme řešili technickou stránku věci a pomohl nám vyřešit některé základní otázky týkající se implementace.

Bohužel během práce na aplikaci se požadavky poměrně významně a často měnily. Některé části aplikace přestaly být důležité, jiné prošly velkými změnami. Jednou z takových změn je například vize neměnných studijních plánů 2.2, se kterou přišel pan Ing. Miroslav Balík, Ph.D. a která ne zcela koresponduje se zadáním práce. V takovýchto případech jsem se rozhodl se striktně nedržet schváleného zadání, ale raději vyjít vstříc skutečným požadavkům na aplikaci.

2.1 Rozdělení na dvě aplikace

Na základě návrhu Ing. Tomáše Kadlece jsem se rozhodl rozdělit budoucí aplikaci na dvě samostatné podaplikace - **backend s API** a **JavaScriptový frontend**.

2.1.1 Backend aplikace

Jako **backend** se označuje část webové aplikace, která slouží k administraci webu a ke zpracování dat [3]. V našem případě bude backend obsahovat veškerou logiku a ukládat stav aplikace do relační databáze. Bude řešit vkládání nových tematických okruhů, zabezpečení uživatelů, správu uživatelských oprávnění, import dat ze vzdálených zdrojů a spoustu dalších úkolů. Klíčo-

vou podmínkou pro použitelnost výsledné aplikace je, aby backend poskytoval jasně definované **API**.

API (Application Program Interface) neboli rozhraní pro programování aplikace je sada postupů, protokolů a nástrojů pro vytváření softwaru. Pomocí API specifikujeme, jak budou jednotlivé komponenty programů či celé aplikace spolu komunikovat [4].

Jako každé dobré API musí být i toto důkladně otestováno a zdokumentováno.

2.1.2 Frontend aplikace

Pojmem **frontend** se v oblasti webových aplikací míní část webu viditelná běžným návštěvníkům [5]. V tomto konkrétním případě bude frontend aplikace to jediné, co uživatel uvidí. Bude obsahovat to hlavní, co studenty zajímá – tedy výpis aktuálních tematických okruhů. Ale zároveň v ní nalezneme veškeré prvky uživatelského rozhraní, které umožní efektivní správu vyučujícím i administrátorům.

Z důvodů popsaných v sekci 1 je pravděpodobné, že aplikace poběží na EDUXu. Ovšem ve školním prostředí se požadavky často mění a jedna z informací, která se ke mně dostala, byla, že DokuWiki by mohlo být nahrazeno nějakou interně vyvíjenou aplikací. Rozhodl jsem se primárně počítat s integrací do EDUXu, ovšem zároveň nedělat aplikaci přímo závislou na DokuWiki. Koneckonců, frontend plně oddělený od aplikační logiky takové situaci nahrává.

Pro případ, že aplikace bude nasazená na EDUXu, je potřeba zajistit co nejjednodušší integraci do stávajícího systému. Jako bakalářskou práci jsem programoval sadu rožšíření pro DokuWiki, které umožnily správu časově omezených přístupových pravidel pro studenty i vyučující. Díky tomu mám poměrně jasnou představu o tom, jak omezující může být práce s neaktuální verzí DokuWiki, která je navíc protkaná pluginy různých autorů [6]. Nicméně zde se jedno bezbolestné řešení nabízí – postavit frontend aplikaci čistě na nějakém skriptovacím jazyce, který poběží v prohlížeči a bude se pomocí **AJAX**ových[14] volání dotazovat vzdálené API, která obstará veškerou aplikační logiku.

Ne vždy musí být frontend a backend části takto striktně odděleny, ale v našem případě to přinese nesporné výhody.

2.1.3 Výhody rozdělení aplikace

Rozdělení na samostatný frontend a backend sice přináší určité ztížení co se programovací části týče, nicméně v tomto případě výhody jasně převažují:

- **Snadné nasazení na EDUX** (či do jiného systému) – jak jsem již naznačil, aplikace psaná čistě pomocí JavaScriptu a HTML (viz sekce

[14] `https://en.wikipedia.org/wiki/Ajax_(programming)`

Technologie 3.2) velmi usnadní nasazení na EDUX, jedná se v podstatě jen o načtení skriptů v hlavičce či v těle stránky.

- **Nezávislost na uživatelském rozhraní** - pokud bude chtít někdo v budoucnu změnit uživatelské rozhraní (například kvůli změně designu zbytku stránky) nebo se objeví další, lepší JavaScriptový framework (což pravděpodobně nebude trvat dlouho) a bude vyžadováno kompletní předělání vzhledu, pak se takové změny vůbec nedotknou backendové API.

- **Znovupoužitelnost backendu** - API není limitována jenom na frontendovou aplikaci, tudíž není problém, aby data o tematických okruzích či o složení předmětů ve studijních plánech mohla využívat i jiná aplikace.

2.2 Vize neměnných studijních plánů

V sekci 1.1.1 jsem zmínil, že studijní plány se často mění každým rokem. V závislosti na tom se také mohou měnit okruhy pro semestr, ve kterém se závěrečné zkoušky konají. Proto například skripty pana Ing. Jirkovského poměrně komplikovaně zjišťují, kterého semestru se studijní plán týká a podle toho verzují a případně kopírují jednotlivé okruhy.

Podle vyjádření Ing. Miroslava Balíka, Ph.D. se ovšem tato situace má změnit. Cíl do budoucna je takový, že studijní plány a okruhy se jednou vytvoří a zůstanou v té samé podobě i několik let. Okruhy budou obecnějšího rázu, aby byly méně náchylné k nutnosti úprav.

Výhody toho řešení jsou zřejmé. Ubyde složitostí při správě okruhů, nastane výrazné zjednodušení pro studenty a zároveň se vyhneme nepříjemným situacím, kdy student skládá státní závěrečné zkoušky o rok později. V tom případě by totiž měl být teoreticky zkoušen z jiných okruhů než jeho kolegové se zkouškou v řádném termínu. V případě, že by se okruhy znatelně lišily, by toto mohl být velký problém. Proto jeden z požadavků na výslednou aplikaci byl, aby počítala se studijními plány a okruhy, které budou zůstávat více semestrů.

2.3 Správa tematických okruhů

Při úvodní analýze se objevil požadavek, který měl určující ráz pro výslednou aplikaci. A to, že v aplikaci budou osoby s oprávněním **garant předmětu**, které budou zodpovědné za seznam okruhů pro svůj předmět. Logicky tedy musí mít možnost vkládat, editovat a mazat okruhy.

Předmět může být součástí více studijních plánů. Od pana Balíka také přišel požadavek, aby byla možnost některé okruhy do studijního plánu neza-

řazovat. Vznikla tedy nutnost, aby byla vytvořena role osoby, která bude mít na starost schvalování okruhů. Tou rolí je **garant studijního plánu**[15].

Garanti předmětů tedy vytvoří jakousi nabídku, ze které si *garanti studijních plánů* vyberou, a budou zodpovídat za výsledný seznam okruhů.

2.4 Správa oprávnění uživatelů

Protože funkce přidělované zaměstnancům školy se často mění a protože momentálně není způsob jak automaticky získat všechny potřebné informace, nezbývá než implementovat ruční správu oprávnění. Ta se týká garantů předmětu i garantů studijních plánů.

Je ovšem důležité zabránit tomu, aby nevznikaly kolize[16], když budou zároveň upravovat garanti předmětů i studijních plánů.

U tematických okruhů je zavedené pravidlo, že od data uveřejnění už **není možné okruhy měnit**. Je to z toho důvodu, aby studenti, připravující se na závěrečné zkoušky, měli jistotu, že se s okruhy už nic nestane. Pro naše potřeby stačí toto pravidlo trochu rozšířit, aby se nepřekrývala období, kdy mohou skupiny uživatelů upravovat okruhy. Bylo by vhodné, aby nějaký pověřený uživatel či uživatelé mohli tato časová pravidla přidělovat nebo odebírat.

Celá situace by pak mohla vypadat například takto:

1. V první fázi vytvářejí garanti předmětů seznam okruhů pro své předměty.
2. V druhé fázi garanti studijních plánů sestavují z nabídnutých okruhů finální seznamy. Garanti předmětů už nemají přístup.
3. V poslední fázi už nemají garanti přístup vůbec. Nejnutnější změny smí provádět pouze několik autorizovaných osob, ale změny už by neměly mít vliv na význam okruhů.

2.5 Import dat z KOSu

Výhody KOSapi jako zdroje dat oproti XML exportu pro Bílou knihu jsou zřejmé:

- Export pro Bílou knihu je objemný a práce s ním pravděpodobně nebude efektivní.
- V KOSapi jsou data logicky strukturována a dobře zdokumentována.

[15] během prvotního sběru požadavků se hovořilo také o správci oboru či zařazení, více v sekci 3.5

[16] kolize nikoliv ve smyslu nějakého nedefinovaného stavu, ale ve smyslu, že si uživatel nevšimne nových úprav jiného uživatele

- S XML exportem dostaneme velké množství dat, které pro naši aplikaci nejsou relevantní, a naopak některá důležitá data nemusí být k dispozici.

Backendová aplikace by proto měla být schopna importovat potřebná data z KOSapi.

2.6 Funkční požadavky

Jak jsem zmínil v úvodu této kapitoly, při sestavování následujího seznamu jsem vycházel více z reálných požadavků na aplikaci, které se postupem času objevovaly, než z původního zadání. Pro jednoduchost jsem nerozlišoval mezi požadavky na frontend a backend aplikace, ale díval se na výslednou práci jako na celek.

Důvod, proč mají být pod studijními programy zobrazeny přímo studijní plány, jsem popsal v sekci *Uživatelské role* 3.5.

- **F1. Aplikace umožní všem uživatelům zobrazit seznam studijních programů a studijních plánů pod ně spadajících.**
- **F2. Aplikace umožní všem uživatelům zobrazit detail konkrétního studijního plánu s přiřazenými předměty a schválenými tematickými okruhy.**
- **F3. Aplikace bude rozlišovat mezi různými skupinami uživatelů (*garanti studijních plánů*, *garanti předmětů*...) a na základě těchto rolí bude spravovat přístupová práva.**
- **F4. Aplikace umožní oprávněným uživatelům přidávat a odebírat *garanty předmětu*.**
- **F5. Aplikace umožní oprávněným uživatelům přidávat a odebírat *garanty studijních plánů*.**
- **F6. Aplikace umožní *garantům předmětu* vytvářet, editovat a mazat tematické okruhy.**
- **F7. Aplikace umožní *garantům studijních plánů* schvalovat či odmítat okruhy z nabízeného seznamu.**
- **F8. Aplikace bude verzovat a zobrazovat historii úprav tematických okruhů oprávněným uživatelů.**
- **F9. Aplikace umožní *garantům studijních plánů* aktualizovat svůj studijní plán pomocí KOSapi.**
- **F10. Aplikace umožní *administrátorům* importovat libovolný studijní plán pomocí KOSapi.**

- **F11. Aplikace umožní *administrátorům* nastavovat časově omezená práva pro přístup ostatních skupin uživatelů.**
- **F12. Aplikace zobrazí všechny okruhy vytvořené v rámci jednoho předmětu.**

2.7 Nefunkční požadavky

Nefunkční požadavky vychází z konzultací s vedoucím a Ing. Kadlecem a z důvodů popsaných v této kapitole.

- **N1. Aplikace bude rozdělena na dvě samostatné podaplikace – backend obsahující veškerou logiku a starající se o uchovávání dat a frontend s uživatelským rozhraním.**
- **N2. Obě aplikace musí být dostupné přes webový prohlížeč.**
- **N3. Frontend aplikace musí být snadno nasaditelný do systému EDUX.**
- **N4. Backend aplikace bude poskytovat API a pouze skrze něj bude frontend komunikovat s modelovou vrstvou.**
- **N5. Backend API musí být správně zdokumentováno a otestováno.**
- **N6. Obě aplikace musí být z důvodu znovupoužitelnosti co nejvíce oddělené.**

Kapitola 3

Analýza a návrh

3.1 Entitně vztahový model

Po vyhodnocení požadavků z předchozí kapitoly jsem navrhl entitně vztahový model[17], jež můžete vidět na obrázku 3.1. Tento model byl využit pro tvrobu databázového schéma. Nyní bych rád stručně popsal navržené entity.

3.1.1 User

Přihlášený uživatel.

Vlastnosti entity:

1. **id** – identifikátor uživatele
2. **login** – přihlašovací jméno, stejné jako na EDUX
3. **authToken** – autentifikační token, unikátní pro každého uživatele

3.1.2 UserRole

Role uživatele sloužící k autorizaci.

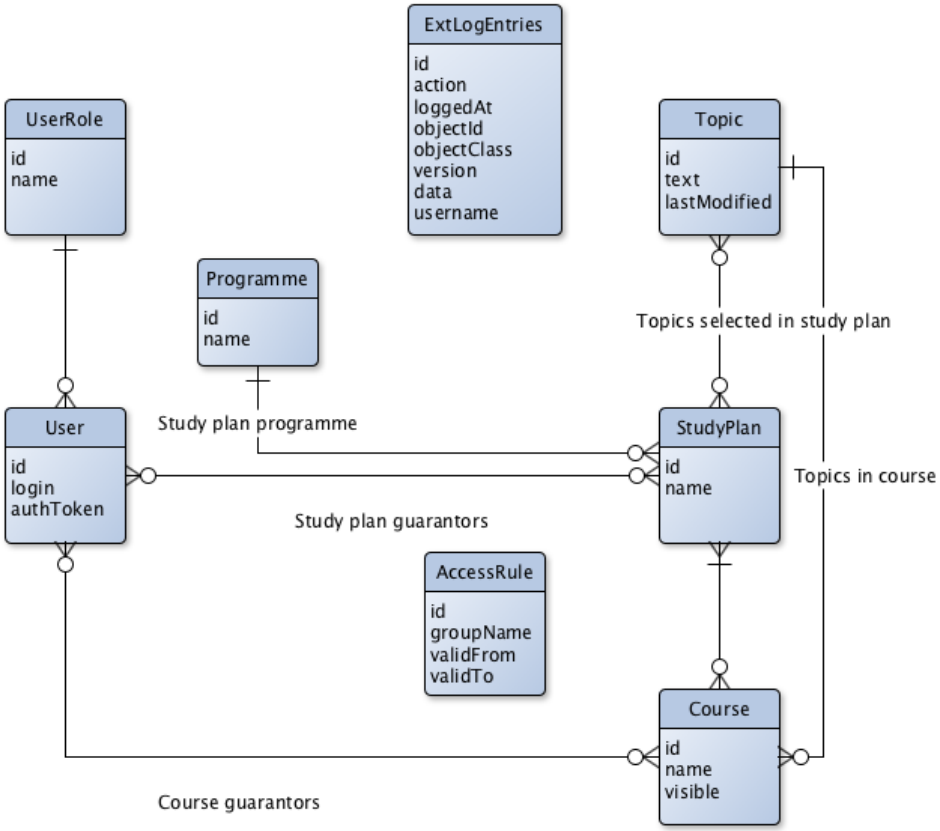
Vlastnosti entity:

1. **id**
2. **name** – textový název role, bude implementován jako výčtový typ

3.1.3 StudyPlan

Studijní plán je ústředním prvkem celého *Modelu*. Má vazby na uživatele (garanty studijního plánu), na předměty, jež pod plán spadají, na tematické okruhy, které mohou být vybrány, a na studijní programy.

[17] Entity-relationship model (ERM), viz `https://cs.wikipedia.org/wiki/Entity-relationship_model`



Obrázek 3.1: Entitně vztahový model (Entity-Relationship Model)

Vlastnosti entity:

1. **id**
2. **name** – název studijního plánu

3.1.4 Course

Vyučovány předmět. Obsahuje vazbu na uživatele prostřednictvím garantů předmětů. Také se odkazuje na seznam tematických okruhů.

Vlastnosti entity:

1. **id**
2. **name** – název předmětu
3. **visible** – příznak, zda je předmět viditelný pro studenty

3.1.5 Topic

Konkrétní tematický okruh.

Vlastnosti entity:

1. **id**
2. **text** – aktuální text okruhu
3. **lastModified** – datum poslední úpravy předmětu

3.1.6 ExtLogEntries

Entita obsahující záznamy o změnách jiných, verzovaných entit.

Vlastnosti entity:

1. **id** – identifikátor záznamu
2. **action** – akce provedená nad entitou
3. **loggedAt** – čas provedení akce
4. **objectId** – identifikátor změněné entity
5. **objectClass** – třída změněné entity
6. **version** – verze entity (pořadové číslo)
7. **data** – nově vložená data
8. **username** – uživatel zodpovědný za změnu

3.1.7 Programme

Studijní program.

Vlastnosti entity:

1. **id**
2. **name** – název studijního programu

3.1.8 AccessRule

Časové pravidlo povolující správu skupině uživatelů.

Vlastnosti entity:

1. **id**
2. **groupName** – název skupiny uživatelů
3. **validFrom** – od kdy pravidlo platí
4. **validTo** – do kdy pravidlo platí

3.2 Technologie

V této sekci bych rád popsal některé technologie, které jsem se rozhodl použít během implementace. Jejich komplexnost je poměrně pestrá, od skriptovacích jazyků, přes frameworky, až po malé knihovny v rámci částí aplikací.

3.3 Technologie v backend sekci

3.3.1 PHP

Nejrozšířenější programovací jazyk na straně webových serverů asi není potřeba představovat [7]. Já osobně mám s programováním v PHP dlouhodobé zkušenosti, proto pro mě byla volba jazyka vcelku jednoduchá.

3.3.2 Symfony

Symfony je balíček samostatných PHP komponent a zároveň webový MVC framework [18]. Obsahuje komponenty mimojiné zajišťující:

- Snadné generování HTML kódu pomocí šablonovacího systému **Twig**
- Podoporu pro snadné vytváření zabezpečených formulářů
- Autentifikaci a autorizaci uživatelů
- Integraci s ORM knihovnou **Doctrine**, viz sekce 3.3.3

Na vývoji Symfony se podílí (nebo v nějaké době podílelo) přes 2000 osob a používá jej přes 300 000 vývojářů [8]. Díky tomu vzniklo a vzniká nepřeberné množství rozšíření usnaďnujících práci vývojářů.

Podobné vlastnosti (byť třeba ne v tak velkém měřítku) nabízejí i jiné frameworky, každý z nich má určité výhody i nevýhody. Pro mě byl ovšem rozhodující fakt, že Symfony je hlavní framework mezi vývojáři na naší fakultě. Výběrem toho frameworku zjednoduším případným následovníkům práci na rozšiřování či úpravách mé aplikace.

3.3.3 Doctrine

Doctrine je sada PHP knihoven poskytujících funkčnost ukládání a získávání dat z databází. Jejími stěžejními částmi jsou **Object Relational Mapper**[19] a **Database Abstraction Layer**[20].[9]

[18] `https://en.wikipedia.org/wiki/Software_framework`

[19] zajišťuje konverzi mezi objekty v programovacím jazyce a tabulkami v relační databázi

[20] vrstva, která pro zjednodušení práce programátorů zakrývá konkrétní rozdíly v implementaci různých databází

Framework Symfony má zabudovanou podporu pro Doctrine snad v každé své části. Dohromady tvoří opravdu silnou kombinaci a dokáží ušetřit spoustu rutinní práce spojené s persistencí dat.

3.3.4 FOSRestBundle

Symfony nabízí velké množství různých *bundlů*, neboli balíčků rozšiřujících specifické části frameworku. FOSRestBundle[21] je určen přímo pro backend API aplikace. Poskytuje například:

- Vrstvu umožňující psaní Controllerů[22]
- Vlastní routovací pravidla[23] pro automatické generování URL podle REST konvencí
- Kontrola a zpracování požadavků s HTTP *Accept*[24] hlavičkou
- Parsování těla HTTP požadavku
- Controllery pro zpracování výjimek

3.3.5 NelmioApiDocBundle

Jelikož dokumentace je základní kámen každé API, je nutné, aby každá metoda byla dobře a přehledně popsaná. Ruční psaní by bylo hodně zdlouhavé, naštestí existuje velké množství aplikací, které se o to za nás postarají, stačí správně anotovat metody.

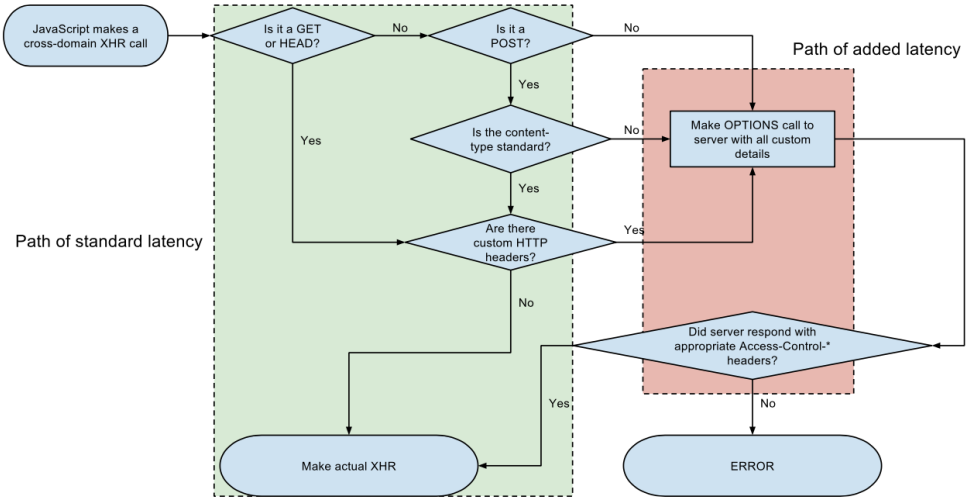
Jednou z takových aplikací je NelmioApiDocBundle, který dokáže pracovat s Doctrine entitami a přehledně zobrazí všechny metody se vstupními a výstupními parametry. Umí také zobrazovat, či schovávat jednotlivé třídní proměnné na základě toho, do jaké výstupní skupiny patří. Například v metodě `/courses` mi budou stačit nějaké základní údaje o předmětu, kdežto u `/courses/MI-SWE/detail` očekávám podrobnější informace.

Příjemný bonus je schopnost bundlu přímo posílat požadavky se zvolenými parametry a zobrazovat odpověď serveru.

3.3.6 NelmioCorsBundle

Při běžném AJAXovém volání je všemi moderními prohlížeči striktně dodržována zásada **same-origin policy**. To znamená, že skript spuštěný z jednoho

[21] https://github.com/FriendsOfSymfony/FOSRestBundle
[22] viz Model-View-Controller architektura
[23] pravidla určující propojení mezi URL a metodou controller objektu
[24] `https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html`



Obrázek 3.2: Fungování Cross-Origin Resource Sharing mechanismu, zdroj: Wikipedia.org [10]

zdroje v prohlížeči nemůže přistupovat k datům z jiného zdroje[25]. Je to logické zabezpečení proti provádění potenciálně nebezpečných dotazů na pozadí webového prohlížeče, o kterých by uživatel vůbec nemusel vědět.

Ovšem občas se mohou takové dotazy hodit, například pokud chcete volat z frontend aplikace API metody běžící na jiné doméně, což je přesně náš případ. Právě k tomu slouží mechanismus **CORS**[26], jehož fungování je popsáno obrázkem 3.2.

NelmioCorsBundle se o celou komunikaci s klientem postará za nás. Stačí v konfiguraci bundlu nastavit, které domény, HTTP hlavičky a metody jsou povoleny.

Ještě malá poznámka na závěr – nikde není psáno, že by frontend aplikace a API nemohly běžet na stejné doméně. Nicméně jsem nechtěl při nasazování nikoho omezit, a proto jsem počítal se složitější variantou už od začátku implementace. Obě aplikace mi běžely na zcela oddělených (lokálních) doménách.

3.3.7 LiipFunctionalTestBundle

Na testování v Symfony existuje opět velká řada balíčků, jak interních, tak od programátorů třetích stran. Zvolil jsem funkční testy s balíčkem LiipFuncti-

[25] zdroj je definován jako kombinace protokolu, domény a portu

[26] Cross-Origin Resource Sharing viz `https://en.wikipedia.org/wiki/Cross-origin_resource_sharing`

onalTestBundle[27], který poskytuje kromě jiného zabudovanou podporu pro Doctrine fixtures. Více o tomto tématu v sekci *Testování* 6.

3.4 Technologie ve frontend sekci

3.4.1 JavaScript

JavaScript je odlehčený programovací jazyk s funkcemi první třídy (*first-class functions*[28]). Nejčastěji se používá jako skriptovací jazyk pro webové stránky, ačkoliv jej využívají i jiná prostředí (například populární **Node.js**), která již nejsou přímo spjatá s webovým prohlížečem [11]. JavaScript:

- je založený na prototypech
- je dynamický jazyk
- podporuje objektově orientované programování
- podporuje imperativní programování
- podporuje funkcionální programování

Volba JavaScriptu jako hlavního skriptovacího jazyka ve frontend části opět nebyla těžká. Je to momentálně nejpoužívanější skriptovací jazyk na webu s nepřeberným množstvím různých nadstavbových jazyků, frameworků a knihoven, které rozšiřují funkčnost nebo opravují některé nedostatky Javascriptu.

3.4.2 React versus Angular

Jak jsem již naznačil, možnost snadné integrace frontendové aplikace na EDUX (případně jinam) byla od začátku hlavní prioritou. Složitá adresářová struktura HTML stránek to může v systému, který má pevně nastavené vlastní generování URL adres hodně ztížit. Proto jsem se rozhodl naplno využít možností JavaScriptu a celou frontend aplikaci napsat jako jednu HTML stránku a pomocí AJAXových volání a manipulace **DOM stromu** [29] přepínat mezi virtuálními stránkami.

V samotném JavaScriptu by to byl poměrně zdlouhavý úkol, který by pravděpodobně vyústil ve vytvoření vlastního frameworku, což by bylo určitě zajímavé, ale mimo téma této práce. Rozhodl jsem se tedy vybrat si jeden ze zavedených JavaScriptových frameworků, které jsou na takovéto úkoly stavěné a ušetří mi spoustu rutinní práce. Jelikož jsem doposud neměl s žádným takovým zkušenosti, vybíral jsem mezi dvěma nejpopulárnějšími současnosti – **Angularem** a **Reactem**.

[27] `https://github.com/liip/LiipFunctionalTestBundle`
[28] `https://en.wikipedia.org/wiki/First-class_function`
[29] `http://www.w3.org/DOM/`

3.4.2.1 React

React je JavaScriptová knihovna od vývojářů Facebooku a Instagramu, určená pro tvorbu uživatelských rozhraní. Často je vnímána jako *View* (pohledová) složka *Model-View-Controller* architektury. React cílí na velké aplikace, jejichž data se často mění [12].

3.4.2.2 Angular

I **Angular** je JavaScriptový framework, ale také podporuje další jazyky jako TypeScript či Dart[30]. Podporuje nejen objektové datové struktury s provázáním dat (*data-binding*), ale také funkcionální programování s neměnnými datovými strukturami (*immutables*) [13].

3.4.2.3 Srovnání

Pokusil jsem se srovnat výhody obou frameworků pro naši aplikaci. Částečně jsem vycházel ze svých pozorování a částečně z kvalitně napsaného článku, který vyšel na serveru AirPair.com [14].

Výhody Reactu:

- strmá křivka učení
- vysoký výkon nezávislý na složitosti modelu (v porovnání s Angularem)
- dobrá úroveň abstrakce – pro používání není potřeba rozumět vnitřnímu fungování frameworku

Výhody Angularu:

- komplexní framework, bez nutnosti používat dodatečné knihovny
- pokročilejší šablonovací systém, využívající HTML 5 atributy a vlastní direktivy
- primárně stavěný pro jednostránkové aplikace

3.4.2.4 Výsledná volba

Nedá se říci, že by jeden z frameworků byl objektivně lepší nebo horší, ale pro účely této aplikace mi přišel vhodnější **Angular**. Je to hlavně kvůli tomu, že obsahuje vše potřebné na jednom místě a naše aplikace je přímo modelovým příkladem použití Angularu. Jak to tak bývá, za mocnější nástroje se často platí větší složitostí softwaru, nicméně v případě Angularu mi to přišlo jako rozumná cena za nabízený produkt.

[30] `https://www.dartlang.org/`

Zvolil jsem si verzi frameworku **2.0**, ačkoliv v době psaní byla ještě ve fázi beta testování. Oproti Angularu 1 totiž přináší mimo jiné významné zlepšení výkonu, jednodušší syntax a sofistikované routovací komponenty.

3.4.3 Typescript

Typescript je typovaná nadmnožina JavaScriptu, vyvíjená Microsoftem, která se kompiluje do čistého JavaScriptu [15]. Některá rozšíření oproti JavaScriptu:

- anotace typů a, kde je to možné, také inferování typu podle kontextu
- třídy
- rozhraní
- generika
- moduly a namespacy
- řešení některých nepříjemných vlastností JavaScriptu, jako kontext funkcí (klíčové slovo *this*), viditelnost proměnných

Kde je to možné, řídí se specifikací **ECMAScript 2015**[31] (případně novějšími drafty), pro zajištění dopředné kompatibility s budoucími standardy.

Ačkoliv jsem doposud neměl s TypeScriptem zkušenosti, výše popsané vlastnosti pro mě byly dostatečnou motivací, abych tento jazyk vyzkoušel. Dalším důvodem je symbióza s Angularem, který je primárně vyvíjený pro tento jazyk. Přirovnal bych to ke kombinaci *Doctrine* + *Symfony*. Obojí jsou užitečné nástroje, ale teprve ve spojení dokážou svoji skutečnou sílu.

3.5 Uživatelské role

Na základě požadavků na výslednou aplikaci (viz kapitola 2) jsem navrhl následující seznam uživatelských rolí a práva a úkoly s nimi spojené.

- Student
- Garant předmětu
- Garant studijního plánu
- Administrátor

[31] `http://www.ecma-international.org/ecma-262/6.0/#sec-terms-and-definitions-prototype`

Dlouho jsem zvažoval nad strukturou *program – obor – zaměření – studijní plán*. Ukázat celou strukturu všem uživatelům? Nechat je proklikat si přes obor a zaměření ke studijním plánům, nebo rovnou zobrazit studijní plány a jejich obory? Rozhodl jsem se pro druhou možnost. Z hlediska tematických okruhů jsou opravdu důležité pouze studijní plány a odstínění oborů a zaměření přinese velké zjednodušení aplikace bez ztráty funkčnosti. Navíc se tím vyřeší problém, že některé obory obsahují zaměření, a jiné rovnou studijní plány.

Z těchto důvodů v aplikaci nenalezneme garanty oborů či zaměření, pouze **garanty studijních plánů**.

3.5.1 Student

Cílová skupina celého systému. Studenti mají právo zobrazit tematické okruhy jednotlivých předmětů podle studijního oboru.

3.5.2 Garant předmětu

Garant předmětu může vytvářet, editovat a mazat tematické okruhy pro svůj předmět. Výsledný seznam témat není viditelný pro studenty, dokud jej neschválí *garant studijního plánu*. Každá úprava tematického okruhu je zaznamenána ve verzovacím systému a je tedy možné dohledat přesnou historii každé otázky. Dále může přiřazovat a odebírat práva na správu předmětu (tedy přidávat další garanty předmětu). Garant by neměl mít možnost spravovat práva pro svůj účet.

3.5.3 Garant studijního plánu

Garant studijního plánu zodpovídá za finální podobu všech tematických okruhů pro svůj plán. Ze seznamu okruhů vytvořených *garanty předmětů* vybere, které budou součástí oboru. Není přitom omezen počtem okruhů, které musí schválit. Může zvolit všechny, stejně tak nemusí vybrat z daného předmětu ani jeden okruh. Garant studijního plánu má také možnost sám upravit tematické okruhy a celkově spravovat předměty. Jedinou podmínkou je, že předmět musí spadat do studijního plánu, jehož je garantem. Tento uživatel má také možnost přidávat roli garanta svého studijního plánu pro ostatní uživatele (stejně jako mohl *garant předmětu* pro svůj předmět). V neposlední řadě může aktualizovat svůj studijní plán importem z *KOS*u, což aktualizuje seznam předmětů, aniž by byla odstraněna stávající témata a vazby.

3.5.4 Administrátor

Uživatel, který má neomezená práva. Může tedy provádět vše, co garant studijního plánu, ale pro všechny obory. Smí importovat libovolný studijní plán (na rozdíl od garanta *SP*, který smí importovat pouze svůj) z *KOS*u a následně mu přiřadit garanty. Má na starost vytváření časových přístupových

práv, které určují, kdy mohou garanti předmětů a studijních plánů spravovat sekce jim přidělené.

3.6 Seznam akcí

V tomto seznamu jsou vypsány nejčastější úkoly a akce, které uživatelé mohou v rámci aplikace provádět. Jsou rozděleny podle jednotlivých uživatelských rolí. Díky hierarchii rolí mohou výše postavené role provádět veškeré akce, které mají povoleny jejich předchůdci.

3.6.1 Student

- zobrazení seznamu programů
- zobrazení seznamu studijních plánů
- zobrazení schválených okruhů v rámci studijního plánu
- zobrazení seznamu všech tematických okruhů v předmětu

3.6.2 Garant předmětu

- přidání garanta předmětu
- odebrání garanta předmětu
- přidání tematického okruhu
- editace tematického okruhu (čili vytvoření nové verze *TO* ve verzovacím systému)
- smazání tematického okruhu

3.6.3 Garant studijního plánu

- přidání garanta studijního plánu
- odebrání garanta studijního plánu
- zařazení tematického okruhu do studijního plánu
- odebrání okruhu ze studijního plánu
- import svého studijního plánu z *KOS*u
- smazání svého studijního plánu

3.6.4 Administrátor

- import libovolného studijního plánu z *KOS*u
- správa garantů libovolného studijního plánu
- nastavování časových pravidel pro správu přístupů

3.7 Návrh uživatelského rozhraní

Cílem uživatelského rozhraní frontendové aplikace je hlavně co nejjednodušším způsobem ukázat všechny funkce backendové API. Nesnažil jsem se udělat graficky dokonalou aplikaci, ale jednoduše a přehledně předvést možnosti Angular frameworku.

Ačkoliv výsledný vzhled nebyl tím nejdůležitějším, rozhodl jsem se vyrobit **low-fi**[32] model. Je užitečný mimo jiné z těchto důvodů:

- poskytne komplexní představu o všech potřebných prvcích
- pomůže rozvrhnout rozložení komponent na jedné stránce
- odhalí některé chyby aplikační logiky
- pomůže určit rozdělení na jednotlivé (virtuální) stránky

V této kapitole popíšu postup při tvorbě a jednotlivé stránky prototypu. Návrh stránek je (pokud nenapíšu jinak) pro situaci, kdy je přihlášen uživatel s nejvyšším oprávněním (tj. administrátor). Aplikace totiž bude některé prvky zobrazovat, či schovávat podle toho, jestli uživatel má dostatečná oprávnění k provedení akce spojené s daným prvkem. Není to z důvodu bezpečnostního (o to se musí postarat API), ale z důvodu přívětivého uživatelského rozhraní.

3.7.1 Balsamiq Mockups

Pro tvorbu prototypů jsem použil nástroj **Balsamiq Mockups**[33] v jeho třicetidenní zkušební době. Tato šikovná aplikace mi vyhovuje svým intuitivním ovládáním, velkým množstvím prototypů komponent (iOS tlačítka, stromové navigace, imitace obrázkových sliderů…), ale hlavně snadnou konfigurací. Každý prvek má něco jako uživatelsky přívětivý zdrojový kód. A kdyby náhodou uživatelské prostředí nevyhovovalo, je možné rovnou upravit relativně jednoduchý XML kód souboru.

[32] low-fidelity, prototyp grafického rozhraní snadný na výrobu, většinou neinteraktivní

[33] `https://balsamiq.com/products/mockups/`

Obrázek 3.3: Lo-fi prototyp – návrh hlavní stránky frontend aplikace

3.7.2 Domovská stránka – výpis programů

Jak můžeme vidět na obrázku 3.3, hlavní stránka je opravdu jednoduchá. Obsahuje výpis programů, tak jak jsou zaneseny v KOSu. Momentálně to jsou bakalářské a magisterské programy i s jejich anglickými verzemi. Dále obsahuje prvky tvořící základní layout[34] stránek, který je společný pro všechny další návrhy.

3.7.2.1 Navigace

Navigační pruh je umístěn podle zavedených konvencí do nejhornější části stránky. Je rozdělen do dvou částí:

- **V levé části** se nachází odkaz na hlavní stránku pro rychlou navigaci. Dále zde najdeme informace o přihlášeném uživateli a odkazy na akce, které může pod svým účtem vykonat. Obsah těchto odkazů se může lišit podle oprávnění přihlášeného uživatele.
- **Napravo** je umístěna sekce pro výběr jazyka aplikace.

3.7.2.2 Název aplikace

Nic jiného, než nadpis „Tematické okruhy“ (samozřejmě dle aktuálního jazyka), aby návštěvník vždy věděl, že se nachází v aplikaci na správu / zobrazení tematických okruhů.

3.7.2.3 Historie

Pět posledních položek historie procházení, v pořadí od nejstarší po aktuální stránku. Vzhledem k hloubce struktury stránek se zdá být pět položek do-

[34] rozvržení celé stránky, struktura, která se příliš nemění s obsahem konkrétních stránek

stačujících, nicméně toto číslo by mělo být možné snadno změnit. Všechny položky kromě aktuální jsou odkazem na navštívenou stránku.

3.7.2.4 Obsahová část

Po výše zmíněných komponentách následuje prostor pro obsahovou část stránky. Ta je samozřejmě pokaždé jiná, nicméně vždy obsahuje v horní části velký nadpis pro snadnou orientaci.

3.7.3 Detail programu

Tato sekce je velmi podobná stránce 3.7.2. Navigace, název aplikace i historie pochopitelně zůstávají jen místo seznamu programů je seznam studijních plánů, pro aktuální program.

3.7.4 Detail studijního plánu

Detail studijního plánu je stěžejní částí této aplikace. Já, jako student zaměření *Webové a softwarové inženýrství, zaměření Webové inženýrství*, chci hlavně co nejdříve vidět, které okruhy se týkají mých státních závěrečných zkoušek. Stejně tak správci předmětů / studijních plánů by rádi měli všechno na jednom místě. Z toho důvodu jsem se snažil umístit velké množství komponent (u kterých to dávalo smysl) právě na tuto stránku.

3.7.4.1 Nástroje

Na prototypu 3.4 lze vidět rozložení této sekce pro **privilegované uživatele**. Pod názvem studijního plánu (který může být opravdu dlouhý) vidíme *toolbox*[35] ve kterém se zobrazí garantům studijního plánu a administrátorům (viz 3.5) odkazy sloužící ke správě. Mohou přidávat / odebírat garanty, provést import z KOSu nebo studijní plán smazat.

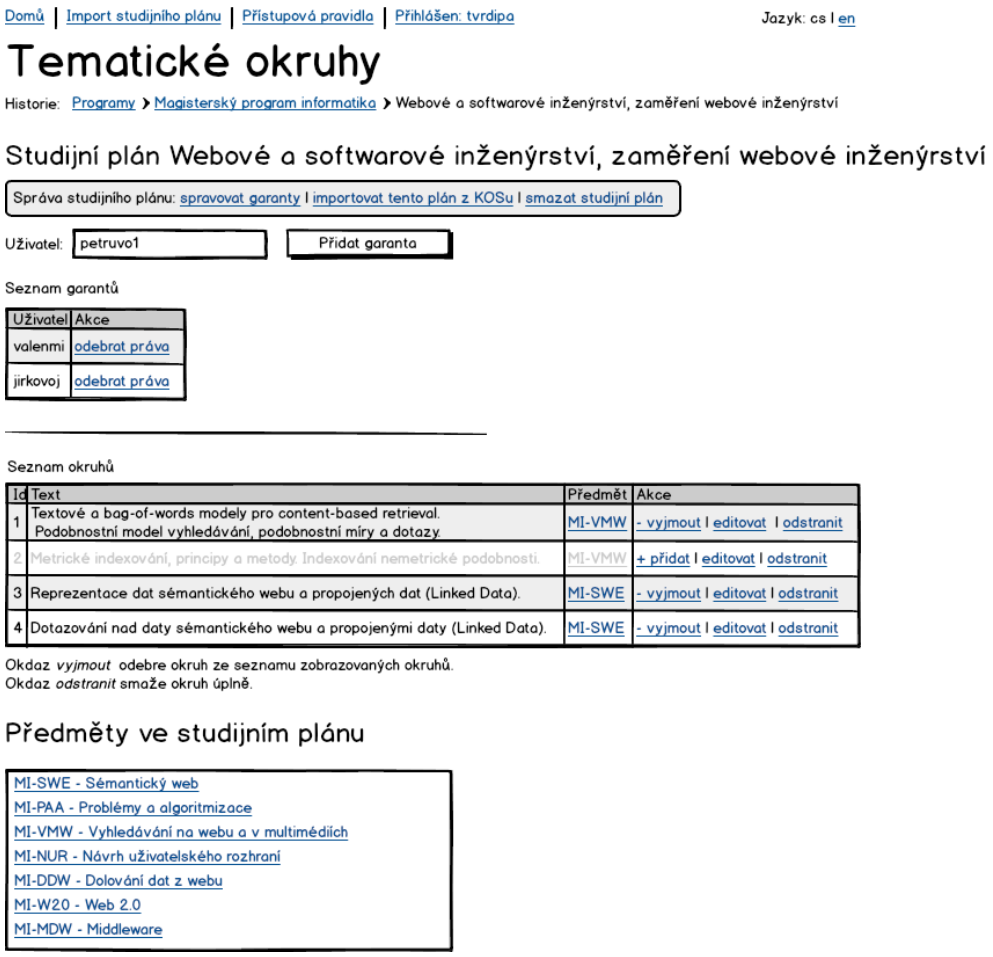
3.7.4.2 Správa garantů

Komponenta pro vkládání a odebírání garantů je ve výchozím stavu schovaná a rozbalí se po kliknutí na odkaz **„spravovat garanty“**. Tuto variantu jsem zvolil, protože komponenta není příliš složitá ani prostorově náročná, tudíž jsem se držel plánu nechat ji v detailu studijního plánu. Zároveň ale většinu času nebude používaná, takže není potřeba, aby byla neustále vidět.

3.7.4.3 Výpis okruhů

Výpis okruhů je asi nejdůležitější částí celé aplikace. Kromě samotného textu okruhu obsahuje tabulka také identifikátor, odkaz na předmět, pod který spadá, a akce, které lze s daným okruhem provádět. Mezi akce patří:

[35] oblast s nástroji pro správu aktuální komponenty



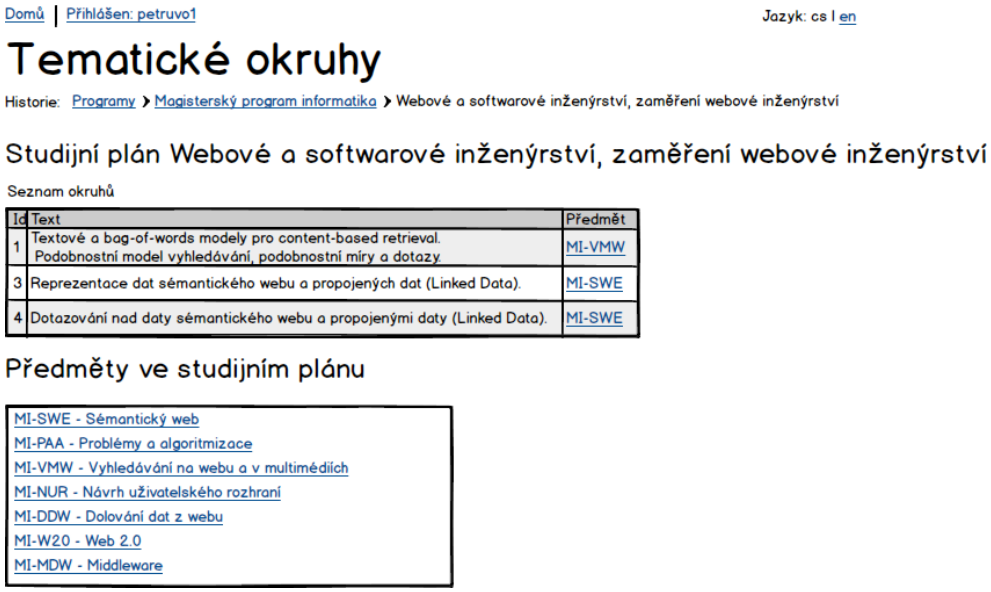
Obrázek 3.4: Lo-fi prototyp – návrh detailu studijního plánu

- **vyjmutí** tématu ze seznamu okruhů pro tento studijní plán
- **editace** tématu na samostatné stránce
- úplné **odstranění** tématu

Aby nevznikaly pochyby, přidal jsem pod tabulku vysvětlení rozdílu mezi *vyjmutím* a *odstraněním* tématu.

3.7.4.4 Předměty ve studijním plánu

Poslední částí na stránce je výpis předmětů ve studijním plánu. Ten slouží jednak jako rozcestník, kterým se uživatel dostane na detail předmětu 3.7.5, ale také jako informace, které předměty tento plán obsahuje (ať už mají nějaké okruhy či ne).



Obrázek 3.5: Lo-fi prototyp – návrh detailu studijního plánu, s přihlášeným uživatelem, který nemá práva administrace

3.7.4.5 Detail studijního plánu pro studenty

Na prototypu 3.5 jsem chtěl ukázat, jak vypadá jedna stránka (detail studijního plánu), pokud je přihlášen uživatel mající práva pouze na čtení. Jak je vidět, stránka je výrazně jednodušší. Uživatelské rozhraní se liší například v těchto bodech:

- v navigaci chybí odkazy na administrátorské úkony a sekce
- kompletně chybí toolbox se správou studijního plánu
- ve výpisu okruhů jsou pouze schválené okruhy
- chybí odkazy na správu okruhů

3.7.5 Detail předmětu

Návrh detailu předmětu 3.6 obsahuje opět nadpis předmětu, následovaný toolboxem pro aktuální předmět. V toolboxu nalezneme:

- **správu garantů** – obdobná komponenta jako v detailu studijního plánu 3.7.4.2, zde se pouze spravují garanti předmětů místo garantů studijních plánů
- **přidání okruhu** – odkaz na samostatnou stránku s formulářem

Obrázek 3.6: Lo-fi prototyp – návrh detailu předmětu

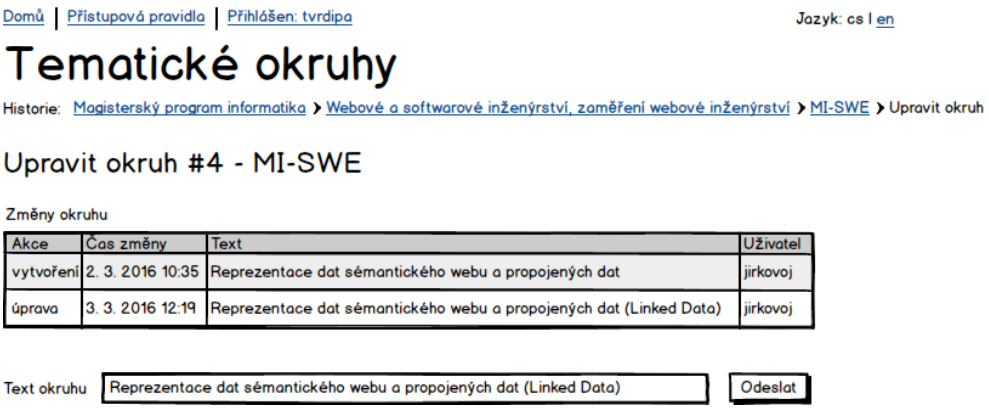
- **kopírování okruhů** – rozbalí, či sbalí komponentu sloužící ke kopírování okruhů z jiného předmětu, tato komponenta je výchozím stavu sbalená

Pod toolboxem se nachází seznam okruhů ne nepodobný tomu v detailu studijního plánu 3.4. Pouze chybí nástroje k přidávání / vyjímání okruhů, tato akce totiž dává smysl pouze u studijních plánů.

3.7.6 Editace okruhu

Po kliknutí na odkaz „editovat“ u výpisu okruhů (ať už na stránce studijního plánu či detailu předmětu) se uživatel dostane na samostatnou stránku, kde může okruh upravovat. Prototyp této stránky lze vidět na obrázku 3.7. Dělí se na dvě části – samotný formulář a **historii úprav okruhu**, což je tabulka obsahující tyto položky:

- **typ akce** – co bylo s tématem provedeno, možné hodnoty jsou:
  - vytvoření
  - úprava
  - smazání
- **čas změny** – datum a čas provedení úpravy
- **text** – nový text okruhu
- **uživatel** – login uživatele zodpovědného za změnu



Obrázek 3.7: Lo-fi prototyp – návrh editace okruhu

Kapitola 4

Realizace backend části

V následující kapitole bych rád popsal výsledné API a jeho dokumentaci. Rozeberu také některá zajímavá místa aplikace a části důležité pro případné úpravy či rozšíření backendu.

4.1 Dokumentace

Pro dokumentaci bylo použito NelmioApiDocBundlu. Toto rozšíření je velmi snadné na používání, stačí správně anotovat **action** metody controllerů, což jsou zároveň metody našeho API. Použití anotace lze vidět na následujícím příkladu:

```
1     /**
2      * Returns single study plan
3      *
4      * @Get(requirements={
5      *     "id": "[A-Za-z\-]*(\.\d+)?"
6      * })
7      * @ApiDoc(
8      * requirements={
9      *      {
10     *          "name"="id",
11     *          "dataType"="string",
12     *          "description"="StudyPlan id"
13     *      }
14     *  },
15     *  statusCodes={
16     *          200="Returned when successful",
17     *          404="Returned when study plan is not found"
18     *     },
19     * output={
20     *      "class" = "AppBundle\Entity\StudyPlan",
21     *      "parsers"={"Nelmio\ApiDocBundle\Parser\
          JmsMetadataParser"},
```

```
22      *       "groups" = {"course_basic", "studyplan_detail", "
            programme_basic"}})
23      */
24     public function getStudyplanAction($id)
25     {
26     /* ... */
27     }
```

Jako příklad jsem záměrně zvolil metodu pro získání studijního plánu, protože je trochu specifická. Pod popisem metody je vidět **@Get** anotace[36] FOSRestBundlu, která specifikuje formát identifikátoru studijního plánu. Plány importované z KOSu totiž mohou mít v názvu tečku (například *MI-WSI-WI.2014*) a adresu `studyplans/MI-WSI-WI.2014` Symfony interpretuje jako žádost o studijní plán *MI-WSI-WI* ve formátu *2014*, což je pochopitelně špatně.

V **@ApiDoc** anotaci se nejčastěji objevují následující položky:

- **requirements** – vstupní požadavky, například id studijního plánu
- **statusCodes** – všechny možné návratové HTTP kódy
- **input** – formát vstupního objektu posílaného v těle HTTP požadavku
- **output** – formát výstupního objektu

Položky *input* a *output* potřebují ke generování dokumentace nějaké entity aplikace, které umožňují serializaci[37]. NelmioApiDocBundle nabízí několik možností serializace, já jsem zvolil **JMS Serializer**[38]. Anotacemi popíšu jednotlivé třídní proměnné entit (ať už Doctrine nebo vlastních) a přiřadím jim skupiny podle toho, v jakých situacích mají být viditelné. Například skupina *course_basic* vypíše pouze základní vlastnosti předmětu, jako identifikátor a název. Tyto skupiny nastavím nejen v anotaci pro dokumentaci, ale také v samotné odpovědi controlleru:

```
1 $view->setSerializationContext(SerializationContext::create()
      ->setGroups(array('course_basic', 'studyplan_detail', '
      programme_basic')));
```

Vygenerovaná dokumentace se nachází na adrese `{backend-url}/api/doc`, náhled je možné vidět na obrázku 4.1.

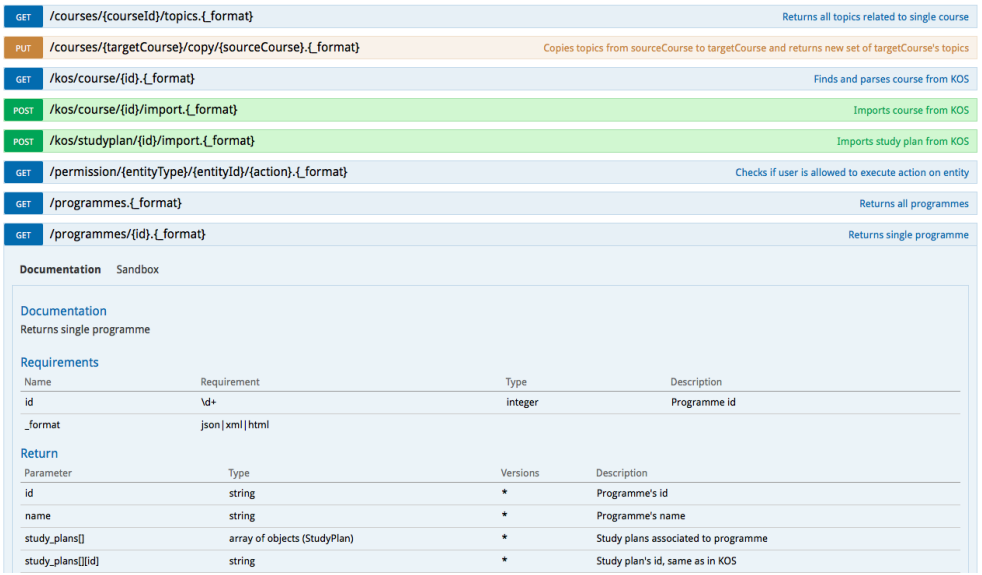
4.2 Import z KOSu

Ze zdrojů, které poskytuje KOSapi, jsou pro naši aplikaci důležité hlavně základní infromace o předmětech a studijních plánech (a některé přidružené

[36] `http://symfony.com/doc/master/bundles/FOSRestBundle/annotations-reference.html#route`

[37] `https://en.wikipedia.org/wiki/Serialization`

[38] `http://jmsyst.com/libs/serializer`



Obrázek 4.1: Náhled vygenerované dokumentace API

entity, například skupiny předmětů). To všechno jsou informace, které lze veřejně získat po autorizaci pomocí OAuth 2.0 protokolu.

Založil jsem si pomocí **AppsManageru**[39] vlastní aplikaci typu **Service Account**. Protože nepotřebuji přistupovat k datům uživatele, pouze se dotazovat KOSapi prostřednictvím své aplikace, je tento typ naprosto dostačující.

Po zaregistrování aplikace jsem získal **client ID** a **client secret**. Pak už jen stačilo v *AppsManageru* povolit přístup k veřejným částem KOSapi a získat autorizační token, například takovýmto cURL[40] dotazem:

```
curl -v --user {client_id}:{client_secret} --data grant_type=
    "client_credentials" https://auth.fit.cvut.cz/oauth/oauth/
    token
```

S tímto časově omezeným tokenem poslaným v HTTP *Authorization* hlavičce (typ *Bearer*), už dostaneme správná data z KOSapi. Přesně takto jsem to udělal v modelové třídě **KOSReader** (`src/AppBundle/Model/KOSReader.php`). Pro vykonávání dotazů jsem používal CiRestClientBundle[41], což je šikovná nadstavba nad PHP CURL[42] knihovnou. Ve výjimečných případech, kdy CiRestClientBundle nesplňoval moje požadavky jsem použil přímo zmíněnou PHP knihovnu.

Třída KOSReader je zodpovědná za získávání entit studijních plánů (případně předmětů) z KOSapi a k tomu jí pomáhá třída **XMLToEntityPar-**

[39] `https://auth.fit.cvut.cz/manager/`
[40] `https://curl.haxx.se/docs/manpage.html`
[41] `https://github.com/CircleOfNice/CiRestClientBundle`
[42] `https://curl.haxx.se/docs/manpage.html`

ser, která získanou XML odezvu přetransformuje do Doctrine entit. Využívá k tomu Symfony DomCrawler komponentu[43].

Entity získané pomocí KOSReaderu stačí už jen uložit do databáze. O to se postarají jednotlivé modelové třídy, například **StudyPlanManager** ve své metodě *importStudyPlan*.

4.2.1 Konfigurace

Konfigurace této části aplikace je opravdu jednoduchá. V Symfony konfiguraci, v poli *parameters*, je možné nastavit pět parametrů týkajících se KOSapi. Parametry se nachází v souboru *src/AppBundle/Resources/config/parameters.yml*, nicméně není problém je přesunout do kteréhokoliv konfiguračního souboru načítaného Symfony. Konfigurace obsahuje následující položky:

- **apiEndpoint** – koncový bod KOSapi, například *https://kosapi.fit.cvut.cz/api/3*
- **tokenEndpoint** – adresa pro získání přístupového tokenu, například *https://auth.fit.cvut.cz/oauth/oauth/token*
- **clientId** – identifikátor klientské aplikace
- **clientSecret** – tajemství klientské aplikace
- **grantType** – typ autorizace, například *client credentials*

4.3 Uživatelské role

V kapitole *Analýza* jsem popsal požadované uživatelské role, viz sekce 3.5. V rámci implementace jsem vytvořil **UserRole** entitu, která je třídní proměnnou entity **User**. Uživatel má přidělenu právě jednu roli z dvojice **user**, **admin**.

Jak je vidět, ve výčtu chybí role *garant studijního plánu* a *garant předmětu*. Je to z toho důvodu, že tyto role nemá smysl explicitně přiřazovat. To, že je uživatel garantem, mu nedává žádná privilegia mimo jím garantovaný předmět (či studijní plán). Správa oprávnění se v takových případech řeší jinak, viz sekce 4.5.

4.4 Přihlašování uživatelů

Nad přihlašováním uživatelů jsem se dlouho zamýšlel. Bohužel však nevím, jakým způsobem se budou uživatelé v nasazené aplikaci autentifikovat. Snažil jsem se to zjistit, ale nedostalo se mi v rozumném čase jasné odpovědi. Možností je opravdu hodně, aplikace může mít přístup k EDUXovému seznamu

[43] `http://symfony.com/doc/current/components/dom_crawler.html`

uživatelů, být přímo připojená na KOS (nebo část jeho databáze), může využívat Shibboleth[44].

Rozhodl jsem pro standardní cestu, která, dle mého názoru, je také nejméně náročná na nasazení:

1. Moje aplikace bude obsahovat v databázi seznam uživatelských jmen a k nim příslušející seznam API klíčů. Tento seznam je možné buď rovnou nahrát do databáze z jiného zdroje, nebo importovat skriptem, za použití metod `UserManager::generateToken()` či `UserManager::addUser($username, $roleName)`.

2. Frontend aplikace bude zabezpečeně (skrze **HTTPS** protokol) komunikovat s backendovou API. Zavoláním metody */users/{username}/token*, získá uživatel autentifikační token. Tato metoda musí ověřit, že uživatel je tím, za koho se vydává, například pomocí Shibbolethu, či dotazem na jinou databázi. Zmíněná metoda momentálně vrací token každému, kdo se dotáže, tudíž je potřeba tuto část při nasazování implementovat.

3. Se získaným tokenem již frontend aplikace nemá problém s API komunikovat.

Pokud by ale bylo nutné používat jinou přihlašovací metodu než tokeny, zásah do stávající aplikace by nebyl nijak dramatický.

4.4.1 Implementace

Základem ověřovacího procesu je výše zmíněná třída **UserManager** (*src/AppBundle/Model/UserManager.php*). Ta implementuje *UserProviderInterface*[45] a díky tomu může být použit v symfony konfiguraci firewallů, jejichž cílem je určit, které adresy musí být zabezpečeny před nepřihlášenými uživateli.

Pravidla zabezpečení jsou poměrně jednoduchá:

```
1   firewalls:
2       dev:
3           pattern: ^/(_(profiler|wdt)|css|images|js)/
4           security: false
5       docs:
6           pattern: ^/api/
7           security: false
8       token:
9           pattern: ^/users/[a-z0-9]*/token
10          security: false
11      secured_area:
12          pattern: ^/
```

[44] http://shibboleth.net

[45] http://symfony.com/doc/current/cookbook/security/custom_provider.html

```
13              stateless: true
14              simple_preauth:
15                  authenticator: apikey_authenticator
16              provider: api_key_user_provider
```

Uživatel musí být přihlášen, když chce přístup do jakékoliv sekce. Výjimky tvoří ladící nástroje Symfony, obrázky, skripty, **dokumentace** a API **metoda pro získání tokenu**, kde uživatel logicky nemůže být ještě přihlášen.

4.5 Správa přístupových práv

Pro autorizaci uživatelů jsem se rozhodl použít **Symfony Votery**. Jsou snadnější na použití než zabudovaný ACL modul[46] a fungují na bázi jednoduchých podmínek [16]. Pokaždé, když je volána metoda `isGranted()` na službě *authorization_checker*, každý Voter hlasuje o tom, zda přihlášený uživatel má mít možnost přistupovat k určitému zdroji, většinou k Doctrine entitě. Symfony má různé strategie rozhodování v případě, že více Voterů hlasuje ohledně jedné entity. Pro účely této aplikace však stačila výchozí strategie, tedy povolení v případě, že alespoň jeden Voter se vyjádřil kladně.

V aplikaci jsou zaregistrovány čtyři Votery:

- **ApplicationVoter** – hlasuje nad abstraktní entitou celé aplikace, například o tom, zda může uživatel importovat libovolný studijní plán, či nastavovat přístupová pravidla.
- **CourseVoter** – hlasuje o zobrazení, editaci a administraci konkrétních předmětů. Pomocí API metod se přidávají a odebírají garanti předmětů, což se projeví ve vztahové tabulce *course_guarantor* a následně je kontrolováno tímto Voterem. Garanti studijních plánů smí provádět všechny operace jako garanti předmětů, pro předměty spadající do jejich plánů.
- **StudyPlanVoter** – podobná funkčnost jako CourseVoter, jen se studijními plány.
- **UserVoter** – zakazuje uživatelům přidávat a odebírat svoje práva.

Kromě těchto omezení ještě existují časově omezená pravidla ve formě **whitelistu**, která nastavuje administrátor. Obsahují:

- **Datum od** – začátek platnosti pravidla.
- **Datum do** – konec platnosti pravidla.
- **Název skupiny** – která skupina uživatelů může v daném časovém intervalu spravovat přidělené zdroje. Možné hodnoty jsou **course_guarantor** a **studyplan_guarantor**.

[46] `http://symfony.com/doc/current/cookbook/security/acl.html`

Na základě zmíněných pravidel se při každém požadavku Voter snaží nalézt alespoň jedno pravidlo povolující přístup pro relevantní skupinu uživatelů v čase požadavku.

Jak to celé funguje si můžeme prohlédnout na útržku následující metody, která rozhoduje nad administrací předmětu:

```
1    private function canAdmin(Course $course, User $user)
2     {
3         //...
4
5         //if there is access rule for current time and study
              plan group,
6         //check if logged user guarantees study plan
              containing this course
7         if ($this->accesRuleRepo->canGroupAccess(
8         AccessRule::GROUP_STUDYPLAN_GUARANTOR)) {
9             foreach ($course->getStudyPlans() as $studyPlan)
                  {
10                if ($studyPlan->getGuarantors()->contains(
                      $user)) {
11                    return true;
12                }
13            }
14        }
15
16        //...
17    }
```

4.6 Výsledná API

Při tvorbě API jsem se snažil dodržovat pravidla návrhu **REST**ful webových služeb[47]. V této sekci popíšu vybrané funkce z celkového počtu **34 veřejně dostupných metod**. Z každé skupiny metod (například správa časových pravidel) jsem většinou vybral jednoho reprezentanta. Detailní popis metod včetně všech vstupních a výstupních parametrů a možných HTTP odpovědí se nachází v online dokumentaci (viz 4.1).

Za každým dotazem je volitelně také přípona, která označuje požadovaný formát výstupu. Možné formáty jsou **.json** a **.xml**, přičemž **.json** je výchozí hodnotou.

4.6.1 POST /accessrules

Metoda POST vloží nové časové pravidlo pro přístup skupiny uživatelů a vrátí vytvořenou entitu, včetně identifikátoru. Tento identifikátor se dále používá při volání metod jako získání jednoho pravidla, či smazání pravidla. Dále

[47] http://www.tutorialspoint.com/restful/restful_introduction.htm

je možné vypsat všechna uložená pravidla. Správa časových pravidel přísluší pouze administrátorům.

4.6.2 GET /courses/{courseId}

Vrátí předmět s id *{courseId}*, včetně tematických okruhů.

4.6.3 DELETE /courses/{courseId}/guarantors/{username}

Odstraní garanta předmětu a vrátí zprávu o úpěchu akce, nebo chybový kód. Uživatel musí k této akci dostatečná práva a nemůže odebrat svoje oprávnění. Existují také metody pro přidávání a výpis všech garantů.

4.6.4 PUT /courses/{targetCourse}/copy/{sourceCourse}

Zkopíruje všechny okruhy z předmětu *{targetCource}* a vloží je do předmětu *{sourceCourse}*. Původní okruhy zůstanou nezměněny. Tato metoda slouží pro garanty předmětů, které jsou odvozeny z předmětu s jiným názvem, ale tematické okruhy se příliš nezměnily.

4.6.5 POST /kos/course/{id}/import

Importuje předmět z KOSu a vrátí základní informace o vloženém předmětu. Metoda skončí chybou, pokud už předmět v databázi existuje. Tato funkce není ve frontendu využívána, nicméně pro testovací účely a pro možné budoucí využití jsem se ji rozhodl zachovat.

4.6.6 POST /kos/studyplan/{id}/import

Importuje studijní plán z KOSu a vrátí nově vloženou entitu. Studijní plán musí spadat pod jeden z přednastavených studijních programů (BI, BIE, MI, MIE). Importováním se načte jméno, program a vazby na předměty. Jelikož tematické okruhy spadají pod předměty, tak pokud importujeme zbrusu nový studijní plán obsahující uložené předměty, **získame rovnou seznam okruhů připravených ke schválení**.

Importovat lze také studijní plán, který už je v databázi uložen. V tom případě se pouze doplní nové vazby na předměty, které do té doby nebyly zaznamenány.

4.6.7 GET /permission/{entityType}/{entityId}/{action}

Tato metoda je velmi důležitá ve frontend aplikaci. Vrací informaci o tom, zda aktuálně přihlášený uživatel může provádět akci nad vybranou entitou. Obsahuje povinné parametry:

- **entityType** – typ entity, například „course“

- **entityId** – unikátní identifikátor entity, například „MI-SWE“
- **action** – požadovaná akce, napříkad „edit“

Tímto způsobem se lze dotázat, jestli uživatel smí editovat předmět *Sémantický web* a zda tedy má smysl mu zobrazovat editační formulář.

4.6.8 GET /programmes/{id}

Vrátí detail studijního programu. Dalšími metodami pro práci s programy jsou například výpis všech programů či výpis studijních plánů přiřazených k programu.

4.6.9 DELETE /studyplans/{id}

Odstraní studijní plán, ale zachová beze změn všechny předměty, které pod něj spadají, jelikož mohou patřit (a většinou patří) i do jiných studijních plánů. Se správou studijních plánů je spojeno mnoho metod, například výpisy předmětů, tematických okruhů, správa garantů a další.

4.6.10 PUT /studyplans/{studyPlanId}/topics/{topicId}/select

Zařadí okruh s id *{topicId}* do seznamu schválených tematických okruhů pro studijní plán *{studPlanId}*. Výstupem je kompletní seznam schválených okruhů po provedení zařazení. Dle očekávání existuje také metoda pro vyjmutí tematického okruhu ze seznamu.

4.6.11 GET /topics/{topicId}/versions

Z metod, které mají na starost správu témat, bych chtěl vypíchnout tuto. Díky ní lze získat přehledně všechny verze okruhu s id *{topicId}*, včetně data úpravy, autora, textu a dalších informací.

4.6.12 GET /users/{username}/token

Vrátí autentifikační token pro uživatele s loginem *{username}*. Při nasazení v produkčním prostředí musí být implementován nějaký způsob ověření uživatele, momentálně metoda vrací token každému, kdo o něj požádá.

4.6.13 GET /whoami

Vrací uživatelské jméno akutálně přihlášeného uživatele.

Kapitola 5

Realizace frontend části

V této kapitole se zaměřím na frontend aplikaci, popíšu některé její stěžejní komponenty a ukážu výsledné screenshoty.

5.1 Routování

Routovací (směrovací) komponenta ve frameworku Angular 2 umožňuje navigaci mezi jednotlivými **View** (pohledovými) složkami aplikace. Drží se podobného modelu jako webové prohlížeče, interpretuje URL jako instrukci pro zobrazení konkrétního *View* a předá mu příslušné parametry z adresy, pro vykreslení požadovaného obsahu [17].

Angular používá routovací komponenty k vytváření odkazů a následně zpracovává události kliknutí. Také umožňuje přesměrování na jiné *View* přímo v uživatelských komponentách. Po přesměrování (které samozřejmě probíhá bez nutnosti znovu načtení celé stránky) se dokonce mění URL adresa. Angular totiž využívá HTML 5 možností pro modifikaci historie prohlížení [48], takže po změně *View* vloží pomocí `history.pushState()` metody novou adresu do historie prohlížeče. Díky tomu lze v adresním řádku nejen vždy přehledně vidět, na jaké stránce se uživatel nachází, ale také používat tlačítka „Zpět“ a „Vpřed“, což považuji za velké plus z hlediska uživatelského rozhraní.

Routovací pravidla nastavuji v hlavní komponentě programu *app/app.component.ts*. Vypadají například takto:

```
1 @RouteConfig([
2     { path: '/', name: 'Home', component:
        ProgrammeListComponent },
3     { path: '/course/:id', name: 'CourseDetail', component:
        CourseDetailComponent },
4     { path: '/course/:courseId/topic/add', name: 'AddTopic',
        component: TopicAddComponent },
5     /*...*/])
```

[48] https://developer.mozilla.org/en-US/docs/Web/API/History_API

Pokud je uživatel přesměrován například na adresu */course/MI-SWE*, do oblasti určené pro router komponentu se vykresí detail předmětu (**CourseDetailComponent**) s předaným identifikátorem předmětu (*MI-SWE*).

5.2 Historie

Historie navštívených stránek je důležitým prvkem uživatelského rozhraní. Jak je vidět na obrázku 5.1, v horní části stránky se nachází odkazy na navštívené stránky a název aktuálního umístění.

O správu historie napříč aplikací se stará služba **NavigationService** (*app/navigation/navigation.service.ts*), která je injektovaná[49] do každé komponenty, která je vykreslována pomocí Angular routeru.

Pro tyto účely jsem vytvořil interface **Link**, obsahující odkaz na **definici routy**, a **text odkazu**. Každá komponenta vykreslená pomocí router komponenty v metodě `ngOnInit()`[50] předá objekt implementující rozhraní Link navigační službě. Ta už se postará o uložení a správné zobrazení seznamu odkazů.

Momentálně se zobrazuje pět posledních navštívených stránek, během testování se to ukázalo jako plně dostačující. Není ovšem problém toto nastavení v *NavigationService*, konkrétně v proměnné `_historySize`, změnit.

5.3 Překlady

Pro překlady jsem využil knihovnu ng2-translate[51]. Její použití je opravdu jednoduché, přesto však poskytuje vše potřebné pro překlad textů. Základem této knihovny je služba **TranslateService**, která se při startu aplikace nakonfiguruje a později je předávána všem komponentám. Konfiguruje se kupříkladu:

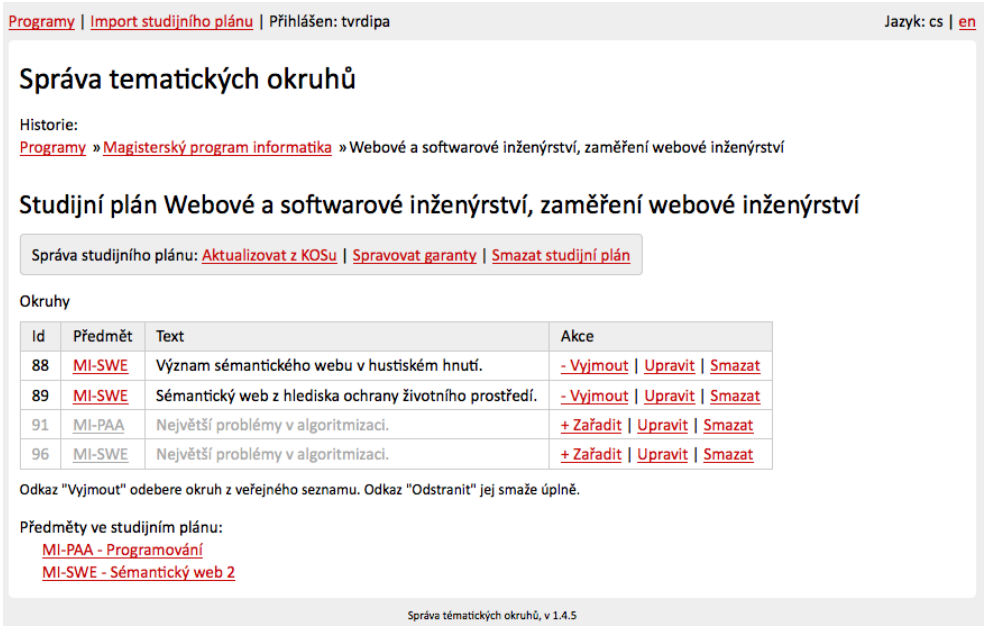
- předpona a přípona překladových souborů, v této aplikaci se nachází v souborech **translations/{kod-jazyka}.json**
- handler, který se zavolá v případě, že překlad nebude nalezen
- výchozí jazyk

Při změně jazyka se volá metoda `TranslateService::use(lang: string)`. Při programování jsem narazil na chybu v knihovně, která způsobuje, že **vždy musí být nejdříve použit výchozí jazyk**, jinak překlady skončí chybou.

[49] viz dependency injection: `https://angular.io/docs/ts/latest/guide/dependency-injection.html`

[50] metoda automaticky spuštěná po inicializaci komponenty

[51] `https://github.com/ocombe/ng2-translate`



Obrázek 5.1: Screenshot frontend aplikace – detail studijního plánu

Překlady se získávají asynchroním voláním výše zmíněné služby, která se dotazuje překladových souborů. Aplikace tudíž není blokovaná po dobu dotazu, byť většinou je doba odezvy velmi malá, protože soubory se nacházejí na stejném serveru jako volající skripty.

Překladové soubory používají jednoduchý .json formát:

```
{
    "appName": "Finals topics administration",
    "link_programmeList": "Programmes",
    "app_loading": "Loading...",
    "programme_title": "Programmes",
    "footer_about": "Topics administration, v {{value}}"
    ...
}
```

Na levé straně je překladový token, napravo pak překládaný text. Lze také použít takzvané **placeholdery**, které budou nahrazeny dynamickou hodnotou, jako například číslem aktuální verze aplikace, jak je možné vidět v předchozím příkladu.

5.4 API služba

Třída **ApiService** (*app/api/api.service.ts* je hlavním propojem frontendu s backendovou API. V třídní proměnné `_apiBase` je odkaz na endpoint API. Při nasazení se tato hodnota musí přepsat na adresu backendového serveru.

Tato třída jako jediná zná strukturu API (názvy funkcí, HTTP metody a potřebné parametry), umí posílat HTTP požadavky a vracet odpovědi. Při prvním volání libovolné metody se získá token pro aktuálně přihlášeného uživatele (momentálně pouze nastavené uživatelské jméno v proměnné `_username`). Požadavek na token se zřetězí[52] s původním dotazem.

API služba sice odesílá vzdálené dotazy, ale veškerá logika typu parsování odpovědi na entity předávané komponentám je na straně jednotlivých služeb *Modelu* (například CourseService).

5.4.1 Zpracování chyb

V podstatě každá metoda API může skončit chybou, ať už je to z důvodu špatné adresy, nesprávného vstupu nebo nedostatečného oprávnění uživatele. Už od začátku jsem chtěl mít správu chyb řešenou na jednom místě. Pro ten účel vznikla služba **error-handler.ts** (*app/api/error-handler.ts*). Do metody `handleError(error: Response)` se předávají výstupy HTTP požadavků, které skončily nějakou chybou.

Metoda nejprve projde všechny názvy chyb (například **CourseNotFoundException**) a pro každou z nich se pokusí najít překlad v aktuálně používaném jazyce. Zavedl jsem konvenci, kde názvy chyb začínají předponou **__error__**. Výstupem této metody je pole přeložených chyb. V případě, že by žádný překlad nedopadl úspěšně, vrátí se text informující o neznámé chybě.

5.5 Flash zprávy

Flash zprávy jsou krátké informace zobrazené uživateli, většinou týkající se výsledku provedené akce. Bývají buď informativního, nebo varovného charakteru.

Pokaždé, když je voláno API, je vrácen buď **HTTP kód 200**, signalizující úspěšné provedení, nebo chybový kód s názvem chyby. V obou případech je ale nutné, aby uživatel byl o výsledku informován. K tomutu účelu slouží třída **FlashService** (*app/flash/flash.service.ts*), která je předávána jako služba do každé komponenty, která chce zprávy zasílat.

Zasílání zpráv funguje přes metodu `addMessage`, s následujícími parametry:

- **text** – text flash zprávy
- **messageType** – typ zprávy, jedna z hodnot výčtovového typu MessageType:
  - MessageType.Success

[52] jedna z vlastností ReactiveJS, hojně využívaného mou aplikací, je možnost řadit za sebe samostatná AJAXová volání, viz https://github.com/Reactive-Extensions/RxJS

Obrázek 5.2: Screenshot frontend aplikace – ukázka flash zpráv

  - MessageType.Error
  - MessageType.Warning

- **translate** – příznak, zda se má text překládat (*true*), nebo už je ve finální formě (*false*)

Jak jsem popsal v sekci 5.4.1, zpráv z API může přijít několik. Stejně tak může uživatel provést najednou několik akcí, a proto se může zobrazit zároveň vícero flash zpráv, například s různým `messageType`, jak je možné vidět na obrázku 5.2. Každá zpráva zůstane zobrazená po deset vteřin, než je odstraněna. Změna *View* nemá na zprávu vliv.

5.6 Výsledné stránky

Při tvorbě uživatelského rozhraní jsem se řídil návrhy v sekci 3.7, s některými drobnými odchylkami.

5.6.1 Celkový vzhled

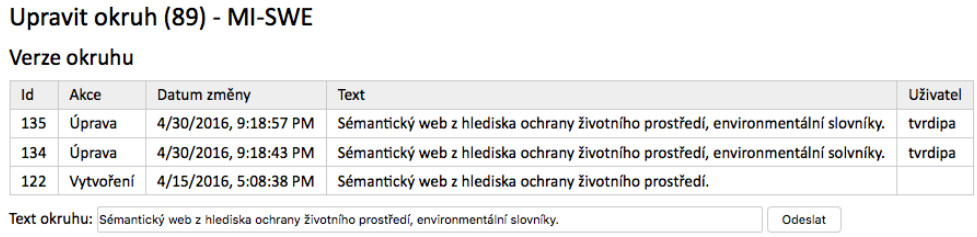
Jak je vidět například na obrázku 5.1, nastavil jsem jiné pozadí pro celou stránku a větší obsahovou část (s nadpisem a historií). Bylo to hlavně z důvodu vizuálního oddělení navigace a patičky od hlavního bloku.

V pravé části horní lišty se v souladu s návrhem nachází toolbox s výběrem jazyka. Všechny jazyky kromě aktuálně používaného jsou odkazy. Protože se načítá celý lokální soubor s předklady při prvním použití, je překlad jednotlivých výrazů téměř okamžitý.

5.6.2 Detail studijního plánu

Detail studijního plánu si lze prohlédnou na obrázku 5.1. Kvůli menší prostorové náročnosti jsem udělal screenshot testovacího studijního plánu, kterému byla vyplněna data „ručně“ během spouštění testovacích skriptů. Ve skutečném studijním plánu je tematických okruhů i zařazených předmětů mnohem více, ovšem na uživatelské rozhraní to nemá velký vliv.

Obrázek 5.3: Screenshot frontend aplikace – ukázka detailu předmětu



Obrázek 5.4: Screenshot frontend aplikace – úprava tematického okruhu

5.6.3 Detail předmětu

V detailu předmětu (obrázek 5.3) můžeme vidět rozbalenou sekci garantů předmětu. Ta se otevře okamžitě po kliknutí na odkaz „Spravovat garanty“ a po druhém kliknutí se opět zabalí. Stejně funguje i v sekci detailu studijního plánu 5.6.2. Aby nevznikala nedorozumnění, přidal jsem text, který vysvětluje, že ne všechny okruhy spadající pod předmět musí být přiřazeny pod studijní plán.

5.6.4 Úprava tematického okruhu

Na obrázku 5.4 můžeme vidět *View* s úpravou tematického okruhu. V horní tabulce se nachází historie úprav okruhu. Okruh byl vytvořen opět během načítání testovacích dat, tudíž není asociován s žádným uživatelským účtem. Při vytváření nových okruhů v aplikaci tato situace nenastává.

Kapitola 6

Testování

Jak jsem zmínil v předchozích kapitolách, testování je základ každé správné API. Testování jsem věnoval velké množství času, který se ovšem vrátil při bezbolestné integraci frontendové a backendové aplikace. V této kapitole rozeberu průběh testování více dopodrobna.

6.1 Funkční versus jednotkové testy

Již od začátku implementace jsem přemýšlel, jakou metodu testování použiji. Zvažoval jsem dvě hlavní možnosti – **funkční testování** nebo **jednotkové testování** (či jejich kombinaci).

Funkční testy testují funkčnost komponent či celého systému. Snaží se hledat odpovědi na otázky typu „Smí uživatel provést danou akci?“ nebo „Funguje tato akce?“ v rámci funkčních požadavků [18]. Z hlediska funkčních testů je aplikace *black box*, tedy černá skříňka, jejíž vnitřní fungování je neznámé.

Oproti tomu **jednotkové testy** se snaží nalézt nejmenší testovatelnou část softwaru (*jednotku*), oddělit ji od zbytku aplikace a určit, zda funguje podle očekávání [19]. Takové testy logicky musí znát vnitřní strukturu aplikace.

Po dlouhé úvaze jsem zvolil **funkční testování**, a to mimojiné z následujících důvodů:

- U klasických aplikací může být nevýhodou funkčního přístupu nemožnost otestovat všechny možné stavy a výstupy aplikace, ke kterým se může uživatel pomocí UI dostat. U API je to o poznání snazší, množina vstupů je výrazně omezena a vždy můžeme snadno určit, zda je výstup korektní či nikoliv.

- Pro mě, jako programátora využívajícího API je důležitější, že mnou volané metody vrací přesně, co mají, než ujištění, že samostatné části aplikace fungují správně.

- V mé aplikaci se většina funkcí točí kolem CRUD[53] operací. Testování ukládání jednotlivých entit (zvlášť, když jsou všechny ukládány přes Doctrine) mi nepřinese tolik informací jako testování celých funkčních částí.

Samozřejmě nic nebrání tomu, aby kromě funkčních testů byly použity i jednotkové. Během vývoje aplikace jsem ovšem nenarazil na situaci, kterou by funkční testy plně nepokryly.

6.2 Test-driven development

Test-driven development je technika psaní softwaru řízená vytvářením testů. Celý proces se skládá z cyklického opakování následujících kroků [20]:

1. Sestavení testů pro následující funkčnost, kterou se chystám naprogramovat.
2. Vytvoření funkčního kódu, který projde testem.
3. Refactoring[54] starého i nového kódu pro zachování správné struktury programu.

Pro test-driven development jsem se rozhodl ze dvou přímočarých důvodů. Prvním z nich je, že každou metodu API jsem chtěl mít otestovanou a touto metodou bych se vyhnul riziku, že na nějaký důležitý test zapomenu. A za druhé, vytváření testů dopředu mě donutilo si včas promyslet strukturu a fungování jednotlivých funkčností, což je u API nesmírně důležité.

6.3 Doctrine fixtures

Fixtures jsou používány k nahrání specifikovéné množiny dat do databáze. Tato data mohou být použita pro testovací účely, či jako iniciální data pro aplikaci [21].

V backendové aplikaci používám fixtures v kombinaci s LiipFunctionalTestBundle (viz sekce 3.3.7). Tento bundle před každým spouštěním testu načte definované fixtures soubory a díky nim vytvoří v databázi požadované tabulky a naplní je daty. Definice fixture vypadá například takto:

```
1 $topic     = new \AppBundle\Entity\Topic();
2 $topic->setCourse($sweCourse);
3 $topic->setText("Význam sémantického webu v husitském hnutí."
    );
4 $topic->setLastModified(new \DateTime());
5 $this->setReference('topic-swe-hussites', $topic);
```

[53] Create, Read, Update a Delete – čtyři nejčastější operace nad databázovými entitami
[54] https://en.wikipedia.org/wiki/Code_refactoring

V tomto příkladu se vytvoří nový okruh, nastaví se jeho parametry a propojí se s předmětem (z toho vyplývá, že data předmětů už v tu chvíli musí být načtena). Důležitá je také metoda `setReference()`, která konkrétní entitu pojmenuje, aby mohla být v jiných fixtures, ale hlavně při testování, načtena. Díky tomu může testování probíhat nezávisle na konkrétních datech (identifikátorech, názvech entit...).

6.4 Struktura testů

Soubory testů se necházejí v adresáři `src/Tests/Controller`. Jedná se o devět testovacích souborů, kde každý má za úkol otestovat jeden z Controllerů. Dále se zde nachází třída **BaseTestCase**, společný rodič pro všechny testovací třídy, který obshahuje některé užitečné funkce:

- Spouští všechny připravené fixtures soubory.
- Umožňuje získat přihlašovací tokeny uživatelů různých rolí (běžný uživatel, garant studijního plánu, administrátor...). To se hodí zejména při testování zabezpečení přístupu ke zdrojům.
- Obsahuje funkci `rowsCount()`, která převede JSON řetězec do pole či objektu (čímž ověří validitu řetězce) a vrátí počet řádků odpovědi. Toho se často využívá při testování, například chceme-li vědět, kolik okruhů vrací funkce před a po smazání jednoho z nich.

6.5 Výsledné testy

Testovací soubory obsahují vícero metod začínajících slovem *test*. Tyto metody většinou ověřují funkčnost jedné API metody a často se v nich opakují následující tři kroky:

1. Nejdříve se pomocí fixtures referencí načtou potřebné entity uložené v databázi.
2. Symfony Client třída[55] následně metodou `request()` simuluje HTTP požadavek na zadanou adresu a vrátí odezvu.
3. Obsah odevzy a HTTP hlavičky se otestují různými **assert** metodami, aby se ověřila správnost odpovědi.

[55] `http://api.symfony.com/2.7/Symfony/Component/BrowserKit/Client.html`

6.5.1 Příklad – smazání studijního plánu

Nemá smysl zde popisovat všechny provedené testy, to by zabralo desítky stránek zřejmě ne zcela záživného čtení. Vybral jsem si jednu metodu – smazání studijního plánu, na které demonstruji postupy při testování.

Nejprve si vytvořím Symfony Client objekt a načtu reference studijního plánu a jednoho jeho předmětu.

```
1         $client    = $this->createClient();
2         $studyPlan = $this->fixtures->getReference('studyplan
              -mi-wse');
3         $courseSwe = $this->fixtures->getReference('course-
              swe');
```

Následně se pokusím smazat studijní plán, přihlášen pod uživatelem, který tuto akci nemá povolenou. Očekávám, že v odpovědi bude token „**AuthorizationFailedException**“ (na popisu chyby tolik nezáleží, ten je pouze informativní pro administrátora) a že status odpovědi bude **403 Forbidden**.

```
1         $crawler = $client->request('DELETE',
2             '/studyplans/'.$studyPlan->getId().'?'.$this->
                  getTeacherTokenString());
3         $content = $client->getResponse()->getContent();
4         $this->assertEquals(
5             403, $client->getResponse()->getStatusCode()
6         );
7         $this->assertContains('AuthorizationFailedException',
              $content);
```

S tokenem garanta studijního plánu očekávám úspěch operace. Tělo odpovědi obsahuje pouze informativní zprávu, tu není nutné testovat.

```
1         $crawler = $client->request('DELETE',
2             '/studyplans/'.$studyPlan->getId().'?'.$this->
                  getSpGuarantorTokenString());
3         $this->assertEquals(
4             200, $client->getResponse()->getStatusCode()
5         );
```

Po opakovém smazání by se měla objevit chyba signalizující chybějící studijní plán.

```
1         $crawler = $client->request('DELETE',
2             '/studyplans/'.$studyPlan->getId().'?'.$this->
                  getSpGuarantorTokenString());
3         $content = $client->getResponse()->getContent();
4         $this->assertEquals(
5             404, $client->getResponse()->getStatusCode()
6         );
7         $this->assertContains('StudyPlanNotFoundException',
              $content);
```

Nakonec zbývá ověřit, že smazáním studijního plánu nedošlo k odstranění přiřazených předmětů. Ověřuji, že dotaz na vybraný předmět vrací jednu položku a že obsahuje všechny nezbytné informace.

```
1         $content = $this->fetchContent('/courses/'.$courseSwe
              ->getId().'?'.$this->getTeacherTokenString(),
2              'GET');
3         $this->assertEquals(1, $this->rowsCount($content));
4         $this->assertContains($courseSwe->getName(), $content
              );
5         $this->assertContains($courseSwe->getId(), $content);
```

6.5.2 Výsledky testů

Pomocí programu **phpunit** bylo celkem spuštěno **45** testovacích metod a ověřeno **305** tvrzení (*assertions*), s následujícím výsledkem:

```
1 .............................................
2
3 Time: 17.88 seconds, Memory: 97.00Mb
4
5 OK (45 tests, 305 assertions)
```

Tečka ve výstupu symbolizuje jednu splněnou testovací metodu. Celkový čas běhu testu relativně kolísá. Je to převážně způsobeno odezvou vzdálené KOSapi během testů importu předmětů a studijních plánů.

Díky zvolené metodě testování mi sice testy žádnou chybu neodhalily (v klasickém smyslu), zato se však velkou měrou podílely na samotném vývoji a analýze jednotlivých funkčností.

Závěr

Podobu výsledných aplikací a plnění cílů bohužel trochu negativně poznamenaly neustále se měnící požadavky. Například přibližně měsíc před termínem odevzdání se po schůzce s Ing. Miroslavem Balíkem, Ph.D objevilo množství nových požadavků, o kterých jsme ani já, ani vedoucí práce Ing. Vojtěch Jirkovský nevěděli a které výrazně ovlivnily výslednou aplikaci. Navzdory tomu byly všechny hlavní požadavky splněny. V této závěrečné kapitole shrnu plnění funkčních i nefunkčních požadavků a podívám se i na možnosti vylepšení.

Splnění funkčních požadavků

Splnění funkčních požadavků 2.6 je popsáno v tabluce 6.1. Požadavek **F11** („Aplikace umožní *administrátorům* nastavovat časově omezená práva pro přístup ostatních skupin uživatelů“) byl splněn pouze částečně. Všechny potřebné metody jsou v API implementovány, otestovány a zdokumentovány. Z časových důvodů však nebylo implementováno uživatelské rozhraní ve frontend aplikaci, které by správu přístupových práv umožňovalo.

Naštestí nastavování pravidel je úkon, který se provádí většinou pouze na začátku semestru, takže případná správa pomocí ručního volání API (například nástrojem *cURL*) není až tak velkou zátěží pro administrátory. Přesto je to nepříjemnost a možnost vylepšení do budoucna.

Splnění nefunkčních požadavků

Jak je možné vidět v tabulce 6.2, všechny nefunkční požadavky byly splněny.

Možnosti vylepšení

Kromě zmíněného formuláře na správu časových pravidel existují i další oblasti, které nabízejí možnost vylepšení či rozšíření, a to hlavně ve frontend

Tabulka 6.1: Tabulka splnění funkčních požadavků 2.6

Funkční požadavek	Splněn
F1	Ano
F2	Ano
F3	Ano
F4	Ano
F5	Ano
F6	Ano
F7	Ano
F8	Ano
F9	Ano
F10	Ano
F11	Částečně
F12	Ano

Tabulka 6.2: Tabulka splnění nefunkčních požadavků 2.7

Nefunkční požadavek	Splněn
N1	Ano
N2	Ano
N3	Ano
N4	Ano
N5	Ano
N6	Ano

aplikaci. Důvod toho je jednoduchý – kvůli časovému presu jsem se v první řadě soustředil na celkovou funkčnost API a na ladění frontendového uživatelského rozhraní už zkrátka nezbylo tolik času.

Možná vylepšení pro budoucí rozvoj aplikace jsou například tato:

- **Zobrazení seznamu studijních plánů možných k importování.** Momentálně administrátor může importovat libovolný studijní plán, nicméně musí znát identifikátor plánu, pod kterým je uložen v KOSu.
- **Možnost sledovat změny vybraného okruhu.** Všechny verze okruhu i s datem úpravy lze zobrazit, nicméně uživatel musí okruh aktivně sledovat. Určitě by nebylo na škodu, kdyby aplikace sama na změnu okruhu upozornila.
- **Drobná vylepšení uživatelského rozhraní frontend aplikace.** Jedná se například o možnost zavřít otevřené flash zprávy, jednoduchá kontrola validity textového pole před odesláním na server, načítací kolečko během AJAXových volání a podobně.

Shrnutí

Cílem této práce bylo vytvoření aplikace, která poskytne vyučujícím nástroje pro snadnou správu tematických okruhů, bude respektovat rozdělení uživatelských zodpovědností a práv a bude snadno integrovatelná do EDUXu. Za těmito účely byly vytvořeny dvě samostatné dílčí aplikace – backend, obsahující veškerou aplikační logiku a poskytující aplikační rozhraní, a frontend, který dává uživatelům možnost skrze jednoduché uživatelské rozhraní s tímto API pracovat. Výhodou tohoto rozdělení je snadná integrovatelnost, flexibilita a rozšiřitelnost.

Z důvodu často se měnících požadavků bohužel nezbyl čas na implementaci některých dílčích funkčností a existuje řada možností k vylepšení. Nicméně jako celek nabízejí obě aplikace všechny požadované nástroje pro pohodlnou správu tematických okruhů a splňují vytyčené cíle.

Literatura

[1] *dokuwiki [DokuWiki]*. [cit. 2016-04-21]. Dostupné z: https://www.dokuwiki.org/dokuwiki

[2] *Main - KOSapi - Projekt KOSapi*. [cit. 2016-04-20]. Dostupné z: https://kosapi.fit.cvut.cz/projects/kosapi/wiki

[3] *Co je Backend / Adaptic*. [cit. 2016-04-20]. Dostupné z: http://www.adaptic.cz/znalosti/slovnicek/backend/

[4] *API - application program interface*. [cit. 2016-04-20]. Dostupné z: http://www.webopedia.com/TERM/A/API.html

[5] *Co je Frontend / Adaptic*. [cit. 2016-04-20]. Dostupné z: http://www.adaptic.cz/znalosti/slovnicek/frontend/

[6] Petrus, V.: *Správa přístupových seznamů pro EDUX*. Květen 2013.

[7] *Usage Statistics and Market Share of Server-side Programming Languages for Websites, April 2016*. [cit. 2016-04-21]. Dostupné z: http://w3techs.com/technologies/overview/programming_language/all

[8] *What is Symfony*. [cit. 2016-04-21]. Dostupné z: http://symfony.com/what-is-symfony

[9] *About – Doctrine Project*. [cit. 2016-04-21]. Dostupné z: http://www.doctrine-project.org/about.html

[10] Flowchart showing Simple and Preflight XHR – Cross-origin resource sharing – Wikipedia, the free encyclopedia. [cit. 2016-05-03]. Dostupné z: https://en.wikipedia.org/wiki/Cross-origin_resource_sharing#/media/File:Flowchart_showing_Simple_and_Preflight_XHR.svg

[11] *JavaScript / MDN*. [cit. 2016-04-22]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[12] *Why React? | React.* [cit. 2016-04-22]. Dostupné z: http://facebook.github.io/react/docs/why-react.html

[13] *One framework. -. Angular 2.* [cit. 2016-04-22]. Dostupné z: https://angular.io

[14] *Angular vs. React - the tie breaker.* [cit. 2016-05-03]. Dostupné z: https://www.airpair.com/angularjs/posts/angular-vs-react-the-tie-breaker

[15] *TypeScript – JavaScript that scales.* [cit. 2016-04-22]. Dostupné z: http://www.typescriptlang.org

[16] *How to Use Voters to Check User Permissions (The Symfony CookBook).* [cit. 2016-04-28]. Dostupné z: http://symfony.com/doc/current/cookbook/security/voters.html

[17] *Routing & Navigation – ts.* [cit. 2016-04-29]. Dostupné z: https://angular.io/docs/ts/latest/guide/router.html

[18] *What is Functional testing (Testing of functions) in software?* [cit. 2016-05-01]. Dostupné z: http://istqbexamcertification.com/what-is-functional-testing-testing-of-functions-in-software/

[19] *Unit Testing.* [cit. 2016-05-01]. Dostupné z: https://msdn.microsoft.com/en-us/library/aa292197(v=vs.71).aspx

[20] *TestDrivenDevelopment.* [cit. 2016-05-02]. Dostupné z: http://martinfowler.com/bliki/TestDrivenDevelopment.html

[21] *DoctrineFixturesBundle (The Symfony Bundles Documentation).* [cit. 2016-05-02]. Dostupné z: http://symfony.com/doc/current/bundles/DoctrineFixturesBundle/index.html

Příloha **A**

Seznam použitých zkratek

TO Tematický okruh

SP Studijní plán

XML Extensible Markup Language

API Application Program Interface

AJAX Asynchronous JavaScript and XML

SZZ Státní závěrečná zkouška

URL Uniform Resource Locator

MB Megabyte

UI User Interface

Příloha B

Instalační příručka

B.1 Backend aplikace

1. Založit projekt na webovém serveru splňujícím požadavky pro spuštění Symfony 2.7[56], Doctrine a dalších technologií zmíněných v sekci 3.2.
2. Zkopírovat obsah složky `src/backend` do kořenového adresáře projektu.
3. V příkazovém řádku spustit v kořenovém adresáři příkaz `composer install`, který nainstaluje všechny potřebné závislosti aplikace. K tomu je potřeba mít nainstalovaný Composer[57].
4. Vytvořit databázi a nastavit parametry připojení v souboru `app/config/parameters.yml`.
5. V kořenovém adresáři spustit příkaz `php app/console doctrine:schema:create`, který vytvoří databázové schéma.
6. (volitelné) Příkazem `php app/console doctrine:fixtures:load` nahrát inicální data do databáze.
7. (volitelné) Pro účely testování je dobré také nastavit samostatnou databázi, parametry připojení se vyplňují v souboru `app/config/config_test.yml`.

B.2 Frontend aplikace

1. Založit projekt na webovém serveru.
2. Zkopírovat obsah složky `src/frontend` do kořenového adresáře projektu.
3. V souboru `app/api/api.service.ts` nastavit třídní proměnnou `_apiBase` na adresu serveru, kde běží backendová aplikace. Tento TypeScriptový

[56] `http://symfony.com/doc/current/reference/requirements.html`
[57] `https://getcomposer.org`

soubor je nutné transpilovat do JavaScriptu. Možností je vícero, já jsem zvolil příkaz `npm start`, který se správným nastavením kromě transpilace i otevře prohlížeč a automaticky synchronizuje všechny změny v TypeScript souborech. K tomu je potřeba balíčkovací manažer npm[58] s podporou pro TypeScript[59].

Tento návod počítá s nasazením frontendu jako samostatné aplikace, ale rozhodně to není podmínkou. Spuštění pod běžící aplikací je v podstatě stejné, pouze s malými úpravami:

1. Obsah tagu *body* ze souboru `index.html` se zkopíruje do běžící aplikace, s tím, že hierarchie načítaných skriptů musí být zachována nebo adekvátně změněna v těle souboru.
2. Nastaví se hlavička `<base href="/">` tak, aby vlastnost `href` odpovídala umístění aplikace vůči kořenovému adresáři webu.
3. (volitelné) Pro stejný vzhled aplikace načíst v hlavičce stránky stylesheet `css/screen.css`.

[58] `https://www.npmjs.com/package/npm`
[59] `https://www.npmjs.com/package/typescript`

Příloha C

Uživatelská příručka

C.1 Zobrazení detailu studijního plánu

1. Na hlavní stránce uživatel klikne na studijní program, pod který plán spadá (například *MI - Magisterský program informatika*).

2. Na stránce detailu programu klikne na název požadovaného studijního plánu (například *Webové a softwarové inženýrství, zaměření webové inženýrství*)

C.2 Zobrazení detailu předmětu

1. Uživatel zobrazí detail studijního plánu C.1.

2. V detailu studijního plánu (viz obrázek 5.1) vybere ze seznamu předmětů ten, který ho zajímá (například *MI-PAA*).

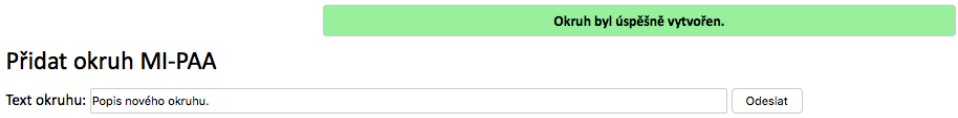
C.3 Vložení okruhu

Týká se *garantů předmětu*, *garantů studijních plánů* a *administrátorů*. Vložení okruhu se provádí vždy na stránce konkrétního předmětu.

1. Uživatel zobrazí detail předmětu, pod který okruh spadá C.2.

2. Na stránce detailu předmětu (obrázek 5.3) v toolboxu „Správa předmětu“ klikne uživatel na odkaz „Přidat okruh“

3. Na samostatné stránce už stačí v texotvém poli vyplnit text okruhu a odeslat požadavek, jak lze vidět na obrázku C.1.



Obrázek C.1: Screenshot frontend aplikace – přidání tematického okruhu

C.4 Úprava a smazání okruhu

Týká se *garantů předmětu*, *garantů studijních plánů* a *administrátorů*.

1. Uživatel zobrazí detail předmětu, pod který okruh spadá C.2.
2. Na stránce detailu předmětu (obrázek 5.3) ve spodní sekci „Okruhy“ najde okruh, který ho zajímá.
3. Podle požadované akce klikne na odkaz „Upravit“ či „Smazat“, který se nachází na stejném řádku jako vybraný okruh.
   a) V případě úpravy okruhu bude uživatel přesměrován na samostatnou stránku s formulářem podobným jako v sekci C.3.

C.5 Správa garantů předmětu

Týká se *garantů předmětu*, *garantů studijních plánů* a *administrátorů*.

1. Uživatel zobrazí detail předmětu C.2.
2. V toolboxu „Správa předmětu“ klikne na odkaz „Spravovat garanty“.
3. Rozbalí se oblast s formulářem pro přidání a seznam aktuálně přidaných garantů.
   a) Pro přidání uživatele vyplní přihlašovací jméno nového garanta do políčka „Login“ a odešle žádost.
   b) Pro smazání uživatele klikne na odkaz „Smazat“ vedle přihlašovacího jména uživatele v tabulce.

C.6 Správa garantů studijních plánů

Týká se *garantů studijních plánů* a *administrátorů*. Garani studijních plánů se spravují stejně jako garanti předmětů, jen na stránce detailu studijního plánu C.1.

Obrázek C.2: Screenshot frontend aplikace – kopírování okruhů z jiného předmětu

C.7 Kopírování okruhů z jiného předmětu

Týká se *garantů předmětu*, *garantů studijních plánů* a *administrátorů*.

1. Uživatel zobrazí detail předmětu C.2.
2. V toolboxu „Správa předmětu“ klikne na odkaz „Kopírování okruhů z jiného předmětu“.
3. Na stránce kopírování okruhů (viz obrázek C.2) uživatel zadá identifikátor předmětu, z kterého ho se okruhy budou kopírovat, a odešle formulář.

C.8 Aktualizace konkrétního studijního plánu

Týká se *garantů studijních plánů* a *administrátorů*.

1. Uživatel zobrazí detail studijního plánu C.1.
2. V toolboxu „Správa předmětu“ klikne na odkaz „Aktualizovat z KOSu“. Tím se načtou data o vazbách na předměty a okamžitě zobrazí.

C.9 Import libovolného studijního plánu

Týká se pouze *administrátorů*.

1. Uživatel klikne na odkaz „Import studijního plánu“ v pravé části horní navigační lišty.
2. Na samostatné stránce (viz obrázek C.3) se mu zobrazí textové pole.
3. Do tohoto pole zadá název studijního plánu (musí být shodný s KOSem) a odešle formulář.

Obrázek C.3: Screenshot frontend aplikace – import libovolného studijního plánu

C.10 Změna jazyka aplikace

1. Uživatel klikne na zkratku jazyka (jiného, než je aktuálně používaný) v pravé části horní navigační lišty.

Příloha D

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `doc`
  - `API_documentation.html` ..... vygenerovaná offline dokumentace
- `src`
  - `frontend` ..... zdrojové kódy frontend aplikace
  - `backend` ..... zdrojové kódy backend aplikace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF