ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Webový designer adaptivních výukových materiálů
Student:	Bc. Pavel Jirásek
Vedoucí:	Ing. Martin Balík, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2015/16

Pokyny pro vypracování

1. Seznamte se s projektem klientské části adaptivního systému (dodá vedoucí).

2. Navrhněte WYSIWYG prostředí kompatibilní s klientskou částí aplikace. Zaměřte se na maximálně intuitivní a snadno použitelné prostředí. V aplikaci bude možné navrhnout nový kurz ale i upravit stávající, včetně ověření jeho konzistence. Datové zdroje bude možné využívat lokální i sdílené na internetu.

3. Navrhněte vhodnou metodiku úpravy aplikace v případě změny doménových objektů kurzu.

4. Editor implementujte jako webovou aplikaci. Použijte frameworky Spring, Hibernate a Primefaces. Jako základ aplikace použijte Adaptive System Framework, který může být v rámci práce rozšířen. Na základě návodu v projektu zabývajícím se tvorbou uživatelského rozhraní (dodá vedoucí), implementujte vhodné prvky uživatelského rozhraní jako znovupoužitelné komponenty rozšiřující framework Primefaces.

5. Aplikaci i jednotlivé komponenty otestujte. Zaměřte se na použitelnost aplikace a přenositelnot základních komponent.

Seznam odborné literatury

Craig Walls: Spring in Action, Third Edition, Manning Publications Co. Greenwich, CT, USA, 2011

Christian Bauer, Gavin King: Java Persistence with Hibernate, Manning Publications Co. Greenwich, CT, USA, 2006

Oleg Varaksin, Mert Caliskan: PrimeFaces Cookbook, Packt Publishing Ltd, 2013

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof.Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 7. ledna 2015

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

Diplomová práce

Webový designer adaptivních výukových materiálů

Bc. Pavel Jirásek

Vedoucí práce: Ing. Martin Balík, Ph.D.

9. května 2016

Poděkování

Chtěl bych poděkovat především panu Ing. Martinu Balíku, za cenné rady, odbornou pomoc a pevné nervy při tvorbě této diplomové práce. Také bych chtěl poděkovat rodině a přátelům, za jejich podporu během mých studií.

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

Jirásek, Pavel. *Webový designer adaptivních výukových materiálů.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Cílem této diplomové práce je vytvoření webového editoru pro tvorbu a úpravu adaptivních výukových materiálů, které se využívají na ČVUT. Tento webový editor umožní vyučujícím snadné úpravy a vytváření materiálů, určených pro aplikaci *Adaptivní kurz Javy*, ze které tato práce vychází. Práce je rozdělena na čtyři stěžejní části. První část práce slouží k seznámení se s adaptivními hypermediálními systémy (AHS) a procesem adaptace jako takovým. Druhou část práce tvoří analýza a následný návrh webového editoru pro adaptivní výukové materiály, dle stanovených požadavků a případů užití aplikace. Ve třetí části je již detailně popsaná samotná implementace webového editoru, včetně nastavení jednotlivých částí aplikace. V poslední čtvrté části je webový editor testován v rámci jeho funkčnosti a uživatelského rozhraní.

Klíčová slova Adaptivita, Adaptivní hypermediální systémy, editor

Abstract

The goal of this master thesis is to create web based editor for creation and management of adaptive learning materials, which are used at ČVUT. This editor will allow teachers to easily create and edit materials for application Adaptivní kurz Javy on which the thesis is based on. The thesis consists of for

main parts. First part of thesis is to explain what exactly is adaptive hypermedia system and process of adaptation. Second part is analysis and design of web editor for adaptive learning materials, based on declared demands and use cases of application. In third part is in detail described, how the editor was implemented and how coresponding frameworks and parts of application were set. In last fourth part the application is tested for its functionality and correctness of user interface.

Keywords Adaptivity, Adaptive Hypermedia Systems, editor

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Cílem každého vzdělávacího systému by měla být efektivní výuka. Jedním z klíčů k efektivnějšímu vzdělávání je právě přizpůsobení obsahu uživateli. Dříve se používali pouze hypermediální aplikace, které nezohledňovali znalosti a potřeby uživatele. Dnes, díky neustálému vývoji ICT, jsou postupně zaváděny tzv. Adaptivní hypermediální systémy (AHS), které umožňují přizpůsobit obsah učiva právě konkrétnímu uživateli. Tento proces je možný díky sledování uživatele, kde dochází k ukládání informací o jeho preferencích, stylu řazení apod. a následně je obsah přizpůsoben konkrétnímu uživateli, podle dat, které o něm AHS nasbíral.
Díky AHS [1] dochází ke třídění informací dle potřeb konkrétního uživatele. To znamená, že se uživatel dostane ke správným informacím, ve správném pořadí a tak jim pravděpodobně i správně porozumí. Toto efektivní třídění (dle správného pořadí a správně rozvržené struktury) umožňuje efektivní vzdělávání. Materiálům, které tyto systémy využívají říkáme "adaptivní výukové materiály". Ty budou podrobněji rozebrány v následujících kapitolách, společně s metodikami jejich adaptace.
Obecně platí, že je vhodné pro práci s adaptivními výukovými materiály využívat specializovaný program, který práci s těmito dokumenty značně ulehčuje. A to je i jeden z důvodů, proč vznikla tato diplomová práce. Vznikla za účelem vytvoření takového webového designeru, který bude mít přehledné a intuitivní prostředí a tak bude vyučujícím zjednodušovat tvorbu a úpravu těchto výukových materiálů.

Struktura práce

Diplomová práce je rozdělena do čtyř stěžejních částí (kapitol): část teoretická, návrh a analýza, realizace a testování.

Popis problematiky - jedná se o teoretickou část, která vychází z nastudované literatury a podrobněji zkoumá problematiku adaptivních hypermediálních systémů, jejich způsoby adaptace, model uživatele a způsoby, kterými lze získat od uživatele data, která se využijí pro další adaptaci.

Návrh a analýza - kapitola obsahuje popis požadavků na výsledný editor a jeho součástí, případy užití a výsledný návrh řešení webového editoru

Realizace - detailně popisuje samotnou implementaci webového editoru, včetně nastavení jednotlivých technologií použitých při jeho tvorbě.

Testování – závěrečná kapitola je zaměřená na testování vytvořeného webového editoru z hlediska jeho funkčnosti a uživatelského rozhraní

Kapitola 1

Popis problematiky

První část diplomové práce slouží jako náhled na téma „adaptivních hypermediálních systémů“. Obsahuje základní pojmy, které jsou nezbytné pro orientaci a pochopení celé práce.

1.1 Adaptivní hypermediální systémy

Adaptivní hypermediální systém (AHS) [1] je systém, který na základě informací o uživateli, který ho právě používá, dokáže zobrazovaný obsah nějakým způsobem upravit tak, aby více vyhovoval potřebám aktuálního uživatele. Informace o uživateli, které se typicky používají pro adaptaci obsahu jsou např. jeho preference, zkušenosti v dané oblasti informací a cíle, souhrnně označované jako model uživatele. Tyto informace mohou být statické, tedy získané v jednom okamžiku pomocí např. dotazníku vyplněného uživatelem nebo získávané dynamicky systémem, který monitoruje uživatelovi akce. Informace, které mohou být získány pozorováním uživatelových akcí, jsou např. jeho preference nějakého typu informací, nebo zvyšující se znalosti uživatele např. díky splnění nějakých testů.

1.2 Historický vývoj adaptivních hypermediálních systémů

S adaptivními hypermediálními systémy se setkáváme již na začátku 90. let,kdy se začaly již plně projevovat nedostatky klasických hypertextů v aplikacích a hledalo se nové efektivnější řešení, které by umožnilo přizpůsobení pro individuálního uživatele. Za přelomový rok ve výzkumu AHS je považován rok 1996, kdy na tuto problematiku vznikají jak nejrůznější konference, tak i o dva roky později kniha od Brusilovského, Kobsa a Vassileva s názvem Adaptive Hypertext and Hypermedia. Obecně se v roce 1996 neprohlubuje jen zájem o danou problematiku, ale mění se i technologická stránka AHS. Adaptivní

hypermediální systémy zde již nejsou implementovány jen jako klasické aplikace, ale implementují se již prostřednictvím webových technologií. Společně s rozvojem webových technologií zároveň dochází i k rozvoji AHS a zájem o tuto problematiku přetrvává dodnes. Důkazem stálého zájmu o adaptivní hypermediální systémy můžou být například mezinárodní konference Adaptive Hypermedia and Adaptive Web-Based Systems, které se pořádají každoročně a jsou zaměřené především na aplikace AHS, adaptační techniky a obecně lepší personalizaci webu i filtrování informací pro uživatele.

1.3 Předmět adaptačního procesu

Možnosti adaptivních hypermediálních systémů jsou rozsáhlé, ale dají se rozdělit do tří základních oblastí:

Přizpůsobení grafického rozhraní aplikace - Přizpůsobení systému podle určitých zkušeností nebo fyzických dispozic uživatele, používá se pro rozlišování zkušeného expertního uživatele systému a uživatele začátečníka, nebo pro úpravu GUI pro handicapované uživatele. Může spočívat např. v skrytí některých částí GUI, které jsou pro uživatele začátečníka nepodstatné nebo je s největší pravděpodobností nebude používat. Díky tomu je GUI přehlednější a pro tohoto uživatele použivatelnější. Pro handicapované uživatele může adaptace spočívat v úpravě velikosti textu či zvýšení hlasitosti zvuku.

Adaptace formy obsahu - Přizpůsobení obsahu podle uživatelovi znalosti v nějaké oblasti, které se zobrazovaný obsah týká. Používá se zejména v adaptivních výukových systémech (AES). Spočívá v zobrazování upravených informací z dané oblasti podle uživatelových znalostí, rychlosti a stylu učení. Díky tomu přistupuje uživatel ke správným informacím ve správném pořadí, což zefektivňuje celý výukový proces.

Adaptace informačního obsahu - Používá se, pokud chceme uživateli zobrazit pouze část z velkého množství dat, která ho s největší pravděpodobností bude zajímat. Tato adaptace je využita v adaptivních výukových systémech, kde se zobrazují pouze informace, které se týkají daného tématu.

1.4 Model uživatele

Jako model uživatele může obecně sloužit jakákoliv informace o uživateli, která se dá použít pro jeho bližší popis. Následuje souhrn nejčastěji používaných charakteristik uživatele.

Znalosti uživatele - Znalost uživatele v určité oblasti zobrazovaných informací je jedna z nejdůležitějších charakteristik uživatele, používaných v

modelu uživatele. U adaptivních výukových systémů to bývá často jediná charakteristika používaná pro adaptivitu. Jde o charakteristiku uživatele, která se s časem může měnit oběma směry. Uživatel se může nové věci naučit, ale také zapomenout staré. AHS musí být tedy schopen tyto změny rozpoznat a adaptovat obsah.

Zájmy uživatele - Zájmy uživatele je důležitá charakteristika uživatele používaná zejména v systémech určených pro vyhledávání nebo filtrování velkého množství informací. Dá se však používat i v AES a jiných systémech, kde se uživateli zobrazují informace podle jeho zájmů.

Zkušenosti uživatele - Jak moc je uživatel zvyklý na práci na internetu a jak moc je schopný se na něm orientovat. Tato charakteristika se dá využít např. pro podporu navigace pro méně zkušené uživatele.

Pozadí znalostí uživatele - Tato charakteristika obsahuje souhrn znalostí a zkušeností uživatele. Jde např. o profesní zaměření uživatele, jeho jazykové znalosti či znalosti AHS.

Preference uživatele - Charakteristiky jako oblíbená barva či typ písma se může projevit v adaptaci jako nastavení pozadí a obecně vzhledu stránky, což může pozitivně ovlivnit práci uživatele se systémem.

1.4.1 Techniky modelování uživatele

Hlavním důvodem, proč se vlastně adaptivní hypermediální systémy používají, je vytvořit dokument, který bude pro uživatele vhodný. K tomu je potřeba znát o uživateli nějaké informace, tedy znát a vytvořit jeho uživatelský model.

Stereotypní modelování - Stereotypní modelování je jednodušší metoda, která neumožňuje individuální adaptaci pro konkrétního uživatele. Uživatelé jsou rozděleni podle podobnosti s určitým stereotypem do skupin a systém přizpůsobuje obsah podle těchto skupin. Hlavním nedostatkem tohoto přístupu je rozdělení uživatelů do skupin, tedy příliš hrubá adaptace. Nelze se přizpůsobovat podle konkrétních potřeb konkrétního uživatele. Na druhou stranu pokud nemáme dostatek informací o uživatelích (např. při čerstvém startu systému), je tento přístup výhodou. Další problém, který s sebou tento přístup nese, je přesnost stereotypů podle, kterých se rozděluje. Stereotyp může být natolik specializovaný, že v každé skupině bude jeden uživatel a pak tyto skupiny postrádají smysl nebo se může stát, že se uživatel nedostane do žádné z těchto skupin.

Překryvné modelování - Překryvné modelování předpokládá kopii doménového modelu pro každého uživatele a je hojně využívané ve adaptivních výukových systémech. Adaptace se většinou odvíjí od stavu uživatelových znalostí v dané oblasti. Pro každého uživatele systém ukládá

informace o tom, jaké znalosti uživatel při práci se systémem nabyl a podle toho také přizpůsobuje zobrazovaný obsah. Na rozdíl od předchozího statického přístupu toto modelování umožňuje adaptaci pro každého konkrétního uživatele. Nevýhodou může být nutnost ukládání informací o každém uživatelovi.

1.5 Adaptační techniky

Adaptace navigace (Adaptive navigation support), rovněž označována jako Adaptace úrovně odkazu (Link Level Adaptation)

Adaptace prezentace (Adaptive presentation), rovněž označována jako Adaptace úrovně obsahu (Content Level Adaptation)

1.5.1 Adaptace navigace

Přímé vedení (Direct Guidance) – uživatel je směrován adaptivním systémem, tj. jsou pro něj vybírány nejvhodnější koncepty na základě aktuálního stavu uživatelského modelu. Bývá nejčastěji realizována pomocí tlačítka „Další“, kdy je uživatel odkázán na nevhodnější příští stránku.

Adaptivní řazení odkazů (Adaptive link sorting) – narozdíl od přímého vedení se nevybírá jeden odkaz, ale systém seřadí všechny odkazy na stránce podle relevance pro konkrétního uživatele. Relevance se vypočítá podle charakteristik uživatele v uživatelském modelu.

Adaptivní skrývání odkazů (Adaptive link hiding) – technika spočívá ve skrývání odkazů, které nejsou pro uživatele v danou chvíli relevantní

- Skrývání – odkaz zůstává na původním místě, jen je mu nastaven takový vzhled, že splývá s okolním textem
- Deaktivace – odkaz je viditelný avšak znepřístupněný
- Odebrání – odkaz je zcela odebrán ze stránky

Adaptivní anotace odkazů – technika, která je jistým rozšířením skrývání odkazů. Odkaz je obohacen o další informace pro uživatele, které mu poskytnou jakýsi kontext k tomu, kde se uživatel nachází nebo jak jsou na tom jeho současné znalosti. Toto obohacení může být ve formě textu, ikony či barevného odlišení odkazu. V AHS se často používá technika zvaná „semafor“. Odkaz je obohacen o ikonu, která svojí barvou ukazuje (podobně jako u semaforu), zda je uživatel schopen obsahu odkazu porozumět v závislosti na jeho uživatelském modelu. Tedy zelená barva označuje obsah, který by měl uživatel snadno pochopit, červená označuje obsah nevhodný pro uživatele na základě jeho znalostí.

Adaptivní generování odkazů (Adaptive link generation)– adaptivní systém hledá souvislosti mezi jednotlivými koncepty a na základě nich dynamicky generuje nové koncepty.

Adaptace mapy stránek (Map adaptation) – adaptivní systém dynamicky vytváří mapu domény v grafické podobě na základě aktuálního stavu modelu uživatele.

1.5.2 Adaptace prezentace

Technika adaptace prezentace umožňuje přizpůsobování obsahu dokumentu v závislosti na uživateli podle jeho uživatelského modelu. Pokud by se vzala pouze jedna z charakteristik uživatele např. jeho zkušenosti, systém bude pro expertního uživatele zobrazovat i funkce nebo obecně obsah, který by uživatel začátečník nedokázal s největší pravděpodobností vůbec využít. V souvislosti s hypermediálními systémy se nedá bavit pouze o textových dokumentech, ale obecně o dokumentech, které mohou obsahovat mnohem více typů multimediálních objektů a proto je třeba rozlišovat adaptaci textu a adaptivní prezentaci multimédií. Každý z těchto objektů bude nazýván jako fragment dokumentu a bude s ním manipulováno při adaptaci prezentace dokumentu.

Vkládání/odebírání fragmentů (Inserting/removing fragments) – nerelevantní fragmenty stránky z pohledu uživatelského modelu nebudou vůbec zobrazovány

Alternativní fragmenty (Altering fragments) – dokument může nabízet několik variant zobrazení daného fragmentu a podle uživatelského modelu se vybere ten nejvhodnější

Rozšiřování textu (Stretch text) – vhodné textové fragmenty jsou rozbalené nebo zabalené, vše ovlivňuje stav uživatelského modelu

Řazení fragmentů (Sorting fragments) – fragmenty jsou pro uživatele seřazeny podle jejich vhodnosti vzhledem k zvoleným částem uživatelskému modelu

Potlačení/Zastínění fragmentů (Dimming fragments) – obdoba techniky odebírání fragmentů, fragment nevhodný pro uživatele ale není kompletně odebrán, nýbrž je zobrazen méně výrazným způsobem

1.6 Současný stav editorů v AHES

Poskytnutí nástroje pro editaci a tvorbu adaptivních materiálů pro adaptivní výukový systém je velice důležité. I když je možné vytvářet tyto materiály bez specializovaného nástroje, bez něj je tento úkol pro uživatele velice obtížný.

Vzhledem k tomu, že adaptivní materiály mohou obsahovat množství informací pro adaptaci, může být pro uživatele složité si pamatovat všechny vazby a závislosti mezi dokumenty, případně udržet správný tvar metadat. Specializované nástroje umožňují uživateli složité operace jako kontrola správnosti obsahu, metadat či struktury dokumentu apod. Díky tomuto přístupu může uživatel vše nechat na nástroji a soustředit se hlavně na tvorbu obsahu výukových materiálů.

1.6.1 Modular Adaptive Learning System (MALS)

[2]Tento systém používá materiály v SCORM formátu. Editor tohoto systému spočívá v spojování jednotlivých fragmentů do jednoho dokumentu. Všechny fragmenty jsou uloženy v LOR repozitáři, odkud mohou učitelé pomocí jednoduchého GUI vybírat fragmenty a určovat vazby mezi nimi, případně pro koho jsou určeny. Tento způsob editace dokumentů je často používán a v některých systémech je dokonce zautomatizovaný, kde samotný systém pozná z uživatelského modelu, jaké fragmenty jsou pro konkrétního uživatele vhodné.

1.6.2 Blackboard

[3]Samostatná aplikace speciálně zaměřená na vytváření a úpravu výukových materiálů. Nejdříve musí uživatel vytvořit strukturu kurzu. Tím se myslí oblasti kurzu, plány lekcí a složky. Pro adaptaci je nutné také vytvořit metadata a povolit programu vygenerovat potřebná data používaná adaptivními systémy pro udržování statistiky o případných uživatelích. Metadata umožňují import a export do a z aplikací, které jsou založené na IMS (Instructional Management Systems) standardu.

1.6.3 NetCoach

[4]Systém umožňující tvorbu adaptivních kurzů bez potřeby znalosti programování. Neumožňuje však adaptaci prezentace textu ani adaptaci navigace. Podle autorů by měl mít student plnou kontrolou nad navigací a přístupem k obsahu a adaptivní systém by měl pouze poskytovat doporučení a nápovědu pro studenta, tedy nenutit studenta jít předvytvořenou cestou, ale umožňuje mu se zaměřovat na části, které ho zajímají a poskytnout mu materiály, které by mu pomohly s pochopením tématu.

1.6.4 MyET a AIMS

[5]Dva systémy používající "mapping paradigm"jako hlavní strukturu pro organizaci terminologie domény a vazeb k položkám kurzů. Uživatel může vkládat 4 typy dat: zvuk, video, grafiku nebo text. Tyto fragmenty jsou pak vazbami sjednoceny do jednotlivých lekcí a kurzů. Každý z těchto systémů poskytuje různou reprezentaci těchto fragmentů. Liší se v atributech, které s sebou tyto

fragmenty nesou. Např. pro textový fragment MyET má povinné atributy: titulek, klíčová slova, potřebné znalosti, závěr a příklady. Tímto způsobem se používají pro vyhledávání nejen první dva textové atributy, ale také dva atributy s odkazy na soubory. AIMS poskytuje možnost zvolit klíčová slova z doménové terminologie. Přes ně je možné získat přímo vazbu na lekci a mapovat materiály vázané na kurz. Podobné rozdíly lze najít i u lekce.

Kapitola 2

Analýza a návrh

Náplní této kapitoly je analýza a návrh editoru pro adaptivní výukové materiály. První část kapitoly se zabývá analýzou požadavků na aplikaci na základě kterých, je možné provést hrubý návrh architektury aplikace a jejích jednotlivých komponent. S ohledem na tuto analýzu je pak možné aplikaci vyvinout.

2.1 Analýza požadavků

Cílem práce je implementovat webovou aplikaci pro tvorbu a úpravu adaptivních výukových kurzů. Tyto kurzy jsou pak nadále používány dalšími aplikacemi určenými pro studium v konkrétním kurzu. Hlavní entitou používanou v těchto aplikacích je tedy kurz. Kurz je rozdělen do několika logických vrstev. Každý kurz je tvořen z lekcí, které se skládají z kapitol. Každá kapitola má určité množství stránek. Nejnižší vrstvou je tzv. výukový objekt, který reprezentuje obrázek, applet, video, kus kódu nebo fragment HTML. Na každé stránce může být větší množství výukových objektů a pro tento účel jsou umisťovány do layoutů, které do sebe mohou být vnořeny. Pro každý výukový objekt, lze definovat pravidlo pro adaptaci v cílových aplikacích. Každý kurz obsahuje seznam klíčových slov, který lze používat pro vysvětlení některých klíčových pojmů kurzu.

2.1.1 Funkční požadavky

Obsahem této sekce jsou požadavky na cílovou aplikaci, které vznikly po diskuzi s vedoucím práce a vlastní analýzou problematiky, spojené s aplikacemi využívajícími model kurzu popsaného výše. Tyto základní požadavky lze nadále dělit na menší snáze uchopitelné požadavky, pomocí kterých pak lze aplikaci navrhnout a vyvinout. Hlavními požadavky na aplikaci jsou:

- editor bude umožňovat tvorbu a úpravu současných i stávajících adaptivních kurzů

- editor bude umožňovat import současných wiki zdrojů
- editor bude mít mód pro zobrazení očekávaného vzhledu v cílové aplikaci
- v editoru bude možné pracovat s lokálními i externími zdroji
- editor bude snadno rozšiřitelný v případě změny doménových objektů
- editor bude schopný vytvořené a upravené adaptivní kurzy exportovat do XML formátu kompatibilního s cílovou aplikací
- editor bude mít maximálně intuitivní a snadno použitelné prostředí

Na základě těchto požadavků je možné editor rozdělit na menší části a pro každou provést analýzu požadavků cílenou na konkrétní oblast v aplikaci.

2.1.1.1 Parser pro wiki zdroje

- parser bude co nejméně závislý na objektovém modelu aplikace
- parser bude schopný konfigurace, kvůli případné změně wiki značkování
- parser bude převádět veškeré značky wiki, kromě vytváření struktury vhodné pro import
- parser bude pracovat s kódováním UTF-8

2.1.1.2 Módy editoru

- editor bude mít editační mód, pro úpravu jednotlivých adaptivních zdrojů
- editor bude mít zobrazovací mód, ve kterém bude možné zobrazit očekávaný vzhled v cílové aplikaci

2.1.1.3 Správa souborů

- editor bude umožňovat správu lokální souborů aplikace
- v editoru bude možné pracovat i s externími soubory
- v editoru bude možné soubory upravovat v prostředí podobné aplikaci Microsoft Word

2.1.1.4 Export souborů

- při exportu bude možné zvlášť exportovat XML soubor pro adaptivní kurz, adaptivní obsah a soubory spojené s kurzem
- při exportu bude možné zadat prefix, kterým se upraví veškeré lokální adresy v XML exportovaného kurzu

Požadavky specifikované výše lze označit za funkční požadavky, tedy požadavky, které specifikují co se musí udělat a identifikuje nutné úkony, aktivity a akce, které musí být vykonány. Naproti tomu jsou tzv. nefunkční požadavky specifikující vlastnosti systému a případné omezující podmínky jako např. použité technologie.

2.1.2 Nefunkční požadavky

- editor bude implementován v jazyce Java EE s podporou Java 1.6
- editor bude používat frameworky Spring, Hibernate a Primefaces
- editor bude používat databázi MySQL
- editor bude běžet na serveru Glassfish 4.1

2.2 Případy užití

Případy užití popisují akce, které lze uživatelem v systému provést. Tyto akce mohou být dále rozděleny podle rolí uživatelů. V aplikaci je pouze jedna role administrátora, tedy veškeré akce zde uvedené je možné provést každým oprávněným uživatelem.

Přihlásit se – Při první spuštění aplikace nemá uživatel žádnou roli a není tedy možné systém používat. Po přihlášení je uživatel autorizován k využívání dalších systémových funkcionalit.

Odhlásit se – Po dokončení práce se systémem se uživatel může odhlásit, kdy je mu odebrána jeho role administrátora a pro další pokračování práce je vyžadováno přihlášení.

Procházet strukturou kurzu – Přihlášený uživatel může procházet strukturou kurzu tedy lekcemi, kapitolami, stránkami, výukovými objekty, klíčovými slovy a adaptačními pravidly.

Upravovat strukturu kurzu – Přihlášený uživatel může upravovat strukturu kurzu tedy přidávat, odstraňovat objekty a upravovat jejich parametry v každé vrstvě výukového kurzu

Exportovat kurz – Přihlášený uživatel může označený kurz exportovat do XML formátu podle XSD schématu kompatibilního s cílovou aplikací

Importovat wiki zdroj – Přihlášený uživatel může převést textový formát wiki stránek přímo na výukové kurzy

Vytvořit a upravit adaptační pravidla – Přihlášený uživatel má možnost vytváření adaptačních pravidel pro jednotlivé výukové objekty

Exportovat adaptační pravidla – Přihlášený uživatel může exportovat adaptační pravidla související s označeným kurzem do XML formátu

Zobrazit zdrojové soubory – Přihlášený uživatel si může zobrazit všechny dostupné soubory

Odstranit zdrojové soubory – Přihlášený uživatel má možnost odstraňovat zdrojové soubory

Upravovat zdrojové soubory výukových objektů – Spolu s editací parametrů výukových objektů může přihlášený uživatel také upravovat soubory související s konkrétním výukovým objektem

Exportovat zdrojové soubory – Přihlášený uživatel má možnost exportovat veškeré soubory související s označeným kurzem

Vytvářet lokální kopie externích souborů – Při úpravě výukových objektů má uživatel možnost zkopírovat externí soubory na lokální úložiště pro jejich případnou úpravu

Vytvořit a upravit výukové objekty – V editoru má uživatel možnost vytvářet a upravovat výukové objekty včetně jejich začlenění do layoutů

Zobrazit očekávaný vzhled stránky v koncové aplikaci – Přihlášený uživatel si může zobrazit předpokládaný vzhled v cílové aplikaci

2.3 Návrh řešení

Kapitola popisuje strukturu modelu výukového kurzu a adaptačních pravidel, používaných v editoru. Vzhledem k tomu, že práce vychází z předchozí práce *Adaptivní kurz Javy*[6] a má být s touto aplikací kompatibilní, je objektový model téměř stejný jako v odkazované práci.

2.3.1 Doménový model kurzu

Základní entitou používanou v editoru je kurz. Kurz se skládá z lekcí, kapitol a stránek. Stránky mohou mít v sobě jeden nebo více výukových objektů. Pro použití více výukových objektů na jedné stránce se používají layouty.

- **AdaptiveCourse** - entita reprezentující "nejvyšší"element v modelu výukový kurz
  - **name** - název výukového kurzu
- **GlossaryKeyword** - každý výukový kurz má vlastní seznam klíčových slov, který se používá pro detailní popis vybraných důležitých pojmů v kurzu
  - **firstChar** - první písmeno klíče pro dané klíčové slovo
  - **key** - samotné klíčové slovo
  - **value** - detailní popis klíčového slova
- **Lesson** - entita reprezentující lekci v kurzu
  - **name** - název lekce
  - **number** - číslo lekce
- **Chapter** - entita reprezentující kapitolu v lekci
  - **name** - název kapitoly
  - **number** - číslo kapitoly
  - **testChapter** - parametr určující, zda se jedná o kapitolu obsahující testové otázky
- **Page** - entita reprezentující stránku v kapitole, jsou v ní umístěné výukové objekty
  - **number** - číslo stránky
- **TestPage** - entita rozšiřující entitu *Page*, reprezentující testovací stránku, která obsahuje testovou otázku
  - **question** - testová otázka
  - **number** - číslo stránky
- **TestAnswer** - entita reprezentující testovou odpověď
  - **value** - odpověď na testovou otázku
  - **correct** - správnost odpovědi

2.3.2 Doménový model výukových objektů

Výukové objekty jako takové obsahují pouze metadata a data s nimi spojená jsou uložena mimo databázi v lokálním úložišti. Každý výukový objekt má proto parametr url díky kterému, se dá odkazovat na příslušný datový obsah. Každá stránka má jednoho potomka typu LearningNode, který může být layout a nebo výukový objekt. Layouty mají proměnné stejné typu a tím je dosaženo možnosti zanořování do sebe s možností obsahu výukových objektů.

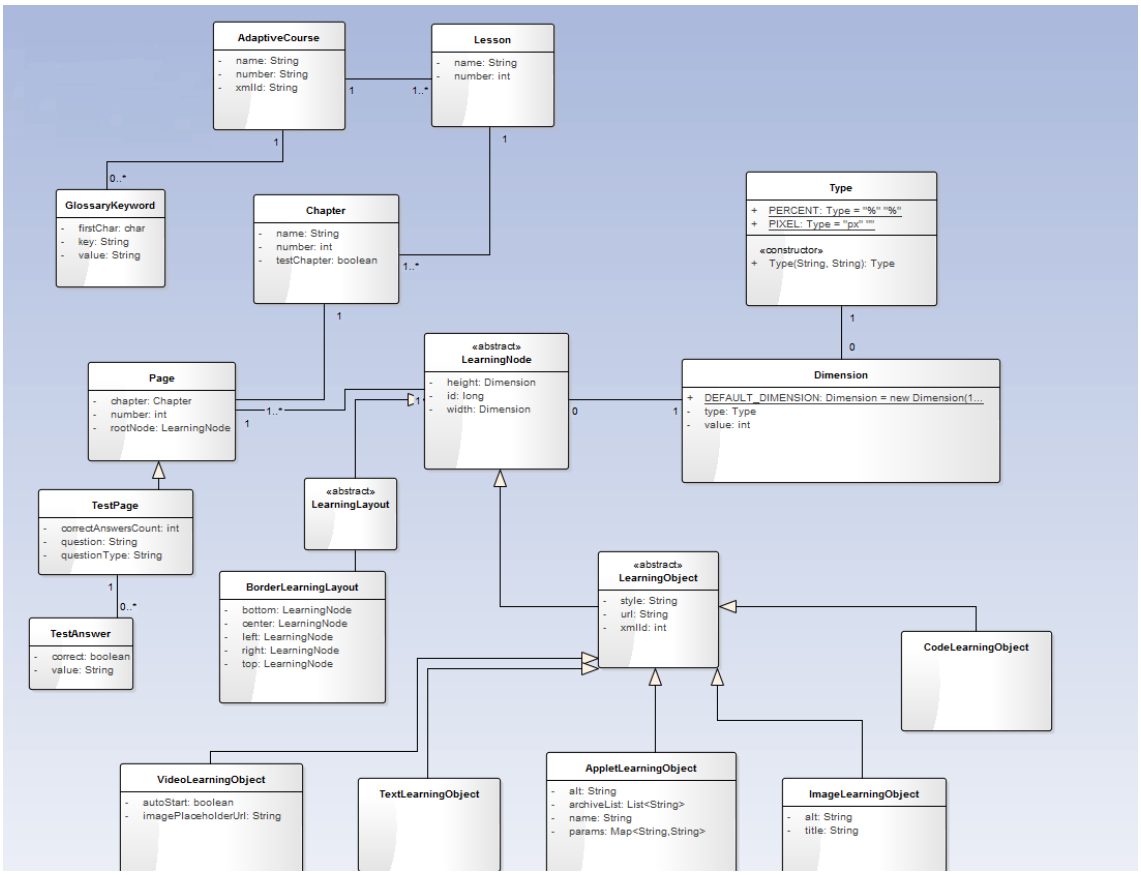
- **LearningNode** - abstraktní entita reprezentující výukový uzel, uzlem může být výukový objekt nebo layout
  - **width** - šířka uzlu
  - **height** - výška uzlu
- **LearningObject** - abstraktní entita rozšiřující LearningNode reprezentující výukové objekty
  - **url** - url adresa lokálního nebo externího souboru, obsahujícího data související s tímto výukovým objektem
  - **xmlId** - identifikátor adaptačního pravidla, které se má pro tento výukový objekt používat
  - **style** - css stylování pro tento výukový objekt
- **LearningLayout** - abstraktní entita rozšiřující *LearningNode* reprezentující výukové layouty

2.3.2.1 Dostupné výukové objekty

V současnosti editor podporuje pět typů výukových materiálů - textový materiál, který může obsahovat fragmenty HTML, kódový materiál, obrázkový materiál, video materiál a applet materiál. Všech pět entit reprezentujících jednotlivé materiály rozšiřuje třídu LearningObject.

- **TextLearningObject** - výukový objekt obsahující HTML fragment
- **ImageLearningObject** - výukový objekt obsahující obrázek
  - **alt** - popisek obrázku v případě, že odkaz na obrázek je z nějakého důvodu neplatný
  - **title** - pomocný popisek pro zobrazení při najetí myši
- **CodeLearningObject** - výukový objekt obsahující fragment kódu
  - **codeType** - jazyk bloku kódu určený ve wiki jako část značky *<code java>*. V případě, že není specifikováno o jaký jazyk se jedná, bude přiřazen jazyk C. Používá se jako koncovka pro soubory obsahující daný blok kódu.

- **VideoLearningObject** - výukový objekt obsahující video
  - **autoStart** - ukazatel, zda se má video spustit po načtení stránky
  - **imagePlaceholderUrl** - adresa obrázku, který se zobrazí při zastavení videa, v HTML 5 tagu <video> se již nepoužívá
- **AppletLearningObject** - výukový objekt obsahující applet
  - **alt** - podobné jako u obrázku, pokud nelze načíst přiložený odkaz na soubor, zobrazí se popisek
  - **name** - název appletu
  - **params** - parametry, které budou předány appletu
  - **archiveList** - seznam odkazů na relevantní soubory pro tento applet



Obrázek 2.1: Kompletní doménový model

2.3.2.2 Dostupné výukové layouty

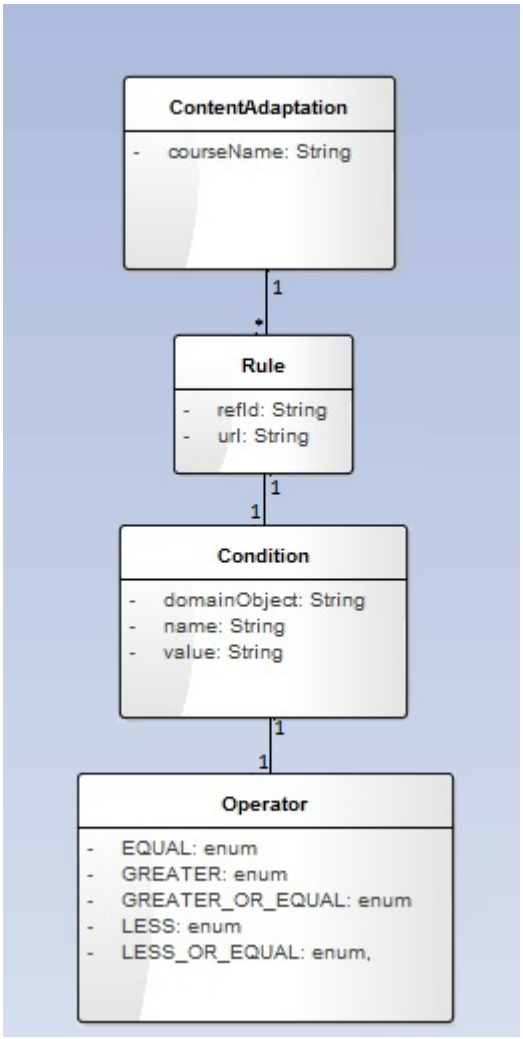
Cílová aplikace nabízí pouze jedinou implementaci LearningLayout, kterou je BorderLayout.

- **BorderLearningLayout** - Tento layout se skládá z až pěti LearningNode objektů (*left*,*center*, *right*, *top*, *bottom*), kde objekt na pozici *center* je povinný.

2.3.3 Doménový model adaptačních pravidel

Adaptační pravidla slouží v cílové aplikaci pro adaptaci obsahu kurzu pro přihlášeného uživatele na základě dostupných informací o jeho aktivitě v kurzu. Doménový model adaptačních pravidel je možné vidět na obrázku 2.2. Pro každý kurz existuje jedna sada adaptačních pravidel *ContentAdaptation*. Každé pravidlo má vlastní identifikátor *refId*, který lze nastavit pro každý výukový objekt do atributu *xmlId*. Pravidlo se vykoná tak, že pokud je splněna jeho podmínka (*Condition*), zamění se url adresa cílových výukových objektů za url daného pravidla.

- **ContentAdaptation** - entita reprezentující sadu adaptačních pravidel pro jeden kurz
  - **courseName** - jméno kurzu, na který se sada adaptačních pravidel vztahuje
- **Rule** - entita reprezentující jedno adaptační pravidlo
  - **url** - odkaz na cílový datový soubor v případě splnění podmínky
  - **refId** - identifikátor pravidla
- **Condition** - podmínka, která určuje, zda se má pravidlo vykonat nebo ne
  - **name** - jméno podmínky, v koncové aplikaci se používají speciálním způsobem generovaná jména pomocí kterých, lze zjistit proti jakým datům se má podmínka testovat např. *lesson_1_test_result_percentage*
  - **value** - hodnota proti které se testují data
  - **operator** - operátor podmínky
- **Operator** - výčet všech dostupných operátorů (LESS, LESS_OR_EQUAL, EQUAL, ..)

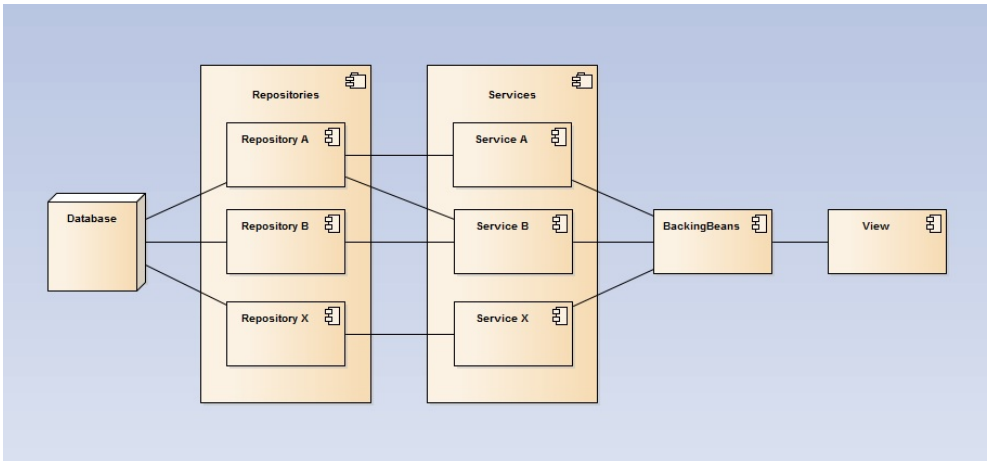


Obrázek 2.2: Model adaptačních pravidel

2.3.4 Architektura aplikace

Návrh aplikace je proveden jako vícevrstevná architektura, jak je vidět na obrázku 2.3.

- **View** - vrstva aplikace kterou uživatel vidí, obsahuje zdrojové soubory JSF,JS,CSS
- **BackingBeans** - vrstva speciálních java tříd standardně používaných v JSF aplikacích, jsou propojeny s UI komponentami na konkrétních stránkách a používají se pro zpracování dat poslaných z těchto stránek a vygenerování odpovědi
- **Services** - služby tvoří mezivrstvu mezi databázovým přístupem a vrstvou *BackingBean*, v této vrstvě jsou obvykle metody, které připravují data pro *BackingBeans*
- **Repositories** - vrstva repozitářů přistupujících přímo do databáze aplikace



Obrázek 2.3: Architektura aplikace

2.3.5 Návrh uživatelského rozhraní

Pro účely návrhu byl vytvořen "low fidelity"prototyp uživatelského rozhraní. Tento typ návrhu obsahuje základní představu o vzhledu aplikace, jeho ovládacích prvcích a jejich rozložení. Vzhled prototypu se může od vzhledu finální aplikace značně lišit. Většinou se používá pro rychlé dodání návrhu aplikace a vyřešení různých problémů, které by mohli následně v průběhu vývoje aplikace nastat. Většinu prototypů lze nalézt na přiloženém DVD.

2.4 Použité technologie

Na základě návrhu částí aplikace v předešlých kapitolách je potřeba zvolit vhodné technologie pro realizaci tohoto projektu. Součástí zadání je použití frameworků Spring, Hibernate a Primefaces spolu s knihovnou ASF. Kromě těchto technologií je dále potřeba technologie pro import,export dat do a z XML formátu a sestavování aplikace.

2.4.0.1 ASF knihovna

Adaptive system framework(ASF) je knihovna, která vznikla pod vedením pana Ing. Martina Balíka, Ph.D. a měla za úkol vytvořit standardy pro práci a vývoj aplikací v adaptivních prostředích na různých platformách. Avšak vzhledem k tomu, že vyvíjená aplikace webový designer adaptivních výukových materiálů nespadá pod skupinu aplikací vyžadující adaptivní chování podle dostupných dat o uživateli, její použitelnost v tomto projektu je menší než byla původně očekávána. Také proto, že části, které by se mohly využít i pro tento projekt, jsou již obsaženy ve frameworku Spring a Hibernate. Proto je tato knihovna v projektu používána spíše okrajově.

2.4.0.2 Spring Framework

Spring framework [7][8] je otevřený aplikační framework pro platformu Java. Používá se pro usnadnění návrhu a vývoje komplexních aplikací v Java EE, podporu použití správných programátorských praktik jako např. použití návrhových vzorů, kvalitní dekompozice kódu nebo použití rozhraní místo tříd. Spring framework je také silně modulární, takže pro vývoj aplikace je možné využít pouze ten modul, který je zrovna potřeba. Stal se velmi populárním v Java komunitě jako funkční náhrada za nevyhovující Enterprise JavaBeans (EJB). Mezi některé výhody Springu patří:

- kód výsledné aplikace je nezávislý na aplikačním prostředí
- minimální až nulová závislost aplikačního kódu na Springu
- Spring je možné použít pro všechny vrstvy aplikace od databázové až po prezentační
- podpora běžně požívaných ORM nástrojů (Hibernate, JDO, iBatis, ..)

2.4.0.3 Hibernate

Hibernate [9] je framework implementovaný v jazyce Java a umožňuje ORM (objektově-relační) mapování. ORM je způsob, jak aplikacím, napsaných v objektových jazycích, umožnit ukládat data objektů do relační databáze. Vybrané třídy, jejichž instance drží data, se kterými se v aplikaci pracuje, reprezentují tabulku v databázi, přičemž datové typy jazyku Java jsou konvertovány

na SQL typy. Hibernate implementuje Java Persistence API(JPA), které je považováno za standard pro objektově relační mapování v jazyce Java. Kromě toho ORM zpracovává také vazby mezi jednotlivými objekty, které jsou pak v databázi reprezentovány cizími klíči a vazebními tabulkami. To výrazně ulehčuje spolupráci databáze a aplikace. Hibernate také podporuje mnoho databázových dialektů jako MySQL, PostgeSQL a další. [10]Díky tomu je vyvíjená aplikace odstíněná od přímého přístupu k databázi a nabízí zjednodušené API pro práci s ní. Pro vývoj repozitářů pro přístup k databázi byl použit modul Spring frameworku Spring Data, který výrazně usnadňuje vývoj této vrstvy. Díky tomuto modulu je možné pouze definovat potřebné repozitáře a rovnou získat potřebný přístup k databázovým tabulkám. Pouhá definice repozitářů zpřístupní nejpoužívanější příkazy na databázi jako SELECT,INSERT a DELETE a další. V případě, že tyto základní příkazy nestačí, je možné přidat do repozitáře speciálně pojmenované definice metod, které pak Spring zparsuje a vytvoří příslušný příkaz do databáze. Pokud by ani tato sada možností dotazů nestačila, je zde samozřejmě možnost vytvářet vlastní příkazy.

2.4.0.4 JavaServer Faces

JavaServer Faces [11] (zkratka JSF) technologie byla vyvinuta společností Sun Microsystems, Inc. Jedná se o framework určený pro vývoj webových aplikací založených na technologii Java. Skládá se ze dvou částí:

- API pro reprezentaci komponent a správu jejich stavu jako např. validace na serveru, konverze dat, definice navigace na stránkách, a další
- knihovny značek pro přidávání komponent na webové stránky a jejich propojení s objekty na serveru

[12]Vývojář definuje komponenty na webových stránkách pomocí speciálních XML značek, které jsou plněny daty ze serveru ze standardních JavaBeans [13]. Díky tomu je aplikace čistě rozdělena na serverovou aplikační logiku a uživatelské rozhraní.

2.4.0.5 Primefaces

Primefaces [14] je otevřená knihovna rozšiřující standardní JavaServer Faces. Obsahuje mnoho UI komponent, které díky silné možnosti parametrizace představují hotové řešení mnoha problémů, se kterými se vývojář může ve vývoji uživatelského rozhraní pro JSF setkat. Krom mnoha hotových komponent je možné si vybrat z mnoha připravených vzhledových témat, které jsou aplikované na veškeré komponenty dostupné v knihovně Primefaces.

2.4.0.6 Serializace

V projektu jsou oblasti vyžadující serializaci/deserializaci objektů z různých formátů. Proto je nutné vybrat vhodnou technologii, která by tuto vyžadovanou funkcionalitu měla.

2.4.0.7 Java Architecture for XML Binding (JAXB)

Pro vytváření samotných výstupů z editoru je potřeba převést objekty uložené v databázi na XML formát, který by byl kompatibilní se vstupem cílové aplikace. Důležité také je, aby formát vstupů a výstupů bylo možné snadno měnit v případě úpravy objektového modelu a bylo možné ho v případě potřeby validovat. Pro tento účel byla vybrána knihovna JAXB [15] od společnosti Oracle Inc. JAXB umožňuje snadný převod mezi XML a objektovou reprezentací s pomocí speciálních anotací objektů a XSD schémat. V případě potřeby exportu jiného formátu dat je velmi jednoduché upravit potřebné anotace a schéma a získat tak potřebnou funkcionalitu.

2.4.0.8 Jackson

V editoru se bude používat kromě exportu do XML pro výstup z aplikace také export objektů do JSON formátu. K tomuto účelu se nejvhodnějšími knihovnami nabízejí knihovna Gson od společnosti Google a knihovna Jackson. Byla vybrána knihovna Jackson díky použití anotací pro převod abstraktních tříd, zatímco u Gson je potřeba implementovat metody pro převod každé abstraktní třídy.

2.4.0.9 Aplikační server

Na výběr je poměrně velké množství aplikačních serverů, které podporují platformu Java EE.

- Apache Tomcat
- JBoss Application Server
- IBM WebSphere Application Server
- Oracle WebLogic
- Glassfish

Ovšem vzhledem k tomu, že na školním serveru běží instance serveru Glassfish, byl použit právě Glassfish [16].

2.4.0.10 JQuery a JQueryUI

JQuery [17] je otevřená javascriptová knihovna s širokou podporou prohlížečů pod licencí MIT. Umožňuje snadno vyhledávat, upravovat a vytvářet nové elementy DOMu včetně jejich atributů s pomocí selektorů stejných jaké se používají v CSS. Je tu také možnost místo selektorů CSS použít selektory XPath. Kromě snadné správy DOM elementů nabízí i další funkce jako jsou události, AJAX, animace a pokročilá práce s poli.

JQuery UI [18] je javascriptový framework, využívající krom jiných knihoven také JQuery a který se zaměřuje spíše na uživatelské rozhraní a přináší balíček hotových komponent a funkcí, které by se jinak musely s větším úsilím implementovat s pomocí samotného JQuery a Javascriptu.

2.4.0.11 Databázový server

MySQL [19] je databázový server, vytvořený švédskou firmou MySQL AB, založený na jazyce SQL. MySQL s sebou přináší řadu výhod jako je podpora všech hlavních platforem, výkon i rychlost a kompatibilita s ostatními systémy. Je dostupný jako open source, tedy je poskytován zdarma. Tento server běží na školní doméně a je proto použit i v tomto projektu.

2.4.0.12 Maven

Apache Maven je nástroj pro správu, řízení a automatizaci kompilace aplikací. V obyčejném projektu, který používá externí knihovny, je potřeba tyto knihovny nejprve stáhnout a až poté je možné projekt zkompilovat. Maven představuje místo, kde je možné nejen stáhnout velkou většinu dostupných aplikací, ale také kontrolovat, zda není dostupná aktualizovaná verze knihovny, čímž značně usnadňuje přenositelnost aplikace a její údržbu. Pro definici závislostí projektu na externích knihovnách se používá značkovací jazyk XML uložený v souboru *pom.xml* v projektu. Tento soubor by v malém projektu mohl vypadat nějak takto:

```
<project>
 <modelVersion>4.0.0</modelVersion>

 <groupId>com.mycompany.app</groupId>
 <artifactId>my-app</artifactId>
 <version>1.0</version>

 <dependencies>
  <dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>3.8.1</version>
```

```
    <scope>test</scope>
  </dependency>
 </dependencies>
</project>
```

Některé značky v tomto souboru jsou povinné.

- **groupId** - jedinečný identifikátor skupiny, do které projekt patří
- **artifactId** - jedinečný identifikátor projektu
- **version** - verze projektu
- **modelVersion** - verze konfiguračního souboru *pom.xml*

Další značky určují závislosti projektu na externích knihovnách.

- **dependencies** - značka označující seznam všech závislostí na externích knihovnách
- **dependency** - značka pro jednu závislost
- **groupId** - podobně jako u značek pro samotný projekt tato značka určuje skupinu ke které knihovna patří
- **artifactId** - identifikátor knihovny
- **version** - verze knihovny, která se má hledat
- **scope** - určuje rozsah závislosti projektu na knihovně

Značka *scope* může obsahovat 6 různých hodnot: *compile, provided, runtime, test, system* a *import*. Určuje rozsah závislosti projektu na knihovně, tedy ovlivňuje kdy a jak se knihovna začlení do projektu.

- **compile** - základní hodnota, pokud *scope* není definován, zkompilovaná knihovna je přístupná všem *classpath* projektu včetně projektů závislých
- **provided** - podobné jako *compile*, ale očekává se, že knihovnu poskytne JDK nebo kontejner při běhu aplikace
- **runtime** - indikuje, že knihovna není potřebná při kompilaci projektu, ale je určená pro běh projektu
- **test** - knihovna je dostupná pouze při fázi testování při kompilaci projektu
- **system** - podobné jako typ *provided* avšak knihovnu musí zajistit vývojář projektu explicitně

- **import** - závislosti by se měli importovat z jiného konfiguračního souboru *pom.xml*, tato hodnota je možná používat pouze v novějších verzích Maven 2.0.9. a vyšších

Kapitola 3

Realizace

Tato kapitola je věnována popisu implementace jednotlivých částí editoru, použitých algoritmů a zásadních částí, které bylo potřeba vyřešit. Pro implementaci prezentační vrstvy byly použity technologie JSF, Primefaces, JQuery a JQueryUI. Aplikační část zastřešuje Spring framework a pro přístup k databázi je použit modul Spring frameworku Spring Data. Kromě toho je v aplikaci nutnost autentizace a autorizace, která je vyřešena pomocí modulu Spring Security. Kromě těchto základních částí aplikace bylo nutno vyřešit import wiki zdrojů, k tomu účelu byla vytvořena vlastní knihovna *Wikiparser*, a převod objektů do různých datových formátů. Pro převod objektů do formátu XML byla použita knihovna JAXB a pro převod objektů z a do formátu JSON byla použita knihovna Jackson.

3.1 Zavedení Spring frameworku

Jako základní vstupní bod pro Spring framework slouží konfigurační soubor *web.xml*. V tomto souboru je možné kromě konfigurace Springu také nastavovat servlety, kódování aplikace a základní webové stránky, které se mají zobrazit při spuštění a chybě aplikace. V první řadě je nutné nastavit Spring. Je potřebovat definovat konfigurační soubor určený konkrétně pro Spring a listener, který tuto konfiguraci zavádí.

```
<context-param>
 <param-name>contextConfigLocation</param-name>
 <param-value>
    /WEB-INF/applicationContext.xml
 </param-value>
</context-param>

<listener>
 <listener-class>
```

```
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
```

Kromě nastavení Spring frameworku je v tomto konfiguračním souboru také nastavení pro kódování webové části aplikace. K tomuto účelu se používají tzv. filtery, které ovlivňují tvar příchozích požadavků a odchozích odpovědí. Následující příklad představuje filter pro zapnutí kódování UTF-8.

```
<filter>
  <filter-name>encoding-filter</filter-name>
  <filter-class>
    org.springframework.web.filter.CharacterEncodingFilter
  </filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>encoding-filter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

První část definice filteru obsahuje informace o jeho jménu, umístění v aplikaci a jaké kódování se má použít. Druhá část obsahuje informaci o sadě adres, kde se má toto kódování používat.

Kromě filteru se zde definuje mapování adres na konkrétní servlety. V projektu se používá JSF framework, proto je mapován na všechny adresy reprezentující webové stránky v projektu. Kromě tohoto základního servletu, který je nutný, aby se webové stránky překládali ze značkovacího jazyka JSF do standardního HTML, je zde servlet pro zpracovávání požadavků na zdrojové soubory z editoru.

```
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>
    javax.faces.webapp.FacesServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>*.xhtml</url-pattern>
```

```
</servlet-mapping>
```

Definice servletů se opět skládá z dvou částí. První je definice samotného servletu, jeho umístění v aplikaci a pořadí v jakém se mají servlety načítat. Samotná značka *load-on-startup* určuje, že se servlet má načíst, jeho hodnota pak určuje pořadí načítaní. Čím nižší hodnota tím dříve se servlet načte, hodnota ovšem musí být kladné celé číslo. Druhá část je mapování servletu na konkrétní tvar adres. Jeden servlet může poslouchat na více adresách. Definice servletu a jeho mapování je párované přes značku *servlet-name*.

Nastavení samotného Spring frameworku se nachází v souboru *applicationContext.xml*, jehož umístění bylo definováno v předchozím konfiguračním souboru *web.xml*. Toto nastavení zahrnuje nastavení připojení do databáze, umístění repozitářů, zabezpečení aplikace, načtení speciálních tříd do kontextu Springu a aktivaci anotací používaných v těchto třídách. Pro samotné fungování jakýchkoli Spring anotací je nutné nastavit balíček, který třídy s těmito anotacemi obsahuje. Je samozřejmě možné nastavit větší množství těchto balíčků. Kromě konkrétní cesty na cílové balíčky je možné přidat filtery, pomocí kterých lze filtrovat třídy procházené pro přidání do Springu.

```
<context:component-scan
  base-package="cz.cvut.fit.prototypeeditor" />
<context:annotation-config />
```

Značka *context:component-scan* označuje umístění balíčků, které obsahují anotacemi označené třídy určené pro Spring. V tomto případě je odkazován základní balíček aplikace, tedy do Springu budou přidány všechny třídy ve všech balíčcích projektu se Spring anotacemi. Těmito anotacemi jsou *@Component*, *@Repository*, *@Service* a *@Controller*. Používají se většinou pro určení, v jaké vrstvě aplikace se daná třída nachází.

- **@Component** - Obecná anotace třídy, může být použita napříč celé aplikace a nevztahuje se k žádné konkrétní vrstvě
- **@Repository** - Touto anotací se označují třídy, které jsou umístěné ve vrstvě persistence a reprezentují databázový repozitář
- **@Service** - Anotace pro třídy umístěné v servisní vrstvě aplikace. V této vrstvě jsou obvykle třídy, které obsahují logiku aplikace anebo jiným způsobem připravují data pro prezentační vrstvu.
- **@Controller** - Tato anotace se použije u tříd, které se nacházejí v prezentační vrstvě.

Tyto anotace však neurčují nutné chování těchto tříd. Je možné všechny třídy označit jako @Service a Spring se načte bez problémů. Je možné, že v budoucnu bude s těmito anotacemi spjata i nějaká hodnota, ale nyní jsou zde hlavně proto, aby vývojáře nutili dodržovat konvence vrstvení aplikace. Kromě

těchto anotací je možné nastavit, jak dlouho bude instance třídy "žít"uvnitř aplikace. Tohoto nastavení se dosáhne pomocí anotace *@Scope* a definice jednoho z modelů životního cyklu instancí tříd ve Springu. Kromě těchto modelů je možné si ve Springu vytvořit svůj vlastní model s pomocí třídy *CustomScopeConfigurer*. Následující modely životního cyklu instancí tříd jsou předdefinované ve Springu a připravené pro použití.

- **@Scope("request")** - Instance takto označené třídy je vytvářena pro každý HTTP požadavek a zaniká s jeho dokončením
- **@Scope("session")** - Tento typ *@Scope* určuje, že instance tříd si drží svůj stav uvnitř HTTP session webové aplikace. Zaniká v případě, že související session zanikne. Instance stejné třídy v různých HTTP session nesdílejí svá data.
- **@Scope("globalSession")** - Podobné jako typ *session*, avšak v tomto případě se jedná o globální session. Narozdíl od typu session jsou však data společná pro všechny ostatní session.
- **@Scope("application")** - Takto označená třída je spjatá s životním cyklem servletů definovaných ve *web.xml*.

Další typy @Scope anotace jsou odděleny od předcházejících, protože jejich životní cyklus není spjatý se zpracováním HTTP požadavku.

- **@Scope("prototype")** - Třída označená takto bude vytvářet nové instance pro každý požadavek na ní. Spring neudržuje žádné záznamy o vytvořených instancích a proto je dobré, aby vývojář vytvořil pro třídu, která používá instanci z takto označené třídy, vytvořil způsob, jakým odstranit vazby na instanci této třídy v případě, že se vytvoří další požadavek na její novou instanci. Tím se předchází případnému *memoryleak* v javě.
- **@Scope("singleton")** - Již z jména je jasné, že se vytvoří pouze jediná instance této třídy v celém Springu. Je velmi podobná jako typ *application*, ale její životní cyklus je spjat přímo s cyklem aplikace. Tato instance je pak uložena v kontejneru Springu a v případě potřeby je předána žadateli.

Pomocí značky *context:annotation-config* se aktivují anotace uvnitř předem již registrovaných tříd. V tomto projektu jsou to hlavně anotace *@Autowired* a *@PostConstruct*.

- **@Autowired** - Touto anotací lze označit prvky, které se mají vyhledávat v kontextu Springu a při tvorbě instance třídy automaticky nastavit

hodnotu nalezené třídy. Kromě proměnných lze anotace přidat i ke konstruktorům nebo setterům. Přednastavené chování anotace je takové, že pokud není nalezena odpovídající třída v kontextu Springu, aplikace se nezkompiluje a vyhodí výjimku. Toto chování nemusí být ve všech případech potřebné a lze vypnout přidáním *(required=false)* nastavení do anotace.

- **@PostConstruct** - Metoda označená touto anotací se spustí ihned po vytvoření instance konkrétní třídy. Většinou se používá pro inicializaci důležitých proměnných. V každé třídě může být pouze jedna taková metoda.

Tato nastavení aktivují Spring pro servisní a prezentační vrstvu. Pro vrstvu repozitářů reprezentující databázovou vrstvu je potřeba definovat další nastavení jako je připojení do databáze, třídu *EntityManager* pro správu jednotlivých entit a manager transakčního zpracování.

```
<bean id="datasource"
    class=
    "org.springframework.jdbc.datasource.DriverManagerDataSource">
 <property name="driverClassName" value="${jdbc.driver}" />
 <property name="url" value="${jdbc.url}" />
 <property name="username" value="${jdbc.username}" />
 <property name="password" value="${jdbc.password}" />
</bean>
```

Toto nastavení určí adresu, kde se databáze nachází, přihlašovací údaje a typ ovladače, který se má pro připojení použít. Pro účely snadné modifikace přístupových údajů databáze je možné definovat externí soubor, ve kterém jsou uloženy potřebná data. V tomto případě se jedná o odkazy typu *jdbc.driver*, které označují jednotlivé *property* uvnitř externího souboru. Takový soubor může vypadat následujícím způsobem:

```
jdbc.dialect=org.hibernate.dialect.MySQL5Dialect
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://url na mysql databazi
jdbc.username=prihlasovaci jmeno
jdbc.password=heslo
```

Tento soubor je cílený pouze na údaje související s nastavení databáze. Je samozřejmě možné definovat podobné odkazy i pro další nastavení a celkově s tím zparametrizovat chování aplikace. Mimo definici údajů o připojení k databázi je nutné definovat třídu, která vytvoří samotné připojení do databáze pro jednotlivé entity a propojí je s frameworkem Hibernate.

```
<bean id="entityManagerFactory" class="org.springframework.orm.
      jpa.LocalContainerEntityManagerFactoryBean">
```

```
<property name="dataSource" ref="datasource"/>
<property name="jpaVendorAdapter">
 <bean class="org.springframework.orm.
     jpa.vendor.HibernateJpaVendorAdapter">
  <property name="databasePlatform" value="${jdbc.dialect}" />
  <property name="showSql" value="true" />
  <property name="generateDdl" value="true" />
 </bean>
</property>

<property name="packagesToScan"
    value="cz.cvut.fit.prototypeeditor.model" />
<property name="jpaProperties" >
 <props>
  <prop key="hibernate.enable_lazy_load_no_trans">true</prop>
 </props>
</property>
</bean>
```

V tomto nastavení je kromě údajů pro připojení do databáze, také adresa balíčku, který obsahuje jednotlivé třídy označené jako *@Entity*, tedy třídy používané v ORM Hibernate. Kromě toho je možné specifikovat další nastavení jako typ dialektu a další. Pro práci s databází je vždy nutné řešit zpracování transakcí a k tomuto účelu slouží následující nastavení:

```
<bean id="transactionManager" class="org.springframework.orm.
    jpa.JpaTransactionManager">
 <property name="entityManagerFactory" ref="entityManagerFactory"/>
</bean>
```

Definuje se s ním manager transakcí a jako parametr vezme předem definovaný *EntityManagerFactory*, který obsahuje potřebná data pro samotné transakce jako ORM mapování a cílovou databázi.

Pro lokalizaci repozitářů se používá následující nastavení:

```
<jpa:repositories
  base-package="cz.cvut.fit.prototypeeditor.repository" />
```

Pomocí tohoto nastavení Spring Data zaregistruje repozitáře pod cílovým balíčkem a s jejich pomocí je pak snadné vytvářet dotazy do databáze. Nové dotazy [20] je možné vytvářet pomocí jmen metod a parametrů např. jméno metody *findByLastnameOrderByFirstnameAsc* vytvoří dotaz do databáze, který vyhledá všechny řádky, kde ve sloupci *lastname* bude hodnota předaná v parametru metodě, seřazené podle sloupce *firstname* vzestupně. Tato schopnost vytváření příkazy v Spring Data pokrývá velké množství příkazů, které by jinak bylo potřeba specifikovat explicitně. Spolu s tím je možné předat do takto pojmenované metody parametr typu *Pageable* z balíčku Spring Data

org.springframework.data.domain, kde je možné nastavit případné stránkování a řazení výsledků. V případě, že generování příkazů s pomocí jmen metod v repozitáři není dostačující, je tu vždy možnost složitější příkazy definovat ručně v anotaci *@Query* nad metodou

```
@Query("SELECT COUNT(o) FROM LearningObject o
            WHERE o.url LIKE CONCAT('%',:urlParam,'%')")
public Long getNumberOfObjectsUsingUrl
            (@Param("urlParam") String urlParam);
```

Samotné query je napsáno v jazyce JPQL, který je velmi podobný SQL.

3.2 Autentizace a autorizace

V předchozí části bylo řečeno, jak se nastaví samotný Spring framework a jeho součásti pro fungování aplikace. Tato část se věnuje nastavení Spring Security, které má na starost autentizaci a autorizaci uživatelů. Toto nastavení je umístěno v stejném konfiguračním souboru jako nastavení celého Springu, avšak z důvodu přehlednosti bylo nastavení pro Spring Security odděleno do jiného xml souboru a následně importováno s pomocí následujícího řádku.

```
<import resource="/securityContext.xml" />
```

Pokud je potřeba do aplikace přidat formu autentizace a autorizace je vždy potřeba definovat několik věcí. Jaké části webové aplikace jsou přístupné jakým uživatelům, šifrování hesla, způsob, jakým se budou vstupní data porovnávat s daty v databázi a případnou navigaci uživatele. Všechny nastavení výše specifikovaná je možné definovat v Spring Security.

```
<bean id="authenticationProvider"
      class="cz.cvut.fit.prototypeeditor.service.impl.
                              SecurityDetailsServiceImpl"/>
<bean id="encoder"
      class="org.springframework.security.crypto.bcrypt.
                              BCryptPasswordEncoder" />
```

Prvotní nastavení v aplikaci je definování služby, která bude porovnávat vstupní data s daty v databázi, a třídy pro šifrování hesla. Toto je pouze jejich definice a přístupnost pod konkrétním identifikátorem používaným dále v nastavení zabezpečení.

```
<ss:authentication-manager alias="authenticationManager">
 <ss:authentication-provider
                  user-service-ref="authenticationProvider">
  <ss:password-encoder ref="encoder" />
 </ss:authentication-provider>
</ss:authentication-manager>
```

Takto vypadá nastavení manageru pro zabezpečení aplikace. Jako parametry bere typ šifrování a službu pro získávání dat o uživateli z databáze definovanou výše. Nyní je Spring Security připraven přihlašovat a odhlašovat uživatele. Dalším krokem je nastavení zabezpečení jednotlivých webových adres proti přístupu neoprávněných uživatelů.

```
<ss:http pattern="/login.xhtml" security="none" />
<ss:http auto-config="true"
                    access-denied-page="/login.xhtml" >
 <ss:intercept-url pattern="/www/**"
                                      access="ROLE_ADMIN" />
 <ss:form-login
    login-processing-url="/j_spring_security_check"
    login-page="/login.xhtml"
    default-target-url="/www/courses.xhtml"
    authentication-failure-url="/login.xhtml?error"
    always-use-default-target="true"
 />
 <ss:logout
    logout-url="/j_spring_security_logout"
    logout-success-url="/login.xhtml?logout" />
</ss:http>
```

S pomocí značky *ss:http* je možné definovat pro různé adresy různá nastavení přístupu uživatelů. Nejjednodušší nastavení je pouhé definování tvarů webových odkazů a přiřazení role uživatele, která k nim má přístup, v předchozím příkladu se jedná o první značku *ss:http*, kde se pro webovou stránku nastaví *security="none"*, což značí přístup všech uživatelů bez zabezpečení. Složitější nastavení je druhá část. Definuje se zde formát webových adres, které je potřeba testovat v *ss:intercept-url* spolu s rolí uživatele, která k nim má přístup, a v případě nesplnění podmínek je uživatel přesměrován na adresu *access-denied-page="/login.xhtml"*. Dále je zde definován formát adresy, která se použije v případě přihlášení nebo odhlášení, a cílové přesměrování v případě úspěchu nebo neúspěchu.

Služba, která slouží k porovnávání dat s databází *SecurityDetailsServiceImpl*definovanou výše je implementací rozhraní *UserDetailsService*, díky čemuž může sloužit účelu porovnávání přihlašovacích údajů s databází v prostředí Spring Security, a skládá se z jediné metody *loadUserByUsername(String string)*, ve které dochází k získávání přihlašovacích údajů uložených v databázi.

```
@Service
@Transactional(readOnly=true)
public class SecurityDetailsServiceImpl
                          implements UserDetailsService {
```

```
 @Autowired
 private PasswordEncoder encoder;

 @Override
 public UserDetails loadUserByUsername(String string) {
  List<GrantedAuthority> authorities =
      new ArrayList<GrantedAuthority>();

  if ("admin".equals(string)) {
   authorities.add
            (new SimpleGrantedAuthority("ROLE_ADMIN"));
   User springUser = new User("admin",
   encoder.encode("admin"), authorities);
   return springUser;

  } else {
   throw new UsernameNotFoundException
                    ("User not found:" + string);
  }
 }
}
```

Jak je vidět tento projekt nepoužívá žádný model uživatelů uložených v databázi. V projektu je pouze jediný uživatel, který má přístup ke všem částem. V případě, že by byla potřeba uživatele nějakým způsobem dělit podle rolí nebo způsobu zobrazení webových stránek, byla by samozřejmě v tomto místě zapojena databázová vrstva. Šifrování hesla je s pomocí anotace *@Autowired*inicializováno na instanci třídy, která konkrétní interface implementuje a je zaregistrovaná v kontextu Springu.

3.3 Doménový model

Samotný doménový model je reprezentován třídami, jejichž atributy představují vnitřní strukturu jednotlivých tabulek v databázi. Pro označení tříd jako představitele struktury databázových tabulek se používají JPA anotace. Pro samotné zaregistrování jako takový typ třídy se používá anotace @Entity. Díky tomu je zaručeno mapování do relačních databázových tabulek. Entity v projektu využívají implementaci primárního klíče, *hashcode* a *equals* metod v třídě *AbstractPersistable*.

```
@Entity
public class AdaptiveCourse extends
                                AbstractPersistable<Long>
```

Je samozřejmě možné, že tabulky budou mít mezi sebou vazby. Podle typu této vazby se v JPA používají různé anotace @OneToOne, @OneToMany, @ManyToOne a @ManyToMany. Kromě toho je možné definovat vlastníka vazby pomocí @MappedBy a název sloupce řazení pomocí @OrderBy.

```
@OrderBy("number")
@OneToMany(mappedBy = "course", cascade = CascadeType.ALL)
private List<Lesson> lessons = new ArrayList<Lesson>();
```

3.3.1 JAXB

Pro export dat do XML formátu se používá knihovna JAXB, která s sebou přináší množství anotací pro označování tříd jako struktury XML. Pro označení třídy jako nejvyššího elementu v XML dokumentu se používá anotace *@XmlRootElement*. Pro označení třídy pouze jako elementu uvnitř XML dokumentu by se použilo označení *@XmlElement*. Kromě těchto dvou základních anotací je k dispozici *@XmlElementWrapper*, který určuje v XML dokumentu značku, která zapouzdřuje všechny elementy v proměnné takto označené. Pokud by třída obsahovala více proměnných z nichž některé nejsou potřeba pro převod do formátu XML, použije se u nich anotace *@XmlTransient*. JAXB totiž automaticky zpracovává všechny gettery jako potencionální proměnné. V případě, že je potřeba zpracovat abstraktní třídu, je potřeba JAXB uvědomit o všech jejích potomcích. XML je vytvářeno s pomocí XSD schématu, které zajišťuje strukturu výsledného XML.

```
@XmlRootElement
public class ContentAdaptation extends
                                AbstractPersistable<Long> {
...
@XmlElement(name = "rule")
@XmlElementWrapper(name = "rules")
public List<Rule> getRules() {...}
...
}

XmlSeeAlso({BorderLearningLayout.class,
                        AppletLearningObject.class,
                        CodeLearningObject.class,
                        ImageLearningObject.class,
                        TextLearningObject.class,
                        VideoLearningObject.class })
public abstract class LearningNode ...
```

3.3.2 Jackson

Vzhledem k tomu, že aplikace využívá také serializaci dat do formátu JSON, je potřeba využít knihovnu Jackson. Podobně jako JAXB nabízí Jackson sadu anotací, kterými lze ovlivňovat výsledný tvar výstupního JSONu.

```
@JsonTypeInfo(
 use = JsonTypeInfo.Id.CLASS,
 include = JsonTypeInfo.As.PROPERTY, property = "@class"
)
@JsonSubTypes({
 @JsonSubTypes.Type(
 value = BorderLearningLayout.class,
 name = "BorderLearningLayout"
)
})
public abstract class LearningLayout {...}
```

Pomocí anotace *@JsonTypeInfo* je možné do JSONu přidat informace o typu serializovaného objektu. V tomto případě je přidán celý název třídy, ale to není nezbytné. Jedná se pouze o identifikátor, který je pro tento typ tříd v JSONu použit a také použit při deserializaci. Není ovšem nutný v případě, že se nejedná o abstraktní třídu. Další anotace, která je spojená s předchozí anotací je *@JsonSubTypes*. V ní se definují všechny třídy, které implementují anotovanou abstraktní třídu.

3.4 Vrstva repozitářů

Je to vrstva nejblíže k databázi. Díky tomu, že projekt používá framework Spring, konkrétně jeho modul Spring Data, je tato část implementace pro vývojáře výrazně ulehčena. Spring Data mu umožňují definovat pouze rozhraní rozšiřující v tomto případě JpaRepository s definicí objektu a typu klíče. To je všechno co Spring Data potřebuje, aby byl schopný vygenerovat podporu pro sadu základních SQL dotazů.

```
public interface ChapterRepository extends
                                JpaRepository<Chapter, Long>{...}
```

JpaRepository je vyšší vrstva původního *CrudRepository*, která také rozšiřuje *PagingAndSortingRepository*, díky kterému je možné také stránkovat a řadit výsledky z databáze. Jak se vytvářejí další složitější query do databáze je rozebráno v kapitole 3.1. Pro správnou funkci Spring Data repozitářů je nutné mít nastaveno v konfiguračním souboru Springu balíček, kde se repozitáře nacházejí.

```
<jpa:repositories
    base-package="cz.cvut.fit.prototypeeditor.repository"/>
```

3.5 Vrstva služeb

Tato vrstva odděluje prezentační vrstvu od repozitářů a nabízí přístup k jejich metodám a zpracování dat před odesláním. Může být využita pro praktický přístup do databáze pomocí různých sad parametrů, které se v této vrstvě mohou převést na požadovaný typ. Základem pro všechny třídy služeb jsou rozhraní pro každou servisní třídu, kde jsou definované metody, které jsou potřeba v aplikaci. Toto oddělení umožňuje zaměňovat různé implementace těchto servisních tříd bez potřeby cokoliv měnit. A díky tomu, že se servisní třídy v ostatních částech aplikace získávají pomocí anotace *@Autowired*, je vše ještě jednodušší. Aplikace si ani nevšimne, pokud by byla celá servisní vrstva vyměněna za jinou. Všechny servisní třídy rozšiřují třídu *AbstractService*, která v sobě má implementaci všech základních metod, které nabízí rozhraní *JpaRepository*. Stejně jako ostatní, má servisní třída vlastní rozhraní *BaseService*, kde se pouze do parametrů určí o jaký typ objektu se jedná a typ jeho identifikátoru.

```
public abstract class AbstractService
              <T extends Object, ID extends Serializable>
              implements BaseService<T,ID>{

@Override
abstract public JpaRepository<T,ID> getRepository();

@Override
public T create(T t){
 return getRepository().save(t);
}

@Override
public void delete(T t){
 getRepository().delete(t);
}
...

@Override
public Pageable createPageAble
  (int pageNum, int pageSize, SortOrder dir, String field){

 Pageable pageAble = new PageRequest(pageNum, pageSize);
 if(field != null){
  pageAble = new PageRequest
  (pageNum, pageSize, new Sort(transformDirection(dir),field));
 }
```

```
 return pageAble;

}

private Direction transformDirection(SortOrder direction){
 if(direction.equals(SortOrder.ASCENDING)){
  return Sort.Direction.ASC;
 }
 else{
  return Sort.Direction.DESC;
 }
}
}
```

Jediné, co je potřeba implementovat v této abstraktní servisní třídě, je typ repozitáře související s konkrétní entitní třídou, pro kterou je tato servisní třída určena. Kromě základní sady metod pro komunikaci s repozitáři je zde i podpora pro stránkování a řazení výsledků z databáze ve spolupráci s objekty, které poskytuje framework Primefaces. Všechny rozhraní servisní vrstvy se nacházejí v balíčku *cz.cvut.fit.prototypeeditor.service* a jejich implementace v balíčku *cz.cvut.fit.prototypeeditor.service.impl*.

3.6 Prezentační vrstva

Následující podkapitola se zabývá implementací prezentační vrstvy s pomocí frameworku JSF a Primefaces. Po přihlášení uživatele je možné procházet doménovým modelem a zobrazovat si jeho jednotlivé vrstvy. Vrstvy jsou zobrazovány pomocí datové tabulky *p:datatable* z frameworku Primefaces. Každá datová tabulka má svojí vlastní BackingBean, které se nacházejí v balíčku *cz.cvut.fit.prototypeeditor.view*.

```
<p:dataTable value="#{coursesView.courses}" .. >
```

Protože ne vždy je dobré zobrazovat všechna dostupná data v tabulce, poskytuje datová tabulka Primefaces způsob, jakým je možné data z databáze stránkovat a řadit. K tomuto účelu je potřeba tabulku plnit speciálním type instance třídy *LazyDataModel*. U ní je nejdůležitější metoda *load*, která ve svých parametrech má potřebné informace pro stránkování.

```
public List<AdaptiveCourse> load
    (int first, int pageSize, String sortField,
     SortOrder sortOrder, Map<String, Object> filters) {
  int pageNum = first/pageSize;
  Pageable pageable = courseService.
  createPageAble(pageNum, pageSize, sortOrder, sortField);
  Page<AdaptiveCourse> page = courseService.findAll(pageable);
```

```
    courses.setRowCount((int) page.getTotalElements());
    return page.getContent();
}
```

Pro přechod do nižší vrstvy než aktuální je potřeba ukládat cestu napříč modelem. Pro tento účel slouží třída *ModelPathComponent*, která je využívána napříč aplikací pro zjištění aktuální pozice v modelu. Každá vrstva obsahuje jiný typ objektů a je plně editovatelná. Je možné objekty jak přidávat, tak i mazat a již existující editovat. Ne všechny vrstvy doménového modelu jsou vhodné pro zobrazení a editaci pomocí datových tabulek. Konkrétně se jedná o úpravu testových stránek, které reprezentují jednotlivé otázky pro testovou kapitolu, a obyčejných stránek, které obsahují strukturu výukových objektů a layoutů.

3.7 Editor výukových objektů a layoutů

Hlavní součástí aplikace je editor výukových objektů a layoutů. Jeho funkcí je ulehčení tvorby a editace adaptivních výukových materiálů, které jsou dále používány jinou aplikací. Kromě intuitivního a snadno použitelného UI nabízí editor i hrubý náhled toho, jak by data mohly na stránce vypadat.

Editor je řešen pomocí "drag-and-drop"ovládacího panelu, pomocí kterého, je možné přidávat výukové objekty do části pro editaci. Ovládací panel se skládá z přepínače módů pohledu, velikosti části pro editaci,ikon reprezentujících dostupné výukové objekty a speciálního prvku reprezentující layout. Obsah ovládacího panelu je generován s pomocí JSONu získaného ze serveru, který obsahuje seznam dostupných prázdných výukových objektů a jejich parametrů. Každý výukový objekt v JSONu má parametr, pomocí kterého může knihovna Jackson zpětně deserializovat JSON na výukové objekty a uložit je do databáze. Tato hodnota bude označována jako "linker", protože se s její pomocí v editoru spojují reprezentace JSON objektů a jejich HTML protějšků. Objekty z JSON jsou převedeny na HTML elementy s odpovídajícími parametry. Tímto způsobem je získán jakýsi obraz prázdných výukových objektů ve formě HTML elementů. Pro vygenerování druhé části editoru, která přijímá elementy z ovládacího panelu, a kontextových menu určených pro editaci jednotlivých elementů, je potřeba od serveru získat další JSON data. Podobně jako u ovládacího panelu je potřeba seznam dostupných layoutů a druhý seznam layoutů, který obsahuje informace o rozmístění jednotlivých prvků vložených do tohoto layoutu. Pro nastavení rozmístění layoutů existují třídy umístěné v balíčku *cz.cvut.fit.prototypeeditor.layoutscontainer*.

```
public class JSONBorderLayout implements JSONLayout{

    private String desc = "BorderLayout";
    private String classType = BorderLearningLayout.class.getName();
```

```
  private JSONLayoutBlock left = new JSONLayoutBlock(...);
  private JSONLayoutBlock center = new JSONLayoutBlock(...);;
  private JSONLayoutBlock right = new JSONLayoutBlock(...);
  private JSONLayoutBlock top = new JSONLayoutBlock(...);
  private JSONLayoutBlock bottom = new JSONLayoutBlock(...);
 ...
}
```

Příklad výše uvádí implementaci třídy určující rozmístění jednotlivých bloků layoutu po stránce. Je nutné, aby všechny *JSONLayoutBlock* byli pojmenovány stejně jako v reprezentovaném layoutu a v atributu *classType* byla stejná hodnota, jako hodnota přidávaná knihovnou Jackson při serializaci abstraktních tříd, která je určená pomocí anotací. Do parametrů konstruktoru třídy *JSONLayoutBlock* se zadává konkrétní umístění bloku v editoru a jeho rozměry. V tomto případě se jedná o reprezentaci *BorderLearningLayout*, jehož třídu je možné vidět zde:

```
@Entity
@Configurable
public class BorderLearningLayout extends LearningLayout
                                  implements Serializable {

  @OneToOne(cascade = CascadeType.ALL, optional = false)
  private LearningNode center;
  @OneToOne(cascade = CascadeType.ALL)
  private LearningNode top;
  @OneToOne(cascade = CascadeType.ALL)
  private LearningNode bottom;
  @OneToOne(cascade = CascadeType.ALL)
  private LearningNode left;
  @OneToOne(cascade = CascadeType.ALL)
  private LearningNode right;
...
}
}
```

Jak je vidět, jména jednotlivých atributů se shodují. Po této části má editor k dispozici JSON reprezentaci každého typu výukového objektu a layoutu s *linkerem* jako jejich identifikátorem a rozvržení všech layoutů se stejným identifikátorem. Tento přístup umožňuje přidávat do této vrstvy doménových objektů nekonečné množství layoutů a výukových objektů bez toho, aby bylo nutné cokoliv měnit, kromě dat získaných od serveru. V případě změny vyšší vrstvy doménového modelu by bylo potřeba upravit tvar datové tabulky a dialogů případně přidat reprezentaci nové vrstvy. Deserializace po dokončení práce na stránce se provádí právě s pomocí *linkerů* u každého použitého HTML elementu, díky kterému je možné získat prázdnou JSON reprezentaci konkrét-

ního výukového objektu a pouze převést parametry nazpět. U layoutů jde o podobný způsob, ale kvůli tomu, že v sobě neobsahují pouze výukové objekty a layouty jako parametry, je nutné nějakým způsobem zjistit, které parametry mají být naplněny reprezentací příslušného objektu anebo pouhým přenesením hodnoty atributu. A k tomuto účelu jsou právě třídy reprezentující rozvržení těchto layoutů. Tyto třídy neobsahují jiné atributy než ty, které představují výukové objekty nebo layouty, kromě neměnných atributů *desc* a *classType*.

3.7.1 Nastavení editoru

Editor je implementovaný jako kombinace Primeface frameworku a javascriptu. Kromě jeho generování na základě vloženého JSONu je možné ho dále nastavovat. Nastavení pro editor se nachází v souboru *Settings.js*. Pomocí něho je možné ovlivnit, jak budou vypadat ikony pro jednotlivé objekty, zobrazení elementů v editační oblasti editoru a v neposlední řadě mapování parametrů. Jak již zde bylo zmíněno, editor neumožňuje přímou úpravu parametrů id, style a type. V případě, že je potřeba nějaký parametr v JSONu zobrazovat pod jiným jménem např. kvůli nevhodnému tvaru s nepovolenými znaku pro HTML atribut, je možné ho namapovat na jiný a při zpětném převodu z HTML na JSON ho používat s původním jménem.

3.7.2 Editace parametrů výukových objektů

Pro účely editace standardních parametrů výukových objektů byla vyvinuta komponenta jako rozšíření frameworku Primefaces [21]. Umožňuje snadné vygenerování tabulky skládající se na každém řádku z jména atributu a vstupním polem pro editaci. Tuto komponentu je možné nalézt balíčku *cz.cvut.fit.prototypeeditor.components*. Pro vytvoření takové komponenty je potřeba zavést reprezentaci komponenty s příslušnými parametry v javě, vykreslovač a její protějšek v podobě javascriptového objektu. V první řadě je potřeba vytvořit samotnou komponentu se všemi potřebnými parametry.

```
@FacesComponent(value = InputParameterGrid.COMPONENT_TYPE)
 @ResourceDependencies({
 @ResourceDependency(library = "primefaces",
                     name = "jquery/jquery.js"),
 @ResourceDependency(library = "primefaces",
                     name = "primefaces.js"),
 @ResourceDependency(library = "editorfaces",
                     name = "input-parameter-grid.js")
})
public class InputParameterGrid extends UIComponentBase
                                    implements Widget{
        ...
}
```

Jsou dvě možnosti jak nově vytvořenou komponentu zaregistrovat do JSF: s pomocí anotace *@FacesComponent* a typem komponenty anebo v konfiguračním souboru *faces-config.xml* s pomocí následujícího kódu.

```
<component>
 <component-type>
  cz.cvut.editorfaces.InputParameterGrid
 </componenttype>
 <component-class>
  cz.cvut.fit.prototypeeditor.components.InputParameterGrid
 </component-class>
</component>
```

Zde je také potřeba kromě typu komponenty uvést její cestu v projektu. V této třídě jsou reprezentace všech parametrů, které jsou pak používané na JSF webové stránce. Pro ukládání jejich hodnot je používán *StateHelper*, což je součást frameworku JSF. Reprezentace hodnoty komponenty vypadá takto:

```
...
public Object getSettings(){
 return getStateHelper().eval(PropertyKeys.settings,null);
}

public void setSettings(Object param){
 getStateHelper().put(PropertyKeys.settings,param);
}
...
```

Pro každou takovou komponentu je potřeba vytvořit její "renderer", který definuje způsob jakým bude komponenta vykreslena na webové stránce.

```
@FacesRenderer(
 componentFamily = InputParameterGrid.COMPONENT_FAMILY,
 rendererType = InputParameterGridRenderer.RENDERER_TYPE
)
public class InputParameterGridRenderer
                                    extends CoreRenderer{...}
```

Podobně jako reprezentace komponenty je nutné i *renderer* zaregistrovat do JSF frameworku s pomocí anotací nebo jeho definice v konfiguračním souboru *faces-config.xml*. Je zde nutné přepsat metody pro definici HTML kódu této komponenty a jejího javascriptu.

```
protected void encodeMarkup
(FacesContext context, InputParameterGrid grid)
{
 ResponseWriter writer = context.getResponseWriter();
```

```
 writer.startElement("div", grid);
 writer.writeAttribute("id", grid.getClientId(), null);
 writer.writeAttribute("name", grid.getClientId(), null);
 writer.endElement("div");
}

protected void encodeScript
(FacesContext context, InputParameterGrid grid)
{
 String clientId = grid.getClientId();
 String widgetVar = grid.resolveWidgetVar();

 WidgetBuilder wb = getWidgetBuilder(context);
 wb.init("InputParameterGrid", widgetVar, clientId);
 wb.attr("onChange", grid.getOnChange());
 wb.finish();
}
```

V první metodě *encodeMarkup* se definuje základní vzhled komponenty na webové stránce. V metodě *encodeMarkup* pak inicializace javascriptového protějšku a nastavené parametrů pro toto volání. Javascriptová část komponenty vypadá takto:

```
PrimeFaces.widget.InputParameterGrid =
PrimeFaces.widget.BaseWidget.extend
({
 init : function(cfg) {
  this._super(cfg);
  ...
 },
...
})
```

Jak je vidět tato část je rozšířením Primeface javascriptové knihovny. Metoda *init* se zavolá, díky inicializaci v *rendereru* a definované parametry se nacházejí v objektu *cfg*. Pokud jsou všechny tyto tři části hotové, je ještě potřeba přiřadit komponentu k jejímu *rendereru* a definovat pro ně *namespace*. K tomuto účelu slouží konfigurační soubor *editorfaces.taglib.xml*.

```
<facelet-taglib>
 <namespace>http://cz.cvut.editorfaces/ui</namespace>
 <tag>
  <description><![CDATA[...]]></description>
  <tag-name>inputParameterGrid</tag-name>
  <component>
   <component-type>
```

```
    cz.cvut.editorfaces.InputParameterGrid
   </component-type>
   <renderer-type>
    cz.cvut.editorfaces.InputParameterRenderer
   </renderer-type>
  </component>
  <attribute>
   <description> <![CDATA[...]] > </description>
   <name>id</name>
   <required>false</required>
    <type>java.lang.String</type>
  </attribute>
  ...
```

Jak je vidět, definuje se zde i sada dostupných atributů pro konkrétní komponentu, jejich typ a nutnost jejich vyplnění. Tím je komponenta hotová a zbývá ji jen přiřadit do projektu do *web.xml* konfiguračního souboru.

```
<context-param>
 <param-name>javax.faces.FACELETS_LIBRARIES</param-name>
 <param-value>/WEB-INF/editorfaces.taglib.xml</param-value>
</context-param>
```

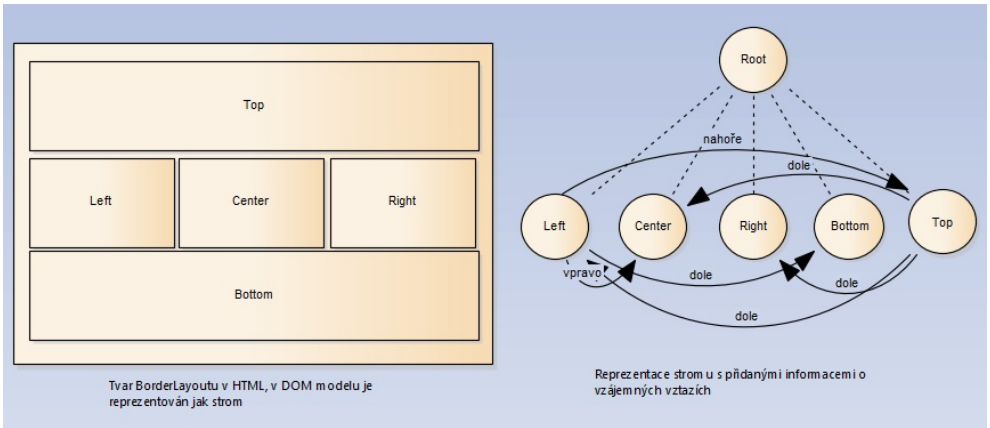
Pro komponentu je dostupná konfigurační třída, kde je možné definovat typovou hodnotu jednotlivých editovaných parametrů pro validaci. Kromě toho lze definovat parametry, které se nemají editovat vůbec. Tato třída se nachází v balíčku *cz.cvut.fit.prototypeeditor.components.settings*. Nastavení komponenty podporuje vstupní formáty typu číslo, *boolean*, JSON pole a JSON objekt. Komponenta se plní daty pomocí javascriptové funkce *setEditedElement()*, kde pro přístup ke komponentě se používá *PF('widgetName')*, což je Primefaces selector pro jednotlivé widgety. Jméno widgetu se udává pod parametrem *widgetVar*. Po zavolání této funkce, vygeneruje komponenta dvojici jméno atributu a vstupní pole pro každý parametr ve vstupním HTML elementu kromě těch, které jsou definované v nastavení jako needitovatelné. Jako výstup je volán javascript definovaný pod parametrem *onChange* spolu s detailem editovaného atributu. Kromě toho je možné zjistit celkový stav všech editovaných atributů, tedy zda je u všech zadaná správná hodnota, s pomocí metody *areInputsValid*. Komponenta automaticky mění atributy HTML elementu v případě korektní editace.

3.7.3 Náhled adaptivních materiálů

Pro uživatele může být někdy obtížné si ve změti v sobě vnořených layoutů představit, jak by vlastně vložená data mohla na cílové stránce vypadat. Náhled to pro uživatele ulehčí tím způsobem, že roztáhne všechny komponenty

na maximální možnou velikost ignorující prázdné nevyužité části layoutů a zachová poměry zbylých komponent v jednotlivých layoutech.

Pro funkci náhledu byl vytvořen algoritmus pracující na speciální datové struktuře, která reprezentuje editovanou HTML stránku. Obyčejná HTML stránka je tvořena HTML značkami, které mají v sobě buď další značky anebo jsou prázdné bez dalších potomků. Tato struktura lze nejlépe reprezentovat pomocí stromu objektů. Avšak pro účely náhledu je strom struktura s nedostačujícím množstvím informací a tudíž je potřeba do stromu zavést něco dalšího. Pro tento algoritmus je každá vrstva stromu reprezentovaná orientovaným grafem, kde každý uzel má vazby na další pokud jsou přímí sousedé, tedy jsou přímo vedle sebe. Krom toho každá vazba je pojmenovaná v závislosti na tom, kde se nachází druhý uzel v závislosti na prvním (vlevo,vpravo,nahoře a dole).



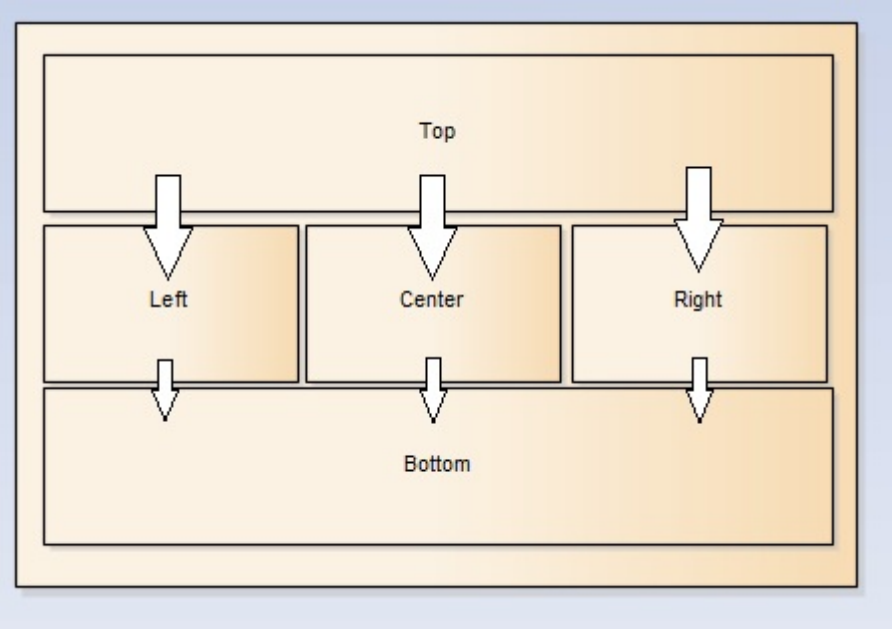
Obrázek 3.1: Stromový model reprezentace HTML elementů v JS

Vazby jsou v obrázku 3.1 uváděny jen pro elementy left a top kvůli přehlednosti.

3.7.3.1 Algoritmus pro rozšiřování HTML elementů

V této sekci bude vysvětleno, jak funguje algoritmus, který je používán pro náhled adaptivních materiálu na stránce. Hned ze začátku je důležité zmínit, že algoritmus má dvě fáze, které jdou za sebou. Jedná se o směr, ve kterém je prováděno rozšiřování (horizontální pro vazby vlevo a vpravo,vertikální pro vazby nahoře a dole). Rozdělení směrů rozšiřování je nutné, aby jednotlivé HTML elementy zůstali přímo u sebe a nevznikali prázdná místa.

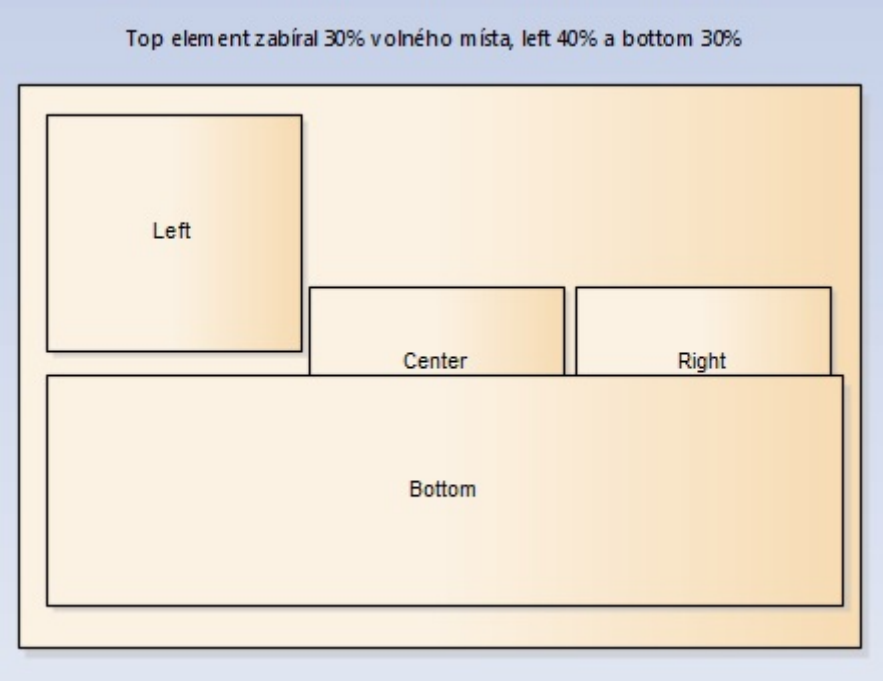
V každé fázi se opakuje pro každý prázdný element stejný způsob rozšiřování. Pokud se používaný graf vezme z pohledu jednoho prázdného uzlu, určujícího dostupné volné místo, jako kořenu tohoto grafu a vazby se omezí pouze na jeden typ, vznikne strom s daným uzlem v jeho kořeni. Takto vytvořený strom se prochází v pořadí pre-order (pro typ vazeb vpravo a dolů) a post-order (pro typ vazeb vlevo a nahoru). Tyto způsoby průchodů jsou zvoleny kvůli tomu, jak jsou jednotlivé elementy v HTML umisťovány na stránku tedy pomocí x,y souřadnice, kde hodnota osy y se zvyšuje ve směru dolů. Je potřeba jednotlivým uzlům grafu měnit souřadnice v závislosti na této skutečnosti. Tedy vždy ve směru osy x,y. Samotný algoritmus funguje následujícím způsobem rekurzivně.



Obrázek 3.2: Stromový model omezený na jeden typ vazeb s kořenem v Top elementu

Vždy se nejdříve provede průchod stromem post-order a až poté pre-order, tedy nejdříve se rozšíří prvky blíže k osám x,y. Při průchodu stromem se akcí, která se má vykonat, rozumí rozšíření a posun souřadnic uzlu v horizontálním

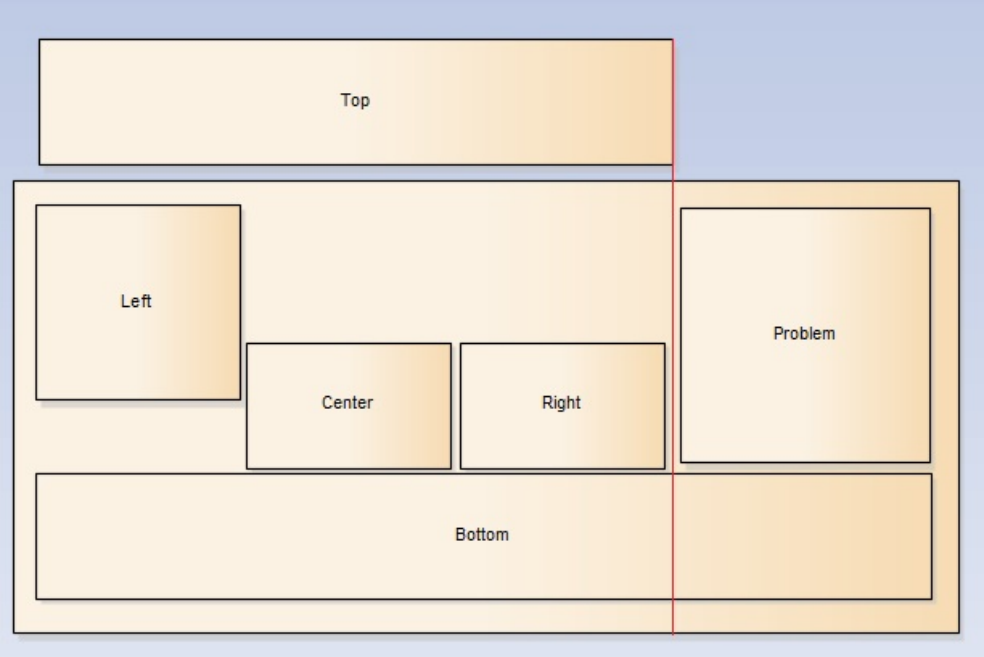
nebo vertikálním směru podle fáze, ve které se algoritmus nachází. Souřadnice je nutné posunout o tolik, o kolik se rozšířili prvky více resp. méně zanořené v stromu v případě průchodu post-order resp. pre-order. U druhého typu průchodu je nutné posun ještě zvýšit o tolik, kolik se rozšířili prvky z prvního průchodu. O kolik se má rozšířit lze jednoduše vypočítat jako část z volného místa podle toho, kolik prvek obsazuje místa v celém grafu. Tedy v případě, že mám volného místa 30%, rozšiřovaný prvek zabírá 10% ze celé velikosti grafu (100%), tak se má rozšířit o 10% z 30% tedy 3%. Tímto způsobem lze zachovat poměry šířek, které prvky mají a tím i strukturu vzniklé stránky. Při průchodu stromem je nutné sledovat, zda tam již algoritmus nebyl, aby se neprovedl víckrát na stejném prvku v případě, že dva prvky mají společného potomka.



Obrázek 3.3: Tvar elementů na stránce po průchodu první větve stromu, následují větve s elementy center a right

V případě, že průchod je prováděn vždy z většího prvku na stejně velký nebo menší výše popsaný algoritmus bude fungovat. V případě, že tato podmínka není dodržena, je potřeba nějakým způsobem řešit vzniklou situaci. Jde o situaci, kdy prvek "přesahuje"přes kořen stromu ve kterém se nachází.

V takovém případě je nutné zjistit, zda je možné prvek rozšířit a zmenšit sousedy, ke kterým přesahuje. To je pravda, pokud jsou tito sousedé všichni prázdní a tudíž odstranitelní. V opačném případě je potřeba veškeré změny provedené v současném stromu zrušit a pokračovat v algoritmu dále na jiném



Obrázek 3.4: Po průchodu až k elementu *bottom* se *bottom* nemůže rozšířit pokud element *problem* není prázdný, protože tento element nespadá do stromu kořenového elementu *top*

prázdném prvku.

3.7.4 Převod dat

Pro převod dat kurzu do formátu XML slouží třída *WikiJAXBXmlConverter* umístěná v balíčku *cz.cvut.fit.prototypeeditor.utils*. V této třídě se používá knihovna JAXB. Díky tomu, že pro převod dat je potřebné XSD schéma je zaručeno, že data budou konsistentní a v požadovaném tvaru pro koncovou aplikaci, která používá stejné XSD schéma.

3.8 Knihovna pro zpracování wiki zdrojů

Kromě samotného editoru adaptivních materiálů bylo potřeba data v současnosti uložená ve wiki systému nějakým způsobem naimportovat do editoru adaptivních materiálů. K tomuto účelu byla vytvořena knihovna, která převede textovou formu wiki zdrojů do objektového tvaru realizovaného jako stromová struktura. Tato kapitola se bude detailněji zabývat realizací této knihovny.

3.8.1 Datová struktura

Jako první je třeba vysvětlit datovou strukturu, kterou tato knihovna používá. Jeden z požadavků diplomové práce byla snaha o co největší přenositelnost jednotlivých komponent. K tomuto účelu bylo potřeba vytvořit vlastní datovou strukturu tak, aby knihovna bylo kompletně nezávislá na okolních aplikacích. Třídy určené pro reprezentaci wiki zdrojů můžeme nalézt v balíčku cz.cvut.fit.wikiparser.stack.linkedlist . Jednotlivé třídy budou vysvětleny v následujícím seznamu.

WikiObject – Pro účely reprezentace wiki zdrojů postačí jediná třída WikiObject, která umožňuje uložit hodnotu, typ a potomky datového zdroje a další parametry, které jsou používané dále v knihovně

TableWikiObject, ListWikiObject – rozšíření základní reprezentace wiki zdrojů pro seznamy a tabulky, kde jsou potřeba další informace pro přesnou reprezentaci těchto typů dat

IWikiObject – V případě, že základní třída WikiObject je z nějakého důvodu nevyhovující pro aplikaci, která tuto knihovnu používá, je možné používat vlastní implementaci reprezentace dat s pomocí tohoto rozhraní.

WikiObjectConstants – jednoduchý kontejner na konstanty používané uvnitř WikiObject

WikiObjectWrapper – třída obalující základní třídu WikiObject a rozšiřující ji o další parametry určené pro spojové seznamy používané v dalších částech knihovny

3.8.2 Logika knihovny

V předešlé sekci bylo vysvětleno, že knihovna používá vlastní datovou strukturu ve formě stromu. Tyto stromy je však nutné nějakým způsobem spravovat, aby ve výsledku reprezentovali celkový wiki zdroj. K tomuto účelu slouží zásobník a akce nad tímto zásobníkem, které můžeme najít v balíčku *cz.cvut.fit.wikiparser.stack* a *cz.cvut.fit.wikiparser.stack.actions* . V následujícím seznamu budou vysvětleny jednotlivé třídy v těchto balíčcích.

IParserStack – Rozhraní reprezentující zásobník používaný v knihovně.

ParserStack – Základní implementace zásobníku v knihovně. Je implementován jako spojový seznam za použití třídy WikiObjectWrapper zmíněné v předchozí sekci. Kromě základních funkcí zásobníku push a pop jsou zde funkce pro ukončení tvorby probíhajícího WikiObject v případě, že nemá specifický ukončovací tag ve wiki zdroji a ukončení celého zásobníku, tedy postupné ukončení všech probíhajících WikiObjectů. V

případě, že by z nějakého důvodu byla tato implementace nevyhovující je možné použít rozhraní IParserStack a používat vlastní implementaci zásobníku. Kde je možné nastavit vlastní zásobník bude vysvětleno v následujících kapitolách.

IStackAction – Rozhraní pro akci, která nějakým způsobem upravuje zásobník

StackAction – Implementace základního způsobu vykonávání akci, kterou nadále rozšiřují další konkrétní akce

***Action** – Všechny tyto třídy rozšiřují třídu StackAction a nějakým způsobem upravují zásobník podle konkrétní akce

Poslední balíček v oblasti zásobníku cz.cvut.fit.wikiparser.stack.actionsfactory je, jak je již z názvu patrné, balíček obsahující továrnu na akce pro úpravu zásobníku.

IActionFactory – Rozhraní reprezentující továrnu na akce upravující zásobník

ActionFactory – Implementace továrny na akce. Kromě standardního getteru na akce pro každý řádek wiki zdroje jsou zde funkce pro úpravu základní sady akcí, kterou továrna používá. Díky tomu je knihovna více ohebná a upravitelná pro aplikaci, která jí používá.

V této sekci by bylo vhodné zmínit balíček cz.cvut.fit.wikiparser.util s jedinou třídou LinksTextParser. Tato třída má na starost vyhledávání a nahrazování značek, které by se mohli nacházet uvnitř řádku wiki zdroje, tedy ne na jeho začátku. Pro příklad lze uvést značky [[,]] nebo {{,}} , které ve wiki reprezentují odkazy na nějaké zdroje na serveru. Tento formát by byl nežádoucí, proto jsou nahrazovány html značkami pro odkaz případně obrázek.

3.8.3 Používání a nastavení knihovny

Pro používání knihovny stačí vytvořit novou instanci třídy WikiParser z balíčku cz.cvut.fit.wikiparser . Kromě funkce pro start zpracování přiložených dat jsou zde funkce pro nastavení komponent používaných v knihovně jako továrna na akce a zásobník. S pomocí rozhraní zmíněných v předchozích sekcích je možné vložit do knihovny vlastní implementaci obou komponent. Kromě třídy WikiParser je zde třída ParserSettingsProvider, pomocí které je možné změnit nastavení pro jednotlivé akce. Je tedy možné změnit identifikátory jednotlivých akcí bez nutnosti vkládat novou sadu akcí, což umožňuje znovupoužití již dostupných akcí bez nutnosti provést vlastní implementaci a následně vložit sadu těchto akcí do knihovny.

Kapitola 4

Testování

Kapitola popisuje, jakým způsobem byla aplikace testována. Kromě Unit testů pro testování logiky aplikace, byl pro test uživatelského rozhraní použit kognitivní průchod aplikací, který umožňuje odhalit chyby uživatelského rozhraní aplikace bez potřeby většího množství uživatelů.

4.1 JUnit testování

Pro testování základní logiky aplikace byly použity JUnit testy a mockovací frameworky Mockito a PowerMockito. Testy pokrývají následující části aplikace:

- Základní logika aplikace

  `cz.cvut.fit.prototypeeditor.logic`

- Prezentační vrstva a komponenty

  `cz.cvut.fit.prototypeeditor.components`
  `cz.cvut.fit.prototypeeditor.view`

Framework Mockito umožňuje vytvářet tzv. *mock* objekty, které jsou jakýmsi obrazem původních objektů bez původní funkcionality. Funkcím těchto objektů je pak možné přiřazovat chování, které zůstává při průběhu testu neměnné. Díky tomu je možné testovat chování konkrétní části programu a odstínit ji od závislostí na ostatních částech programu.

Testy jsou spouštěny s pomocí třídy *PowerMockRunner*, která umožňuje vytvářet *mock* objekty anebo odchytávat události i ze statických tříd a metod.

```
@RunWith(PowerMockRunner.class)
public class ChaptersViewTest {
```

```
 @PrepareForTest(MsgsUtil.class)
 @Test
 public void testEditChapter() {
 PowerMockito.mockStatic(MsgsUtil.class);
 Chapter test = new Chapter(1, "Test 0");
 instance.setNewChapterName("Test 1");
 instance.setNewChapterNum(2);

 ArgumentCaptor<Chapter> chapterCaptor =
                 ArgumentCaptor.forClass(Chapter.class);

 when(chapterSer.find(any(Long.class))).thenReturn(test);
 when(
  chapterSer.findByNumberAndLesson
            (any(Integer.class),any(Lesson.class))
 ).thenReturn(test);
 when(
  chapterSer.update(chapterCaptor.capture())
 ).thenReturn(new Chapter(1,"Test 1"));

 instance.editChapter();

 verify(chapterSer, times(1)).update(any(Chapter.class));

 Chapter capture = chapterCaptor.getValue();
 assertEquals("Test 1", capture.getName());
 assertEquals(new Integer(1), capture.getNumber());
 ...
 }
}
```

4.2 Kognitivní průchod

Pro účely testování uživatelského rozhraní existuje více možností jak rozhraní testovat. Jednou z nich je **kognitivní průchod**, který zjišťuje použitelnost aplikace pro nové uživatele, tedy srozumitelnost a obtížnost orientace v nich. Při testování jsou testerem nebo skupinou testerů prováděny předem definované úlohy na stránkách a při nich je zvažováno, jak by daná operace byla obtížná pro nového uživatele, jestli je dostatečně srozumitelná, jestli má uživatel dostatek informací pro další kroky atd. Pokud je množství informací uživatele označeno při testování jako dostatečné, je i úkol označen za splněný, a v opačném případě je potřeba ukázat na problémy v návrhu uživatelského

rozhraní. V každém kroku, který v průběhu konání úkolu tester udělá je potřeba si pokládat následující otázky:

Stanoví si uživatel správný cíl? – Tato otázka určuje, zda byl uživatel schopen, na základě informací z uživatelského rozhraní, rozpoznat další krok pro vyřešení úkolu.

Je akce, kterou by měl uživatel udělat, vidět? – Určuje, zda je způsob provedení dalšího kroku pro uživatele viditelný např. není skrytý uvnitř složitých menu apod.

Je akce správná? – Uživatel by měl ihned vědět, zda ho provedená akce vede k cíli. Na příklad uživatel není puštěn do dalšího kroku dokud nezadá potřebné vstupy.

Rozumí uživatel zpětné vazbě? – Rozhraní by mělo být schopné uživateli předat dostatečné informace o výsledku jeho akce, minimálně tedy ne/úspěšnost jejího provedení.

Pro aplikaci testování kognitivním průchodem je potřeba nejprve definovat cílovou skupinu uživatelů, veškeré úkoly, které by měl uživatel být schopný splnit spolu s jejich předpokládaným průchodem. Poté následuje samotný průchod rozhraním s aplikací otázek definovaných výše. Aplikace by měla být určená uživatelům technicky velmi pokročilým se znalostí struktury výukových kurzů, které jsou v aplikaci editovány. Předpokládá se, že se nebude jednat o uživatele s jakýmkoliv zdravotním postižením, které by mohlo ovlivnit jejich průchod aplikací. Také se předpokládá, že uživatel rozumí anglickému jazyku.

4.2.1 Obecný popis úloh

- **Vytvoření,editace jména a smazání kurzu**
  1. Přihlášení uživatele s pomocí poskytnutých údajů
  2. Kliknutí v menu na **Add new**
  3. Vyplnění potřebných údajů a potvrzení
  4. Kliknutí na editační tlačítko, upravení údajů o kurzu a potvrzení
  5. Kliknutí na tlačítko pro odstranění
- **Import nové lekce**
  1. Označení kurzu pro import lekce
  2. Kliknutí na tlačítko v menu pro import
  3. Zkopírování dat a potvrzení

- **Otevření editace výukových objektů v nově vytvořené stránce**

  1. Kliknutí na tlačítko "detailu"kurzu
  2. Kliknutí na tlačítko "detailu"lekce
  3. Kliknutí na tlačítko "detailu"kapitoly
  4. Přidání nové stránky pomocí menu
  5. Kliknutí na tlačítko "detailu"přidané stránky

- **Vytvoření Borderlayoutu v jedné ze stránek s textovým výukovým objektem**

  1. Přetáhnutí layout ikony do oblasti editace
  2. Nastavení BorderLayoutu pomocí kontextového menu
  3. Přetáhnutí textové ikony do oblasti editace

- **Vytvoření obrázkového výukového objektu s referencí na lokální kopii externího zdroje a uložení práce**

  1. Přetáhnutí ikony pro obrázkový výukový objekt do oblasti editace
  2. Pomocí kontextového menu otevřít editaci parametrů
  3. Nastavit externí adresu zdroje
  4. Stisknout tlačítko **copy**
  5. Zavřít dialog a stisknout tlačítko **Save page**

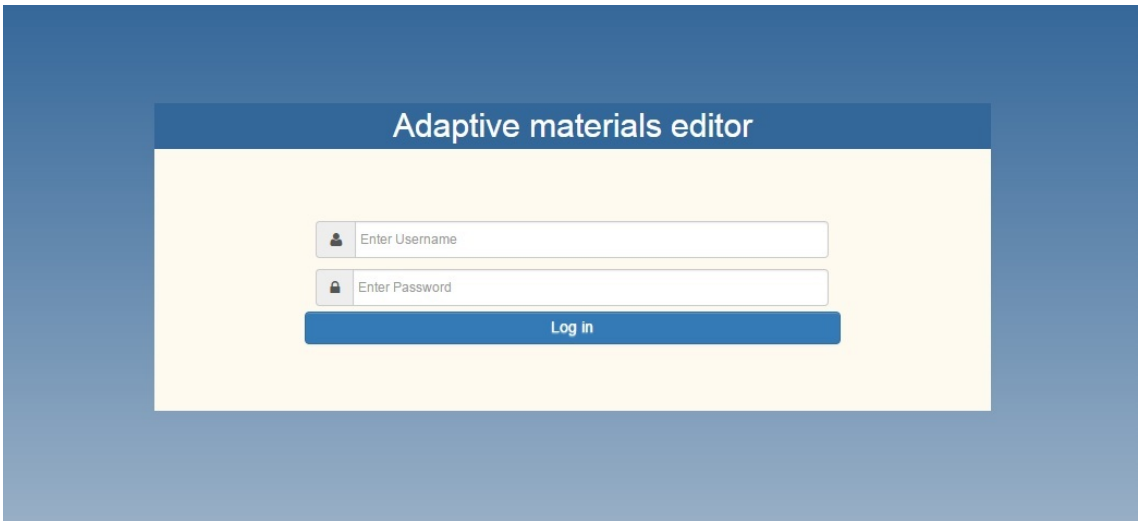
- **Export XML reprezentace kurzu**

  1. Navigace kamkoliv zpět pomocí breadcrumb menu
  2. Kliknutí v menu na **Export course**
  3. Stisknout tlačítko **Export course sources**

4.3 Průběh testování

4.3.1 Vytvoření, editace jména a smazání kurzu

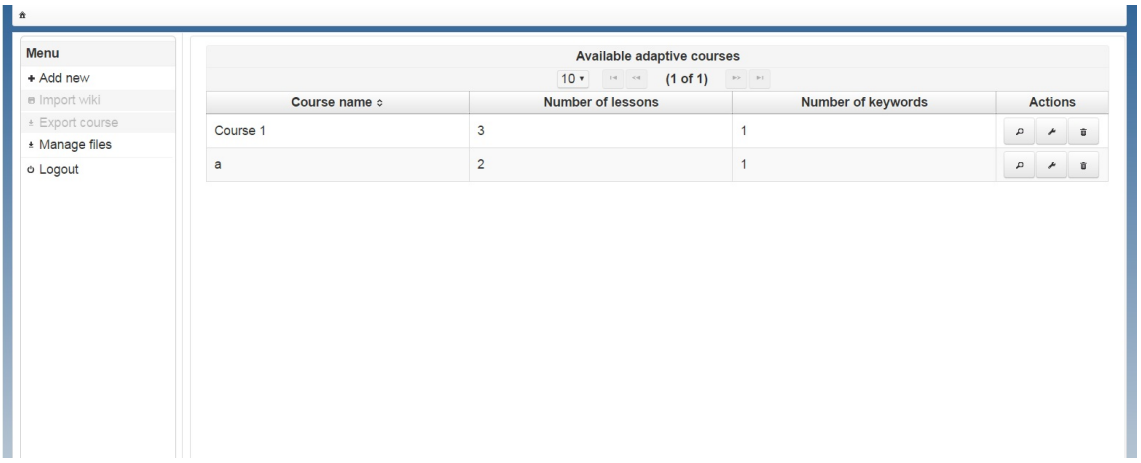
4.3.1.1 Krok 1



Obrázek 4.1: Úvodní obrazovka aplikace

- **Stanoví si uživatel správný cíl?**
  - Ano, vstupní pole a tlačítko pro přihlášení je jasně vidět.
- **Je akce, kterou by měl uživatel udělat, vidět?**
  - Ano, vstupní pole a tlačítko pro přihlášení je jasně vidět.
- **Je akce správná?**
  - Ano, uživatel je v případě správně zadaných údajů přesměrován na stránku editoru
- **Rozumí uživatel zpětné vazbě?**
  - Ano, v případě špatně zadaných údajů se zobrazí informační box

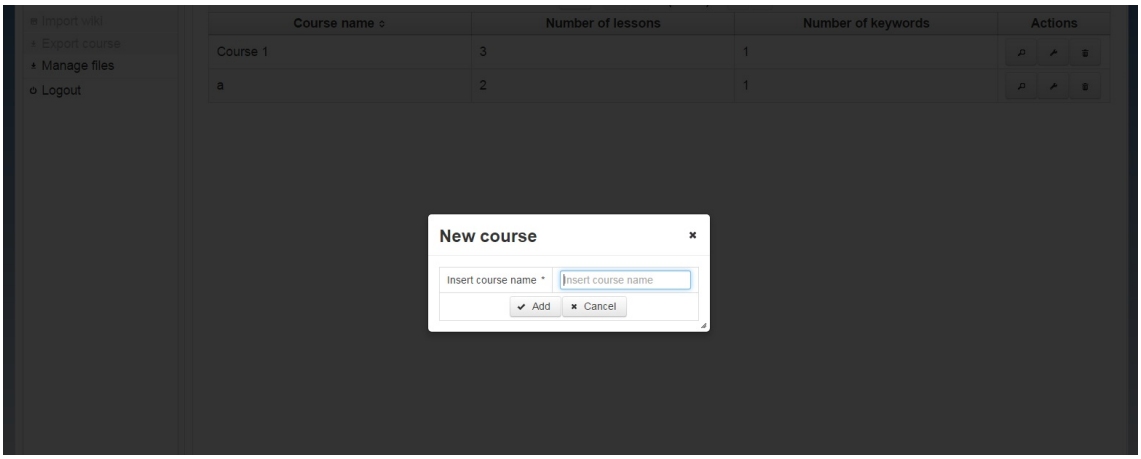
4.3.1.2 Krok 2



Obrázek 4.2: Stav aplikace po přihlášení

- **Stanoví si uživatel správný cíl?**
  - Ano, v menu je jasně vidět tlačítko Add new
- **Je akce, kterou by měl uživatel udělat, vidět?**
  - Ano, v menu je jasně vidět tlačítko Add new
- **Je akce správná?**
  - Ano, uživateli se zobrazí dialog pro vyplnění údajů o novém kurzu
- **Rozumí uživatel zpětné vazbě?**
  - Ano

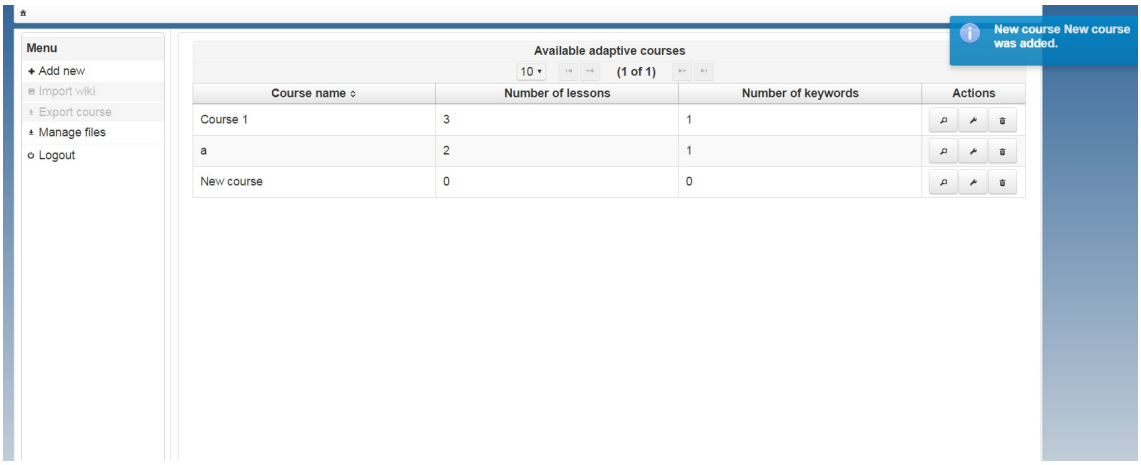
4.3.1.3 Krok 3



Obrázek 4.3: Zobrazení dialogu pro vyplnění údajů o kurzu

- **Stanoví si uživatel správný cíl?**
  - Ano, vstupní pole a tlačítko pro potvrzení je jasně vidět
- **Je akce, kterou by měl uživatel udělat, vidět?**
  - Ano, vstupní pole a tlačítko pro potvrzení je jasně vidět
- **Je akce správná?**
  - Ano, uživateli zmizí dialog, nový kurz se objeví v tabulce a vypíše se hláška o úspěšném přidání kurzu
- **Rozumí uživatel zpětné vazbě?**
  - Ano, v případě špatně vyplněných údajů není uživateli umožněno pokračovat a špatně vyplněné pole se označí

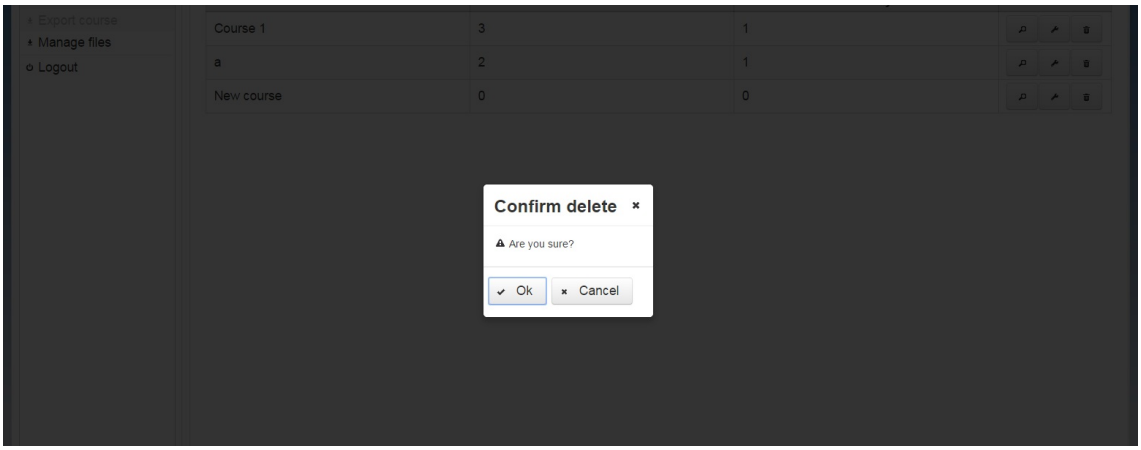
4.3.1.4 Krok 4



Obrázek 4.4: Zobrazení nového kurzu v tabulce

- **Stanoví si uživatel správný cíl?**
  - Ano, nový kurz byl vytvořen je potřeba ho upravit
- **Je akce, kterou by měl uživatel udělat, vidět?**
  - Ano, avšak prvotní pokus o vyhledání akce úpravy proběhne v hlavním menu
- **Je akce správná?**
  - Ano, uživateli se zobrazí dialog pro editaci údajů
- **Rozumí uživatel zpětné vazbě?**
  - Ano, v případě špatně vyplněných údajů není uživateli umožněno pokračovat a špatně vyplněné pole se označí

4.3.1.5 Krok 5

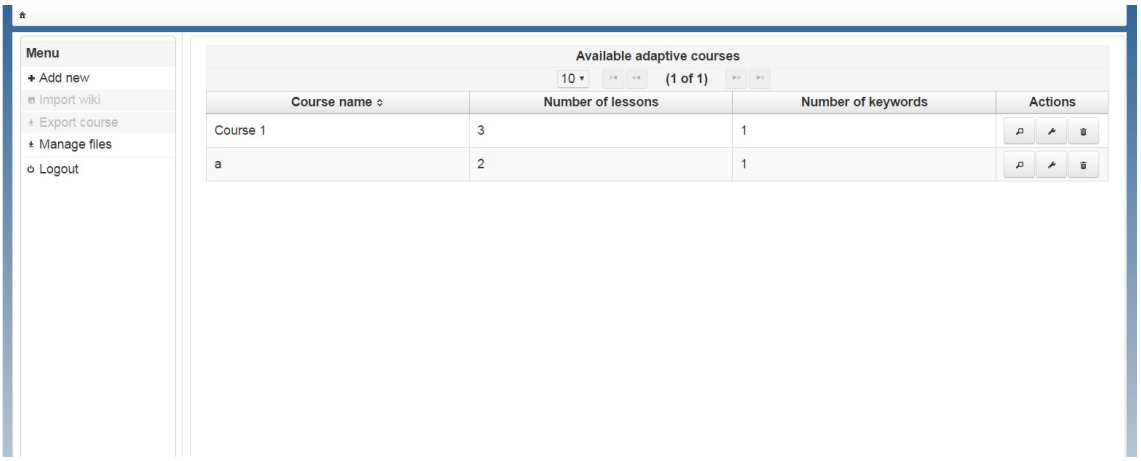


Obrázek 4.5: Zobrazení dialogu pro editaci

- **Stanoví si uživatel správný cíl?**
  - Ano.
- **Je akce, kterou by měl uživatel udělat, vidět?**
  - Ano, vstupní pole a potvrzovací tlačítko je jasně viditelné
- **Je akce správná?**
  - Ano, uživateli se zobrazí tabulka s upraveným kurzem a zobrazí se hláška o úspěšné editaci
- **Rozumí uživatel zpětné vazbě?**
  - Ano, v případě špatně zadaných údajů není uživateli umožněn postup dál a špatně vyplněné pole se označí

4.3.2 Otevření editace výukových objektů v nově vytvořené stránce

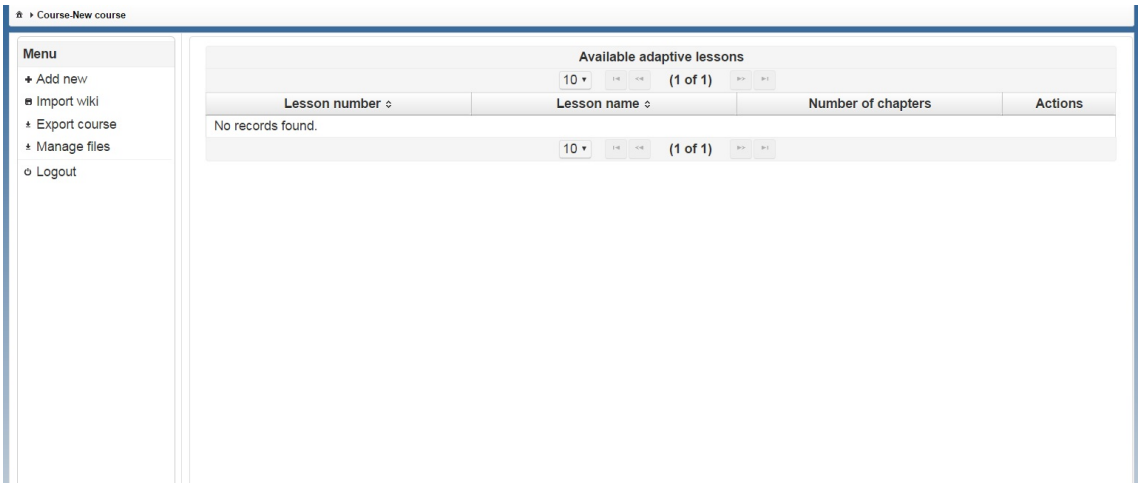
4.3.2.1 Krok 1



Obrázek 4.6: Stav aplikace po přihlášení

- **Stanoví si uživatel správný cíl?**
  - Ano, je ale pro něj nutné si uvědomit posloupnost doménových objektů v modelu
- **Je akce, kterou by měl uživatel udělat, vidět?**
  - Ano, jedná se o poslední tlačítko s ikonou lupa
- **Je akce správná?**
  - Ano, uživateli se zobrazí nová tabulka s lekcemi kurzu
- **Rozumí uživatel zpětné vazbě?**
  - Ano

4.3.2.2 Krok 2



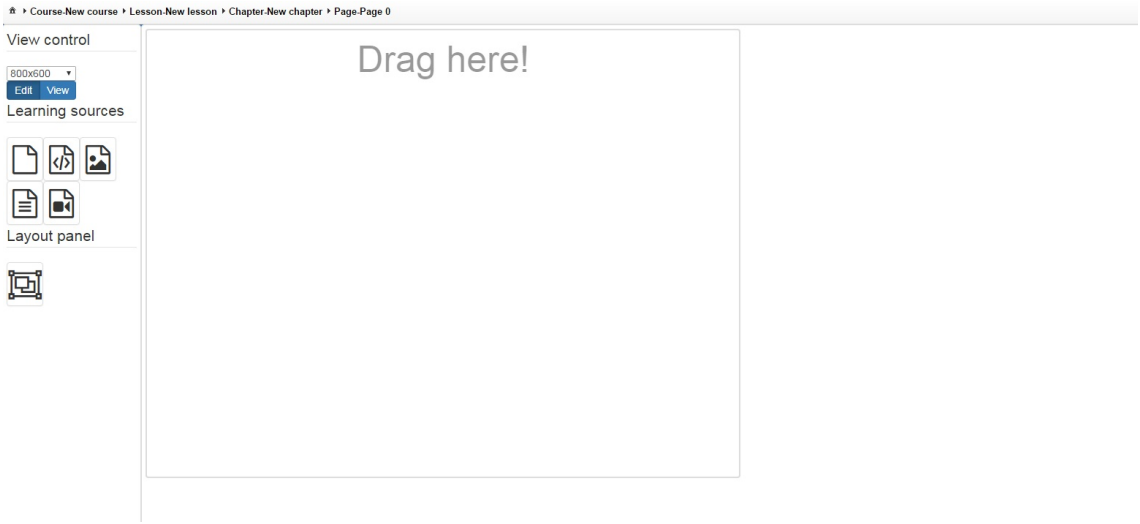
Obrázek 4.7: Zobrazení prázdné tabulky lekcí

- **Stanoví si uživatel správný cíl?**
  - Ano, není možné pro něj pokračovat stejným způsobem a pro postoupení do další vrstvy je potřeba vytvořit novou kapitolu.
- **Je akce, kterou by měl uživatel udělat, vidět?**
  - Ano, byla součástí předchozího testovacího úkolu.
- **Je akce správná?**
  - Ano, uživateli se objeví nová kapitola v tabulce po vyplnění potřebných informací do dialogu a potvrzení
- **Rozumí uživatel zpětné vazbě?**
  - Ano, nyní je možné pokračovat předešlým způsobem do další vrstvy modelu.

Následující kroky jsou přeskočeny, protože jejich výsledky jsou naprosto stejné. Skládají se z posloupnosti vytváření jednotlivých vrstev doménových objektů. Stránka vždy pouze zobrazí jiná data v tabulce při každém přechodu do nižší vrstvy doménového modelu. Při přechodu z nově vytvořené stránky níže se zobrazí editor výukových objektů.

4.3.3 Vytvoření obrázkového výukového objektu s referencí na lokální kopii externího zdroje a uložení práce

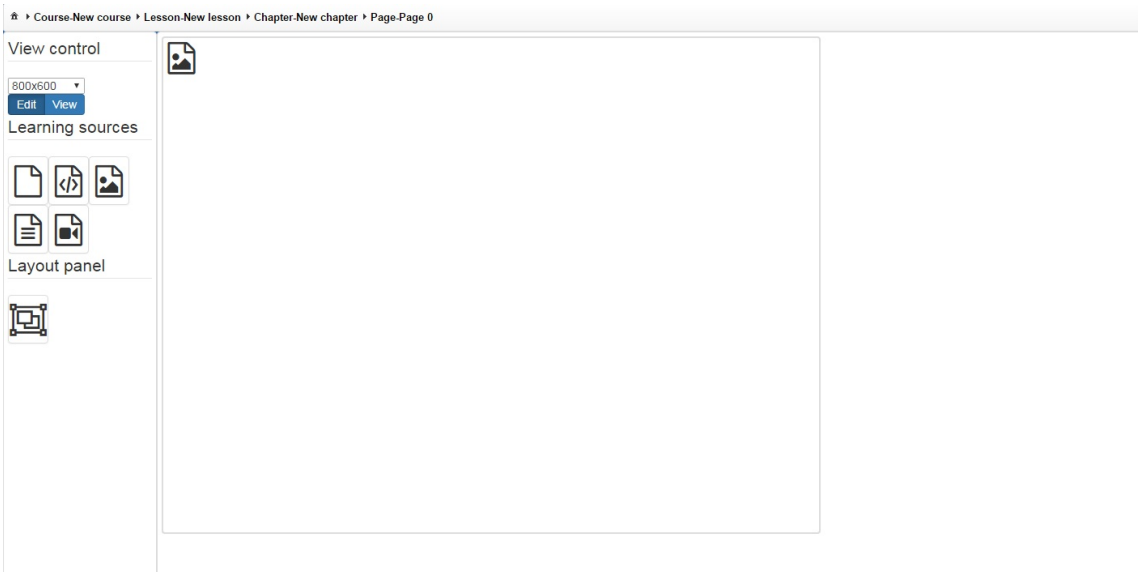
4.3.3.1 Krok 1



Obrázek 4.8: Základní stav editoru výukových objektů

- **Stanoví si uživatel správný cíl?**
  - Ano. Je potřeba vytvořit obrázkový výukový objekt.
- **Je akce, kterou by měl uživatel udělat, vidět?**
  - Ano, na stránce jsou jasně vidět ikony, které mají při najetí myši popisek. Případný vzhled ikony napovídá typu výukového objektu pod ní. Text *Drag here!* v panelu vpravo napovídá, co za akci by se mělo provést.
- **Je akce správná?**
  - Ano, ikona se zobrazí uvnitř panelu vpravo a zmizí z něj nápis.
- **Rozumí uživatel zpětné vazbě?**
  - Ano, není možné ikonu přesunout nikam jinam.

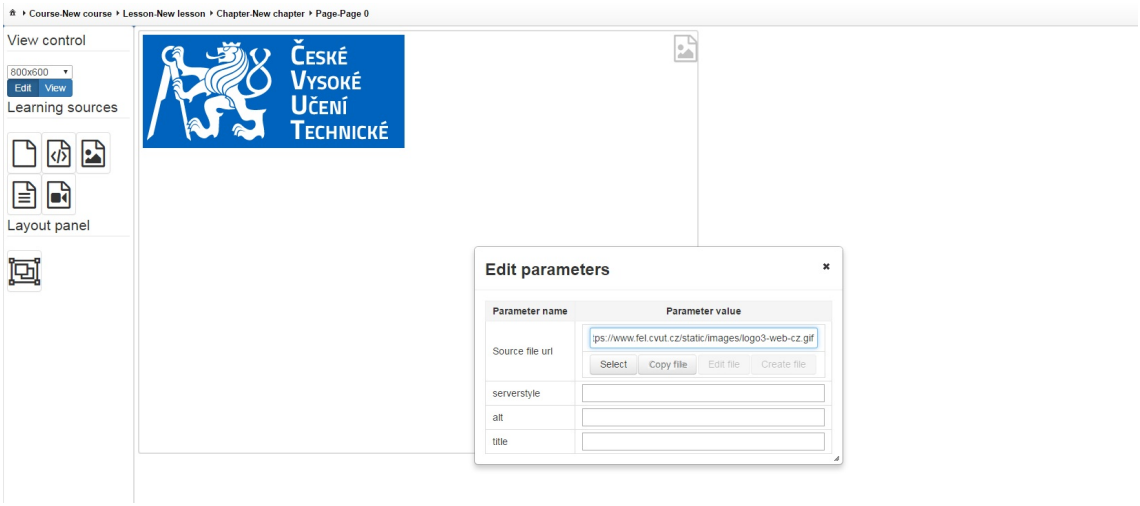
4.3.3.2 Krok 2



Obrázek 4.9: Stav editoru po vložení obrázkové ikony do panelu

- **Stanoví si uživatel správný cíl?**
  - Ano. Je potřeba obrázkovému objektu přiřadit url zdroje.
- **Je akce, kterou by měl uživatel udělat, vidět?**
  - Ne, akce je schovaná v kontextových menu obrázkového výukového objektu.
- **Je akce správná?**
  - Ano, po vložení adresy url externího zdroje se zobrazí požadovaný obrázek a ikona se přemístí vpravo
- **Rozumí uživatel zpětné vazbě?**
  - Ano, pokud je url nesprávná, zobrazí se pouze HTML obrázek reprezentující nedostupné url.

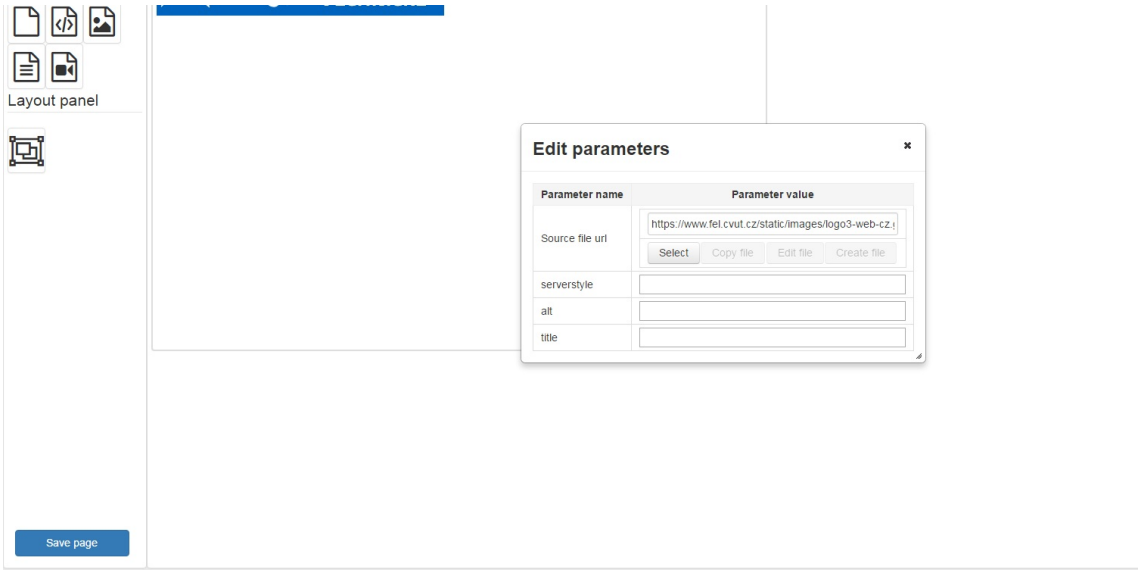
4.3.3.3 Krok 3



Obrázek 4.10: Stav editoru po zkopírování adresy do vstupního pole

- **Stanoví si uživatel správný cíl?**
  - Ano. Je potřeba vytvořit lokální kopii.
- **Je akce, kterou by měl uživatel udělat, vidět?**
  - Ano, tlačítko *Copy file* se po zkopírování URL zpřístupnilo.
- **Je akce správná?**
  - Ano, po kliknutí na tlačítko *Copy file* se změní externí adresa na lokální, vypíše se hláška o úspěšném kopírování a znepřístupní se kopírovací tlačítko
- **Rozumí uživatel zpětné vazbě?**
  - Ano, pokud je url nesprávná, nezpřístupní se tlačítko *Copy file*

4.3.3.4 Krok 4



Obrázek 4.11: Stav editoru po zkopírování adresy do vstupního pole

- **Stanoví si uživatel správný cíl?**
  - Ano. Je potřeba vytvořit uložit upravenou stránku.
- **Je akce, kterou by měl uživatel udělat, vidět?**
  - Ano, tlačítko *Save file* je jasně viditelné.
- **Je akce správná?**
  - Ano, po kliknutí na tlačítko *Save file* se vypíše hláška o úspěšném uložení stránky
- **Rozumí uživatel zpětné vazbě?**
  - Ano.

Výsledky zbylých průchodů aplikací lze nalézt v následující tabulce, která shrnuje nalezené problémy v průběhu plnění úkolu uživatelem.

Úkol	Nalezené problémy
Import nové lekce	Import sice nepřijme prázdný obsah, ale v případě, že je wiki zdroj nevalidní, tedy nelze z něj vytvořit žádnou lekci chybí chybová hláška o neúspěchu importu
Vytvoření BorderLayoutu s textových výukovým objektem	Pro uživatele může bt matoucí, že po vložení ikony reprezentující layout, je potřeba mu ještě nastavit jeden z dostupných typů layoutu
Export XML reprezentace kurzu	Bylo by dobré přidat popisek pro pole s prefixem pro url, kde by bylo vysvětleno, k čemu přesně dané pole slouží a jak nahrazuje prefix url

Tabulka 4.1: Shrnutí výsledků zbylých průchodů

4.3.4 Změny rozhraní na základě výsledků testování

Testování odhalilo pouze malé množství drobných nedostatků, které byly vyřešeny změnou vzhledu nebo umístění komponent. Hlavním nedostatkem je asi použití kontextového menu pro editaci výukových objektů. Tento problém lze vyřešit poskytnutím uživatelské příručky uživatelům, kde bude zmíněná právě tato funkcionalita. Díky tomu by měl být seznámený uživatel schopný používat aplikaci bez problémů.

Závěr

Cílem této diplomové práce bylo vytvořit webový designer adaptivních výukových materiálů pro Adaptivní kurz Javy. Tento projekt vznikl v roce 2012 na Fakultě informačních technologií pod vedením pana Ing. Martina Balíka, Ph.D. a vývojáře pana Ing. Miroslava Hořejšího. Díky tomuto projektu byl umožněn adaptivní přístup k výuce programování, avšak kvůli chybějícímu editoru adaptivních výukových materiálů byla práce s materiály náročná. Proto vznikla tato diplomová práce, která tento nedostatek řeší a usnadňuje tak správu učebních materiálů i přehlednost obsahu jednotlivých kurzů. Aby bylo vytvoření webového editoru vůbec možné, bylo potřeba se podrobněji seznámit s detaily cílového systému pro adaptivní materiály teoreticky. Proto byly adaptivní hypermediální systémy zkoumány z historického hlediska. Byl zkoumán samotný adaptační proces, model uživatele i nejrůznější adaptační techniky a jejich rozdělení. Tato teoretická část posloužila jako náhled do problematiky a díky ní bylo možné specifikovat základní vlastností, které by tento webový editor měl mít jako je struktura upravovaných materiálů, způsoby adaptace v cílové aplikaci a požadavky na funkčnost editoru. Následně byla provedena analýza již dostupných způsobů řešení, které se týkaly tvorby adaptivních výukových materiálů, adaptivních systémů a samotné cílové aplikace. Pro implementaci bylo použito množství technologií jako Spring, Hibernate, Primefaces, JSF a samozřejmě Java EE. Samotná realizace editoru je v práci podrobně rozepsaná. Neřeší jen samotnou implementaci, ale i nastavení a způsob, jakým se editor dokáže přizpůsobit změně doménového modelu. Na závěr bylo provedeno testování pomocí JUnit testů a kognitivního průchodu, které neodhalilo žádné závažné chyby v aplikaci. Ty menší byly opraveny změnou vzhledu, nebo umístěním komponent. Jako větší nedostatek se dá považovat použití kontextového menu pro editaci výukových objektů. Tento nedostatek je vyřešen uživatelskou příručkou, která informuje o této funkci. Proto lze říci, že vytvořený webový editor splnil zadané požadavky a může být využit jako doplňující editor pro Adaptivní kurz Javy, jak bylo původně zamýšleno.

Literatura

[1] Brusilovsky, P.: *The Adaptive Web: Methods and Strategies of Web Personalization*. Heidelberg: Springer Verlag, 2007, ISBN 978-3-540-72079-9.

[2] Tseng, S.-S.; Su, J.-M.; Hwang, G.-J.; aj.: *An Object-Oriented Course Framework for Developing Adaptive Learning Systems*. 2008, [cit. 2015-02-15]. Dostupné z: `http://www.ifets.info/journals/11_2/14.pdf`

[3] Blackboard Inc.: *Blackboard Help*. 2014, [cit. 2015-02-15]. Dostupné z: `https://help.blackboard.com/en-us/Learn/9.1_SP_12_and_SP_13/Instructor/070_Course_Content/000_Creating_Content/050_Editing_and_Managing_Course_Areas_and_Content`

[4] Weber, G.; Kuhl, H.-C.; Weibelzahl, S.: *Developing Adaptive Internet Based Courses with the Authoring System NetCoach*. [cit. 2015-02-15]. Dostupné z: `http://wwwis.win.tue.nl/ah2001/papers/GWeber-UM01.pdf`

[5] Cristea, A.; Aroyo, L.: *Adaptive Authoring of Adaptive Educational Hypermedia*. [cit. 2015-02-15]. Dostupné z: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.65.9557&rep=rep1&type=pdf`

[6] Hořejší, I. M.: *Adaptivní kurz Javy*. 2012, [cit. 2016-05-08]. Dostupné z: `https://dip.felk.cvut.cz/browse/pdfcache/horejmi4_2012dipl.pdf`

[7] Pivotal Software: *Spring framework documentation*. [cit. 2016-05-04]. Dostupné z: `http://spring.io/docs/reference`

[8] Walls, C.: *Spring in action*. Manning Publications Co. Greenwich, CT, USA, třetí vydání, 2011.

[9] RedHat: *Hibernate ORM dokumentace*. [cit. 2016-05-04]. Dostupné z: `http://hibernate.org/orm/`

[10] Bauer, C.; King, G.: *Java Persistence with Hibernate*. Třetí vydání, 2006.

[11] Oracle Inc.: *JavaServer Faces dokumentace*. [cit. 2016-05-04]. Dostupné z: https://docs.oracle.com/javaee/7/tutorial/jsf-intro.htm

[12] Geary, D.; Horstmann, C. S.: *Core JavaServer Faces*. Třetí vydání, 2010.

[13] Oracle Inc.: *JavaBeans dokumentace*. [cit. 2016-05-04]. Dostupné z: http://www.oracle.com/technetwork/articles/javaee/spec-136004.html

[14] Primefaces: *Primefaces user guide 5.1 First edition*. [cit. 2016-05-04]. Dostupné z: http://www.primefaces.org/docs/guide/primefaces_user_guide_5_1.pdf

[15] Oracle Inc.: *JAXB dokumentace*. [cit. 2016-05-04]. Dostupné z: http://www.oracle.com/technetwork/articles/javase/index-140168.html

[16] Oracle Corporation: *GlassFish Server Documentation*. 2013. Dostupné z: https://glassfish.java.net/documentation.html

[17] *JQuery dokumentace*. [cit. 2016-05-04]. Dostupné z: https://api.jquery.com/

[18] *JQueryUI dokumentace*. [cit. 2016-05-04]. Dostupné z: http://api.jqueryui.com/

[19] Oracle Corporation: *MySQL 5.6 Reference Manual*. 2015. Dostupné z: http://dev.mysql.com/doc/refman/5.6/en/index.html

[20] Pivotal Software: *Spring Data query generation*. [cit. 2016-05-05]. Dostupné z: http://docs.spring.io/spring-data/jpa/docs/current/reference/html/

[21] Varaksin, O.; Caliskan, M.: *PrimeFaces Cookbook*. Packt Publishing Ltd, 2013, [cit. 2016-05-04].

Příloha **A**

Seznam použitých zkratek

AES Adaptive Hypermedia Educational System

AHS Adaptive Hypermedia Systems

AHES Adaptive Hypermedia Educational System

API Application program interface

GUI Graphical user interface

HTML Hypertext markup language

Java EE Java Enterprise Edition

JAXB Java Architecture for XML Binding

JPA Java Persistence API

JPQL Java Persistence Query Language

JS Javascript

JSF JavaServer Faces

JSON JavaScript Object Notation

MALS Modular Adaptive Learning System

ORM Object-Relational mapping

SQL Structured query language

URL Uniform Resource Locator

XML Extensible markup language

XSD XML schema definition

Příloha B

Instalační příručka

Aplikace byla vyvinuta v IDE Netbeans na platformě Windows. Proto bude příručka zaměřená na zprovoznění aplikace právě pod prostředím Netbeans. Pro běh aplikace je potřeba mít v Netbeans přístupné JDK 1.6, Maven build framework a Glassfish server 4.1.

B.1 Příprava aplikace

Nejprve je potřeba vyřešit závislosti aplikace na externích knihovnách. O většinu závislostí se postará framework Maven. Jediné závislosti, které je potřeba vyřešit manuálně jsou vazby na knihovnu ASF frameworku. Potřebné *jar* soubory je možné nalézt ve složce *lib* projektu. Po nastavení potřebných vazeb je ještě nutné nastavit správné připojení do databáze. K tomu účelu je soubor *WEB-INF/jdbc.properties*, kde jsou vypsané všechny parametry, které je potřeba změnit. Po tomto nastavení je možné aplikaci zkompilovat a spustit.
Aplikace má pouze jednoho uživatele *admin* s přístupovým heslem *admin*.

Příloha C

Uživatelská příručka

Po spuštění se zobrazí přihlašovací obrazovka. Pro přihlášení zadejte přihlašovací jméno *admin* a heslo *admin*.

C.1 Hlavní stránka

Hlavní stránka se skládá z několika částí. Menu, tabulky s přehledem dostupných kurzů a breadcrumb panelu pro navigaci. V menu lze nalézt následující:

Add new Slouží pro přidávání nových objektů ve vrstvách doménového modelu

Import wiki Slouží pro import lekce z wiki zdroje do označeného kurzu

Export course Slouží pro export označeného kurzu

Manage files Správa vytvořených souborů v aplikaci

Logout Odhlášení uživatele

Po vytvoření prvního kurzu se zobrazí v tabulce a u něho jsou tlačítka pro procházení, editace a smazání kurzu. Další vrstvy vypadají podobně akorát se liší o jaký objekt se jedná. Jedinou výjimkou je tvorba adaptačních pravidel, která pro vytvoření potřebuje informace o lekci a stránce, kde se nachází, a proto je funkce přidání pravidla dostupná pouze v editoru výukových objektů.

C.2 Editor výukových objektů

Editor se zobrazí při pokusu o procházení stránky. Skládá se z ovládacích prvků vlevo a editačního panelu vpravo. Pro vložení nového výukového objektu, je potřeba přetáhnout ikonu reprezentující příslušný typ objektu do oblasti pro editaci. Pro úpravu jeho parametrů je nutné vyvolat kontextové

menu pomocí pravého tlačítka myši. Pro vytvoření více výukových objektů na stránce je nutné použít ikonu reprezentující layout a nastavit ji jeden z dostupných (v současnosti jeden) layoutů pomocí kontextového menu. Pak je možné na stránku přidávat více objektů. Pro přidání více objektů než podporuje vybraný layout se použije stejný způsob jako s první layoutem. Pro změnu rozlišení editační části se použije ovládací prvek pro rozlišení. Pro zobrazení předpokládaného vzhledu v cílové aplikaci je v panelu ovládacích prvků tlačítko *View*.

Příloha D

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `documents` ..... obsah práce
  - `mockups` ..... sada návrhů GUI
  - `text` ..... text práce ve formátu PDF
- `src` ..... editor a potřebné projekty pro nasazení