VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2015

Marek Novák

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Marek Novák**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: SCOVECO, s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Mgr. Jiří Dvorský, Ph.D.**

Konzultant bakalářské práce: Ing. Zdeněk Velart, Ph.D.

Datum zadání: 01.09.2014
Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2015 ..... Marek Novák

Rád bych na tomto místě poděkoval celému kolektivu společnosti SCOVECO, s.r.o. za vytvoření příjemného pracovního prostředí a hlavně konzultantovi mé práce Ing. Zdeňku Velartovi, Ph.D., za odborné vedení a poskytování cenných rad v průběhu mé bakalářské praxe.

Abstrakt

Tato bakalářská práce se zaměřuje na popis tvorby portálového řešení založené na platformě Liferay Portal s cílem popsat technologie použité při vývoji reálného projektu a poskytnout čtenáři přehled o základní konfiguraci jednotlivých technologií. V úvodu práce je uveden současný stav informačního systému a důvod přechodu na novější platformu. Další část se zabývá jednotlivými technologiemi použitými při vývoji softwarového řešení a na závěr práce je rozebrána vybraná část z projektu s poskytnutými snímky z obrazovky. U každé ukázky je popsán postup při její tvorbě s vysvětlením jednotlivých částí zdrojového kódu.

Klíčová slova: webová aplikace, Liferay Portal, JSF, PrimeFaces

Abstract

This bachelor thesis is focused on description of creating portal solution based on Liferay Portal with the goal to describe technology used in development of the real project and provide some overview to the reader about basic configuration of the technologies. Current state of the information system and the reason to migrate to new platform are mentioned at the beginning of the thesis. In the next part there is the overview of used technologies in software development and chosen part of the project with some screenshots is described in detail in the final part. At the every example there is described approach with its explanation of piece of the source code.

Keywords: web application, Liferay Portal, JSF, PrimeFaces

Seznam použitých zkratek a symbolů

JSP	–	JavaServer Pages
JSF	–	JavaServer Faces
JSTL	–	JSP Standard Tag Library
MVC	–	Model-View-Controller
EL	–	Expression Language
JVM	–	Java Virtual Machine
POM	–	Project Object Model
CSS	–	Cascading Style Sheets
ŠkVP	–	Školský vzdelávací program
ŠVP	–	Štátny vzdelávací program
FOP	–	Formatting Objects Processor
XML	–	Extensible Markup Language
XLST	–	Extensible Stylesheet Language Transformations
SSH	–	Secure Shell
HQL	–	Hibernate Query Language
SQL	–	Structured Query Language
FTL	–	FreeMarker Template Language

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Odbornou praxi jsem vykonával ve společnosti Scoveco, s.r.o. sídlící v Ostravě na ulici Františka Lýska 1605/3. Její kancelář se nachází na ulici Studentská 6202/17 v budově podnikatelského inkubátoru, který byl založen v areálu Vysoké školy báňské – Technické univerzity Ostrava.

Jde o mladou společnost založenou v roce 2011, která se zaměřuje na poskytování konzultací a školení v oblasti řízení procesu a kvality vývoje softwaru, dále na vývoj softwarových systémů a cloudových aplikací pro různé platformy.

1.1 Obsah praxe

Pracoval jsem pod vedením pana Ing. Zdeňka Velarta, Ph.D., který mě postupně seznamoval s jednotlivými technologiemi používanými k aktualnímu projektu, k němuž jsem byl později přiřazen. Jednalo se o rozsáhlý projekt, kde se část vyvíjela zde a další ve společnosti DWC Slovakia a.s. na Slovensku.

Jednalo se o zakázku pro Ministerstvo školství, vedy, výskumu a športu SR, které požadovalo vytvoření informačního systému pro mateřské, základní a střední školy s cílem poskytnout elektronický přístup za pomocí nejmodernějších technologií k tvorbě školského vzdělávacího programu (ŠkVP), učebnímu plánu a také přípravě na vyučování s elektronickou podporou domácí přípravy na vyučování pro samotné žáky. Systém je určený pro potřeby učitelů, odborné veřejnosti, žákům a jejich rodičům. Celý projekt je financován z fondů evropské unie, státního rozpočtu a z vlastních zdrojů.

V současnosti ministerstvo školství provozuje vzdělávací portál pro střední školy a druhý stupeň základních škol, který je přístupný školám, žákům, učitelům a veřejnosti. Ten poskytuje uživatelům vzdělávací materiály pro zefektivnění a zatraktivnění výuky. Slouží také k přípravě výukových materiálů, k práci se žáky během výuky a ke kontrole jejich domácích úloh. Portál je uzpůsobený pro používání na počítačích prostřednictvím prohlížečů. Nový systém by měl být optimalizován pro dnes velmi rozšířené tablety při zachování veškeré funkcionality výukových materiálů (videa, animace, simulace, prezentace, ilustrace, 3D modely, obrázky, interaktivní cvičení, lekce). Jde hlavně o přizpůsobení ovládání pro dotyková zařízení, které využívají různá gesta, a umožnění zobrazení obsahu na všech platformách, kde nemusí být přítomen Adobe flash player. [15]

2 Použité technologie

Náplní mé praxe se stalo programování v Javě, při kterém jsem přišel do styku s technologiemi Hibernate, Liferay Portal, FreeMarker, DocBook a Git. Se všemi technologiemi jsem se setkal poprvé a bylo potřeba, abych se naučil základy práce s nimi a mohl se tak začít podílet na projektu. Prvním krokem bylo seznámení se s Hibernate, na kterém staví Liferay Portal. Na zaučení jsem dostal jednoduchý úkol, který se týkal objektově-relačního mapování.

2.1 Hibernate

Pokud uchováváme data v databázi, se kterými budeme dále pracovat v aplikaci, potřebujeme je převést na objekty. Hibernate mapuje databázové tabulky na objekty a poskytuje vlastní dotazovací jazyk HQL nad objekty, který skrývá před programátorem SQL dotazy. To usnadňuje programátorovi vývoj aplikace, šetří čas bez nutnosti opakovaně vytvářet spojení k databázi při každém dotazu a následnému mapování do objektu. Hlavní rozdíl mezi těmito jazyky je, že HQL pro dotazy používá název třídy místo názvu tabulky a název třídního atributu místo názvu sloupce tabulky.

Tento framework je určený pro vývoj aplikací v Javě, který slouží k zajištění persistence dat, což znamená, že data nám zůstávají uložena i po ukončení aplikace. U persistentních tříd jsou zachovány vlastnosti objektově-orientovaného programování, které zahrnují dědičnost, polymorfismus, asociaci, kompozici a kolekce.

Nabízená podpora databází je široká. Mezi nejpoužívanější patří Oracle 11g, Microsoft SQL Server, MySQL a PostgreSQL. Napsané aplikace je tak možné snadno přenášet mezi různými databázemi jen s minimálními úpravami v konfiguraci. [1]

2.1.1 Kešování

Časově nejnáročnější operací je přístup k persistentním datům. Hibernate se tak snaží omezovat počet zaslaných dotazů na databázi kešováním. [2]

Kešování první úrovně

Tato úroveň souvisí s navázaným spojením do doby, než objekt udržující spojení zanikne. Prochází ní všechny transakce, kde se každá z nich provede pouze jednou. Stejný dotaz nebude zaslán na databázi, ale načte se z keše.

Kešování druhé úrovně

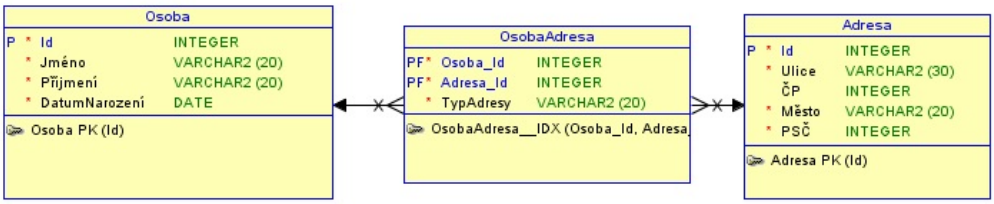
Jde o volitelnou techniku, kdy se kešují objekty, které jsou poté přístupné celé aplikaci. Například pro zobrazení nějakých dat se zašle dotaz na první úroveň. Pokud zde není požadovaný objekt, vyhledá se v druhé úrovni. V případě nalezení se objekt uloží do první úrovně a vrátí se výsledek. Když neobsahuje ani jedna úroveň tento objekt, vykoná se dotaz nad databází. Výsledek dotazu se uloží do obou úrovní a následně se vrátí objekt.

Pokud se provedou změny v databázi, je doporučené aktualizovat keš, abychom dostali platné záznamy. [2]

2.1.2 Mapování tříd

Hibernate využívá specifikaci javax.persistence API [3], kterou dále rozšiřuje o vlastní implementaci těchto tříd. Mapovaní se provádí pomocí mapovacího souboru hbm.xml, nebo pomocí anotací (ukázka č. 1). Oba způsoby poskytují Hibernate metadata k tomu, aby věděl, jak načíst a uchovávat data v persistentních třídách. V následujícím části ukázky kódu budou použity anotace.

Příklad 2.1
Mým prvním úkolem bylo se naučit základy tvorby dotazů nad objekty a jejich mapování s tím, že tyto znalosti dále využiji při spolupráci na projektu, kde se pro přístup k datům používalo objektově-relační mapování. Zadání tvořily tři jednoduché tabulky (obr. 1) `Osoba, Adresa a OsobaAdresa`. Tabulka `OsobaAdresa` představuje vazbu M:N mezi tabulkami `Osoba a Adresa`. U osoby se může evidovat jak adresa bydliště, tak i například adresa zaměstnaní. Zároveň na jedné adrese může být více osob.



Obrázek 1: Relační model příkladu

Nad těmito tabulkami jsem provedl dotazy:

1. setřídit osoby podle příjmení
2. vypsat všechny osoby a všechny jejich adresy
3. vypsat osoby a města, ve kterých mají domov

Mapování entity je znázorněno na ukázce třídy `Osoba`, kde každá třída musí obsahovat get a set metody pro všechny své atributy. Navíc u každého atributu, nebo jeho get metody, se uvádí anotace. Jednotlivé anotace se nachází v balíčku javax.presistence.

- @Entity - Deklaruje třídu jako entitu.
- @Table - Umožňuje definovat podrobnosti o tabulce jako například jméno. Pokud není anotace uvedena, Hibernate použije název třídy jako výchozí jméno tabulky.
- @Id - Označuje atribut třídy, který má být primárním klíčem.

- @GenericGenerator - Uživatelsky definovaný generátor, který umožní definovat strategii generování primárního klíče.
- @GeneratedValue - Specifikuje strategii generování primárního klíče, ve výchozím stavu je nastavena na AUTO. V mém příkladě jsem k tomu využil GenericGenerator.
- @Column - Označuje mapovaný atribut třídy a v parametrech anotace je možné upřesnit vlastnosti atributu v tabulce (např.: název sloupce, délku typu).

```
@Entity
@Table(name = "Osoba")
public class Person implements Serializable{
    private long personId;
    private Set<PersonAddress> relationship = new HashSet<PersonAddress>(0);

    public Person() {
    }

    @Id
    @GenericGenerator(name = "generator", strategy = "increment")
    @GeneratedValue(generator = "generator")
    @Column(name = "Osoba_ID")
    public long getUserId() {
        return userId;
    }
}
```

Výpis 1: Třída Osoba

■

2.2 JavaServer Pages

JSP slouží ke psaní dynamických stránek. Tento framework je navrhnutý kolem architektury MVC, která rozděluje aplikaci do třech částí:

- Model - Reprezentuje data a business logiku aplikace (JavaBean).
- View - Zobrazuje uživatelské rozhraní (JSP stránka).
- Controller - Prostředník, který má na starosti tok událostí v aplikaci a propojuje obě komponenty (Servlet).

Soubory .jsp, což jsou textové soubory, se při prvním požadavku na zobrazení převedou servletovým kontejnerem na servlet (soubor .java), následně se zkompiluje do souboru s příponou .class. Při dalším volání servletu se volá metoda service(), která vrací aktuální servlet. Pouze při změně .jsp souboru dochází k rekompilaci. [9]

2.2.1 Životní cyklus

Životní cyklus je proces, ve kterém prochází webová stránka fázemi od zaslání požadavku až k jejímu zobrazení [12]. V následujících fázích se mohou provádět námi požadované úlohy:

- Kompilace - provádí se, pokud se změnil .jsp soubor, nebo pokud nebyl ještě zkompilován
- Inicializace - typicky jde o navázání spojení k databázi či otevření souboru
- Zpracování požadavku - provádí se generování odpovědi na odeslaný požadavek a na HTTP metody
- Odstranění objektů z paměti - zde se například provádí ukončení spojení s databází, zavírají soubory

2.2.2 Servlet

Jedním z důležitých prvků pro komunikaci se serverem je servlet. Jde o třídu, která zodpovídá za komunikaci mezi klientem a serverem přes protokol HTTP. Nejčastěji vyřizovanými požadavky protokolem jsou:

- GET - zasílá požadavek na stránku na dané URL.
- POST - odesílá data na server.

Psaní HTML stránky je pomocí servletů zdlouhavé, proto se využívají technologie, které schovávají část kódu za tagy. Na základě servletu jsou postaveny JSP a JSF, které při požadavku vygenerují patřičný Java kód. Zdrojová stránka je tak podstatně přehlednější díky redukovanému množství psaných znaků [10].

Abychom mohli webovou aplikaci sestavit, je nutné ji nakonfigurovat a to v souboru web.xml nebo pomocí anotací. Uvádí se zde název třídy servletu a na jaké URL je servlet mapován. Tím se zajistí, že servlet bude přístupný z prohlížeče. Poté se za pomoci nástrojů Ant nebo Maven vytvoří komprimovaný soubor .war, který je následně možné nasadit na server.

2.2.3 JavaBean

V rámci MVC architektury obstarává aplikační logiku JavaBean. Je to opakovatelně použitelná programová komponenta, implementovaná jako serializovatelná Java třída s výchozím bezparametrovým konstruktorem. Třída obsahuje pro každou svou proměnnou get a set metodu, které dodržují velbloudí zápis. Například pro proměnnou jmeno bude metoda get ve tvaru getJmeno(). [11]

2.2.4 Skriplet

Dříve se do HTML stránky, mezi značky <% %> vkládaly sekce Java kódu, které se nazývají skriplety. Dovolují nám vkládat do stránky dynamický obsah, ale tento způsob tvorby webových stránek vedl k tomu, že se kód stával méně udržitelný a hůře opravitelný. Preferovaný způsob tvorby stránek je pomocí jazyka EL a JSTL tagů.

2.2.5 EL

Jazyk EL se používá k přístupu k datům z requestu, session, stránky a aplikace. Podporuje následující operace:

- logické (and, &&, or, ||, not, !, empty)
- aritmetické (+, -, *, /, div, %, mod)
- relační (==, eq, !=, ne, <, lt, >, gt, <=, le, >=, ge)
- podmíněný operátor

Pro přístup k datům jsou podporovány dva způsoby vyhodnocení výrazů, a to bezprostřední a odložený. U bezprostředního vyhodnocení je výraz vyhodnocen a výsledek vrácen v okamžiku, kdy je stránka prvně vykreslena. Tyto výrazy používají syntaxi `${}`. Odložené vyhodnocení výrazu umožňuje vyhodnotit výraz později v průběhu životního cyklu než je stránka vykreslena. K tomuto vyhodnocování se užívá syntaxe `#{}`. [14]

2.2.6 JSP Standard Tag Library

Ve výchozím stavu JSP obsahuje pouze tagy pro vkládání a přesměrování stránek, pro přístup k třídním proměnným JavaBeanu a definování samotného JavaBeanu. Tato knihovna rozšiřuje funkcionalitu JSP o:

- cykly, větvení, práce s URL a s proměnnými
- vytváření a manipulaci XML a XSLT
- formátování a zobrazování textu, data, času podle zvoleného jazyka
- práci s databází
- funkce pro manipulaci s řetězci

.

2.3 JavaServer Faces

JSF je framework určený k vytváření uživatelského rozhraní. Má stejnou strukturu jako JSP, ze kterého vychází, s tím rozdílem, že pro zobrazení využívá technologii Facelets a vytvářené xhtml soubory jsou čisté XML. [10]

2.3.1 Životní cyklus

JSF se liší také od JSP zpracováním stránek, které prochází více fázemi:

- Vytvoření stromu komponent
- Nastavení hodnot z požadavku k jednotlivým instancím komponent
- Provedení validace mezi hodnotami instancemi komponent a hodnotami z požadavku
- Nastavení zvalidovaných hodnot z komponent do Managed beanů
- Zpracování událostí vyvolaných uživatelem
- Vykreslení komponent na základě průchodu tagy

2.3.2 Facelet

Ve verzi JSF 2.0 došlo ke změně frameworku z JSP na Facelets. Ten rozšiřuje funkcionalitu o další tagy a podporuje tvorbu stránek ze šablon, které umožňují znovu využít kód stránky (ukázka č. 2). Vytořená stránka je tak tvořena tagy a JavaScriptovými funkcemi bez Java kódu, jak je tomu u JSP.

Šablony

- ui:insert - definuje jaký obsah bude umístěn do šablony. Uvádí se v souboru, který se bude skládat z dalších částí.
- ui:define - definuje obsah vkládaný do šablony
- ui:include - vkládá obsah jedné stránky do druhé
- ui:composition - načítá šablonu, kterou předáme atributu template. Bez atributu umožňuje vkládat skupinu komponent do xhtml stránky.

```
<html xmlns="http://www.w3.org/1999/xhtml"
            xmlns:ui="http://java.sun.com/jsf/facelets" >
</html>
```

Výpis 2: Deklarace facelet tagu

2.3.3 Managed bean

JavaBean třídy jsou v JSF nahrazeny Managed beany, které rozšiřují jejich funkcionalitu o možnost definování, jak dlouho bude instance třídy existovat. Rozlišujeme následující rozsah působností:

- @RequestScoped - bean třída žije po dobu HTTP požadavku a po jeho dokončení třída zaniká
- @NoneScoped - bean třída se vytvoří před vyhodnocením EL výrazu a po vyhodnocení třída zaniká
- @SessionScoped - bean třída se vytvoří na základě HTTP požadavku zahrnující tuto třídu v navázaném spojení a po ukončení spojení třída zaniká
- @ApplicationScoped - třída žije po dobu běhu webové aplikace (od prvního HTTP požadavku, který zahrnuje tuto třídu, až po ukončení webové aplikace)
- @ViewScoped - bean třída žije po dobu, co uživatel a webová stránka mezi sebou vzájemně komunikují
- @CustomScoped - můžeme definovat vlastní třídu působnosti, která bude uchovávat managed beany v mapě. Tyto třídy potom žijí po dobu, co jsou uloženy v mapě.

2.3.4 Lokalizace

V JSF máme možnost používat dynamické texty, které nám umožňují aplikovat textové zprávy definované v souboru na grafické komponenty a statusové zprávy. Další možností je jejich použití pro lokalizaci webové aplikace. Texty se umisťují do souboru s příponou properties.

Příklad 2.2
Budeme chtít lokalizaci v českém jazyce. Vytvoříme soubor messages_cs.properties, kde tlačítku k odeslání formuláře přiřadíme text: `submitButton=Odeslat`. Následně se do konfiguračního souboru faces-config.xml (výpis č. 3) přidají výchozí a podporované jazyky do tagu `<locale-config/>`. Cesta k souboru .properties se uvádí v `<base-name/>` a v tagu `<var/>` je proměnná, přes kterou se bude přistupovat k jednotlivým textům.

```
<application>
    <locale−config>
        <default−locale>en</default−locale>
        <supported−locale>cs</supported−locale>
    </locale−config>
    <resource−bundle>
        <base−name>com.tutorialspoint.messages</base−name>
        <var>msg</var>
    </resource−bundle>
</application>
```

Výpis 3: faces-config.xml

Na výstupu se do atributu value přiřadí text z proměnné msg.

```
<h:commandButton value="#{msg['submitButton']}" type="submit" action="#{user.goLoginPage}" />
```

Výpis 4: Použití lokalizace na výstupu

■

2.4 Primefaces

V JSF se pro tvorbu webových aplikací používají zejména frameworky grafických komponent RichFaces, IceFaces a PrimeFaces. Z nich vyniká právě PrimeFaces, které obsahuje velké množství komponent. Na stránkách projektu [5] je k dispozici podrobná dokumentace i s tutoriálem.

Jde o poměrně nový framework, jehož vývoj začal v roce 2008. Obsahuje sadu komponent pro aplikace vyvíjené v JavaServer Faces. Tyto komponenty dále využívají jQuery a AJAX, což jsou JavaScriptové technologie. V současné době se jedná o jednu z nejpoužívanějších komponent jednak díky dobře zpracovanému vzhledu a také své nenáročnosti. Používají je i světově renomované společnosti jako jsou nVidia, BMW, eBay, Intel a další.

PrimeFaces vyvíjí také grafické komponenty optimalizované pro mobilní zařízení. Jsou postaveny na jQuery Mobile, který je uzpůsoben pro dotyková zařízení.

2.4.1 Stylování grafických komponent

Stažené komponenty z oficiálních stránek mají výchozí téma, které se dá zaměnit za jiné. K dispozici jsou zdarma nebo v rámci placených edicí elite a pro. Ty se dají stáhnout ve formě .jar souboru. Poté se musí přidat mezi `<context-param/>` do POM souboru konkrétní název tématu. V případě, že si budeme chtít definovat vlastní styl na komponenty, nejjednodušší variantou je použití grafického nástroje ThemeRoller, který nevyžaduje znalosti tvorby CSS.

2.4.2 Částečné zpracování

Primefaces poskytuje mechanismus k aktualizování komponent pomocí AJAXu. K tomu je zapotřebí, aby aktualizovaná komponenta měla své id. Pokud se aktualizovaná komponenta nachází ve stejném kontejneru jako komponenta, která spustila tuto akci, nemusí kontejner mít své id. V případě, že se aktualizovaná komponenta nachází v jiném kontejneru, musí mít také kontejner své id. Nejsme pouze omezeni na aktualizování jedné komponenty, ale můžeme aktualizovat více komponent. Do atributu `update` přidáme seznam id oddělených čárkou, mezerou, nebo kombinací obou.

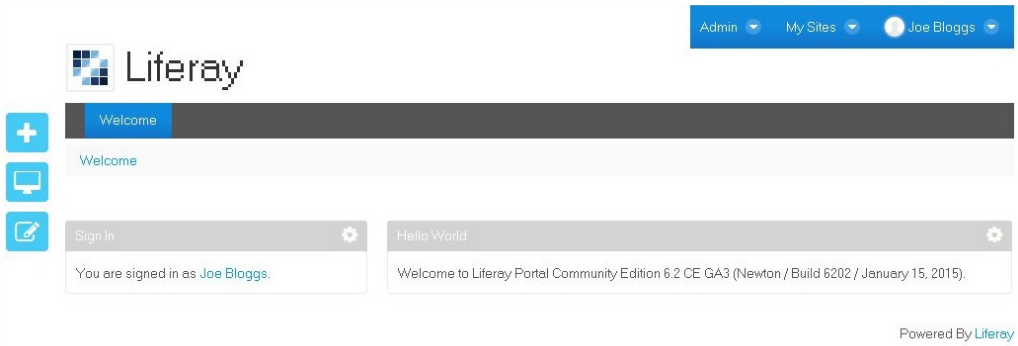
2.5 Liferay Portal

Liferay je open-source projekt určený k vývoji portletů, který ve své instalaci zahrnuje webový server Apache Tomcat a předvytvořené portlety určené k nasazení na webové stránky, bez nutnosti psaní aplikace. Pokud tedy chceme vytvořit internetové stránky, nemusíme nutně umět programovat. Portlet, což je námi implementovaná funkcionalita, se vytváří za pomoci jedné z následujících technologií: Liferay MVC, JSF, nebo Vaadin. Podporováno je také velké množství databází, z nichž nejznámější jsou: MySQL, PostgreSQL, SQL Server. [4]

Liferay je dostupný ve dvou edicích, a to v Enterprise Edition, která je placená, a v Standard Edition, jejíž licence je zdarma. Zásadní rozdíly u komerční edice jsou:

- vyšší stabilita díky většímu času stráveným při testování
- vlastní vývojové prostředí místo pluginu do eclipse
- aktualizace softwaru v rámci dané verze po dobu 4 let

Ve výchozím stavu máme k dispozici jednu stránku (obr. 2), která se po přidání dalších stránek může dále větvit do stromové struktury podobné u správce souborů. Druhou možností je přidávání stránek v podobě tabů. Vytvářené stránky mohou být veřejně přístupné nebo soukromé, záleží tedy na obsahu, který chceme publikovat.



Obrázek 2: Úvodní Liferay portal stránka

Další užitečnou vlastností je, že programátor může pracovat na změnách portletu a uživateli ponechat prozatimní funkční verzi. Liferay to řeší tak, že prováděné změny se publikují do další verze portletu. Obě verze přitom mohou běžet na stejném serveru. Novější verze se po dokončení nahradí v rámci plánované odstávky serveru nebo manuálně.

2.5.1 Portál

Portál je technologie umožňující tvorbu, publikaci a zobrazování aplikací a webového obsahu na jednom místě. Můžeme tak z centrálního prostředí spravovat vzhled stránek, měnit jazykovou lokalizaci, nastavovat přístup k jednotlivým stránkám a další.

2.5.2 Podpora mobilních zařízení

Pokud tvoříme stránky, musíme počítat s tím, že ke stránkám se může přistupovat se zařízeními o různých velikostech displejů. Po kliknutí na ikonu s displejem se zobrazí konfigurátor, ve kterém nastavíme požadované rozlišení displeje a poté se nám vykreslí stránka ve fiktivním zařízení.

Můžeme také nastavit odlišný vzhled zobrazování stránek pro mobilní zařízení a počítače, pomocí definování pravidel aplikovaných na určitou skupinu se společnými vlastnostmi. Základní informace zařízení samo o sobě poskytuje, ale může se stát, že budeme

potřebovat znát podrobnější charakteristiku. K tomuto účelu je v Liferay Marketplace, například aplikace Device Recognition Provider EE, která slouží k získání podrobných informací o zařízení uložených v open source databázi. Liferay Marketplace je centrum aplikací, kde jsou aplikace buď zpoplatněné, poskytovány zdarma, nebo vázané s enterprise edicí, a zmíněná aplikace je poskytována pouze k edici enterprise.

2.5.3 Service Builder

Jak sám název napovídá, tento nástroj slouží k vytvoření servisní vrstvy. Objektově relační mapování je zajišťováno prostřednictvím frameworku Hibernate. Programátor pouze definuje entity do souboru `service.xml` (ukázka č. 5) a Service Builder se postará o vygenerování modelové vrstvy, persistentních tříd a základních operací nad tabulkami (create, update, delete, find). Finder metody specifikují v service.xml souboru parametry, na jejichž základě získáme z databáze objekty.

Při vytváření servisní vrstvy máme možnost si nechat vygenerovat vzdálenou a místní servisní vrstvu. Místní služby mohou být spuštěné klientským kódem běžícím ve stejné JVM. Vzdálené služby mají navíc přidanou kontrolu pro oprávnění přístupu a jsou přístupné na internetu či na lokalní síti.

Po spuštění Service Builderu se vygenerují v souvislosti s findery následující metody fetchBy, findBy, removeBy a countBy ve třídách Persistence, které jsou následně implementovány ve třídách PersistenceImpl. V ukázkovém příkladě se budou metody jmenovat fetchByName, findByName, removeByName, countByName.

```
<entity name="Customer" local-service="true" remote-service="true">
    <!-- PK fields -->
    <column name="id" type="long" primary="true" />
     <!-- Audit fields -->
     <column name="createDate" type="Date" />
     <!-- Other fields -->
     <column name="name" type="String" />
     <!-- Order -->
     <order by="asc">
       <order-column name="id" />
     </order>
     <!-- Finder methods -->
     <finder name="Name" return-type="Collection">
     <finder-column name="name" />
     </finder>
</entity>
```

Výpis 5: Ukázka definování entity zákazník

Mezi dalšími vygenerovanými třídami jsou velmi důležité LocalServiceImpl/RemoteServiceImpl a LocalServiceUtil/RemoteServiceUtil. Ve třídách ServiceImpl se přidávají metody, které plní námi požadované operace nad modelem. ServiceUtil třídy slouží k volání těchto metod z portletu.

2.6 Apache Maven

Maven je nástroj pro správu a automatizaci sestavování aplikací. Původně byl určen k usnadnění vývoje projektu Jakarta Turbine, který se skládal s menších projektů. Záměrem bylo, vytvořit definici projektu a jeho závislostí, ze kterých se skládá, a poskytnout jednoduchý způsob, jak sdílet .jar soubory napříč projekty [13].

Maven je také poskytován ve formě pluginu pro vývojové prostředí NetBeans a Eclipse. Z vytvářených projektů je možné snado sestavit výsledný .jar nebo .war soubor, který poté budeme moci spustit.

2.6.1 Project Object Model

Veškerá konfigurace Mavenu se provádí v souboru `pom.xml` (ukázka č.6), ten musí obsahovat:

- groupId - unikátní identifikátor skupiny projektu, do které projekt patří.
- artifactId - unikátní identifikátor projektu (obvykle název projektu).
- version - využívá se k identifikaci konkrétní verze projektu.

Knihovny, na kterých projekt závisí, se definují v tagu `<dependencies/>`. Ty jsou uloženy do lokálního repositáře .m2 v domovském adresáři. Element `<modelVersion/>` označuje verzi POM, kterou používáme. Ve verzi 4.0 byla přidána schopnost měnit své nastavení v závislosti na zvoleném profilu. Více profilů se hodí zejména u projektů, které se sestavují v prostředích s odlišnou konfigurací. Může se jednat například o odlišné operační systémy, nebo o sestavení určené pro testovaní a nasazení.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven
    .apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.codehaus.mojo</groupId>
  < artifactId >my-project</artifactId>
  <version>1.0</version>
  <dependencies></dependencies>
</project>
```

Výpis 6: Struktura souboru pom.xml

2.7 Freemarker

Jde o šablonovací systém, sloužící ke generování textového výstupu z html kódu na základě definované šablony. Výstup se zpravidla dále ukládá do formátů, které obsahují text (TXT, XML).

Vstupní data se přebírají z kolekce `Map<String, Object>`, kde se k datům přistupuje přes námi pojmenovaný klíč. Hodnoty mohou být primitivní datové typy, kolekce, nebo uživatelem definované typy.

Pro generování dynamického obsahu se používají následující typy direktiv:

- ${...} - ve výstupu bude nahrazena tato direktiva hodnotou uvnitř složených závorek
- FTL tagy - tyto tagy, které slouží jako instrukce Freemarkeru k provedení příkazů, nejsou ve výstupu ze šablony. Jedná se konkrétně o tagy začínající # a o uživatelsky definované tagy @.

2.8 DocBook

Vývoj software pro zákazníka je spojený s tvorbou dokumentace, kde se nachází informace o produktu a nápověda, jak s ním pracovat. Někdy je také nutné poskytnout dokumentaci v různých podobách. Může se jednat o tištěnou nebo elektronickou verzi a také o nápovědu přímo v aplikaci. DocBook tuto práci usnadňuje tím, že z vyvtvářené předlohy se dají generovat dokumenty různých formátů.

Jde o formát založený na SGML/XML určený k tvorbě formátovaných dokumentů, které mohou být dále převedeny do formátů: HTML, PDF, RTF. Může se jednat o dokumentaci, článek nebo i knihu. Používá se například ke psaní dokumentace Linuxu, PHP a tam, kde nejsou kladeny velké požadavky na vzhled. Předloha se může skládat z několika menších souborů, které je možné využít pro opakující se sekce v dalších formátech dokumentace. Při úpravách textu v jedné z částí, se změna promítne do všech dokumentací, které vytváříme z předlohy. Tato vlastnost nám ušetří spoustu času při aktualizaci dokumentů a hlavně udržuje jejich konzistenci.

2.9 Git

Při vývoji větších projektů často potřebujeme zaznamenávat informace o provedených změnách v souborem a mít také možnost tyto změny vrátit zpět. Řešením tohoto problému je použití systému pro správu verzí a jedním z nich je právě Git. Vývoj samotného Gitu žačal v roce 2005, když systém pro správu verzí BitKeeper byl zpoplatněn. Tento systém používala komunita, která vyvíjela Linuxové jádro, a ta se rozhodla vyvinout vlastní bezplatný nástroj založený právě na znalostech z používání BitKeeperu. [6]

Git je distribuovaný systém správy verzí, kde každý člen týmu má na svém disku kompletní kopii projektu i s jeho historií provedených změn. V případě pádu serveru, může kdokoli z týmu poskytnout svou kopii projektu k obnovení repozitáře. Jeho předností oproti jiným systemům je, že umožňuje vývojáři pracovat offline, snadno vytvářet a slučovat větvě.

Další jeho výhodou je, že při každém uložení projektu zaznamená stav všech souborů projektu ve formě snímku a ukládá si reference na pořízený snímek. Pokud stav jednoho ze souborů zůstane nezměněný, uloží se odkaz na předchozí snímek tohoto souboru. Navíc není omezem pouze pro práci s textovými soubory, ale je možné ho použít i pro sledování změn obrázků.

2.9.1 Protokoly

V týmu je často potřeba zajistit, aby každý z nich měl možnost si kdykoli stáhnout aktuální verzi projektu. To nás vede k tomu, abychom zřídili vzdálený repositář. Pro přenos dat mezi stanicemi se musí zvolit jaký protokol použijeme. Git nabízí tyto protokoly: Local, HTTP, SSH a Git. S výjimkou protokolu HTTP není potřeba mít na serveru nainstalovaný Git.

Lokální protokol

Nejjednodušší volbou je si založit vzdálený repozitář ve složce na místním disku a nastavit u ní veřejný přísup. Pokud jsou členové týmu v místní síti, získávají tak velmi rychlý přístup k repozitáři. To se stává zároveň nevýhodou v okamžiku, kdy pracujeme mimo místní síť a chceme uložit novou verzi do repozitáře.

HTTP a HTTPS

Protokol HTTP/HTTPS se spolu s protokolem Git používají především ke čtení, protože se u nich obtížně nastavují práva k zápisu do repozitáře. Předností tohoto protokolu je jednoduchost nastavení přístupu ke čtení z repozitáře, který stačí umístit do kořenového adrasáře a nastavit skript post-update k provedení operací po nahrání nové verze projektu. Další z jeho výhod jsou minimální nároky na server, na druhou stranu stahování dat z repositáře trvá déle kvůli většímu počtu síťových operací.

SSH

Jde o nejpoužívanější z uvedených protokolů, který umožňuje snadno číst a zapisovat do repozitáře po autentizaci uživatele. Podporuje také šifrování při přenosu dat jako HTTPS, navíc dovoluje data komprimovat před vlastním přenosem a tím urychlit stahování dat ze vzdáleného repozitáře. Přístup do repositáře je povolen na základě SSH klíčů. Na straně klienta jde o privátní klíč a na serveru o veřejný klíč. Ověření proběhne tak, že server pošle náhodně vygenerovaná data klientovi. Ten je privátním klíčem podepíše, odešle na server a ten pomocí veřejného klíče ověří pravost podpisu. Tajná informace (privátní klíč) zůstává u klienta a není tak možné získat neoprávněný přístup do repositáře. [7]

Git protokol

Podobné vlastnosti jako SSH nabízí také protokol Git s tím rozdílem, že přístup do repozitáře je bez autentizace a bež šifrovaného přenosu. Pokud je repozitář přístupný, každý má k němu přístup pro čtení nebo i zápis. Proto se tento protokol využívá v případech kdy vyžadujeme rychlý přístup k získání kopie repozitáře a šifrování by zbytečně přenos dat zpomalilo.

3 Projekt

Ministerstvo školství SR požadovalo vybudování řešení na podporu výuky za pomocí nejmodernějších technologií pro mateřské, základní a střední školy. V rámci navrhovaného řešení budou zřízeny digitální třídy napojené na informační systém. Pro základní a střední školy bude systém určený k rozvoji digitální gramotnosti, kdy se žáci budou moci zapojovat do tvorby obsahu výuky formou různých vědomostních soutěží pomocí tabletů.

Našim úkolem bylo implementovat softwarovou část projektu. Vzhledem k jeho velikosti a množství kladených požadavků, byl pro přehlednost vyvíjen v menších portletech:

- skvp - hlavní portlet, který závisí na níže uvedených portletech.
- dvo - správa digitálního vzdělávacího obsahu
- help - nápověda v rámci systému pro uživatele
- study-materials - editace a tvorba studijních materiálů
- enums - výčtové typy s nadefinovanými hodnotami
- notif - slouží k přeposílání zpráv o událostech uživatelům v rámci Liferay
- workflow - určen pro tvorbu procesů, jejich schvalování

Z uvedených portletů jsem se podílel na `skvp`, který slouží k tvorbě ŠkVP a zobrazení státného vzdělávacího programu (ŠVP). V ŠVP se definují všeobecné cíle a kompetence, ke kterým má vzdělávání směřovat, a na jeho základě si vytváří škola individuální ŠkVP.

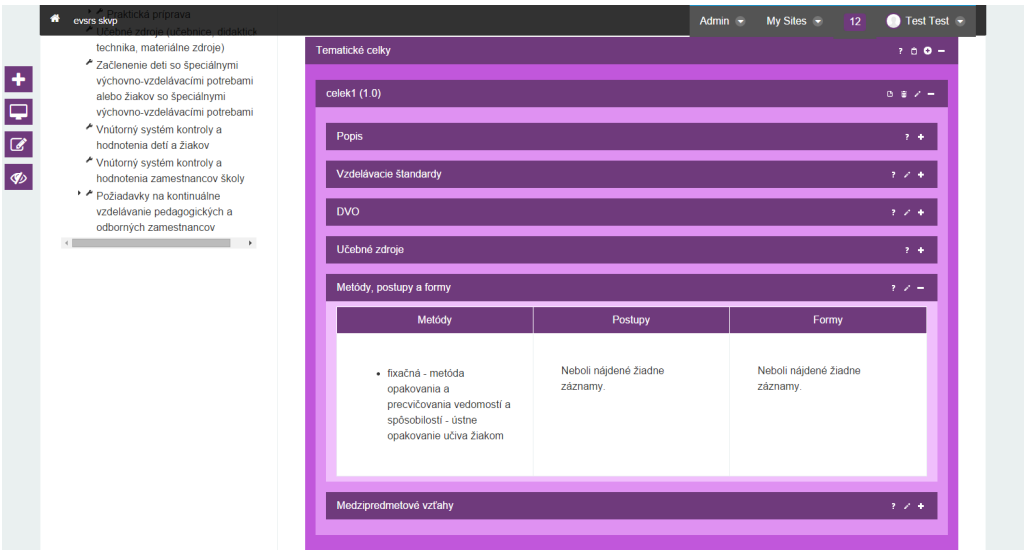
Každý portlet byl vytvářen s pomocí pluginu Maven, což umožnilo snadněji udržovat závislosti pro konkrétní projekt, které tvoří jeden funkční celek. Dalším důvodem bylo jednoduché sestavení souboru war, který se následně může nahrát na server jako webová aplikace.

Celý projekt byl spravován ve vzdáleném git repozitáři, ke kterému se přistupovalo přes SSH protokol. Abych získal oprávněný přístup do repositáře, musel jsem si vygenerovat RSA klíč - například pomocí programu PuTTYgen. Poté stačilo přidat vygenerovaný veřejný klíč na server, kde byl umístěn repositář, a na místní stanici ponechat privátní klíč. Kadý den, kdy jsem na projektu pracoval, bylo potřeba stáhnout jeho aktuální verzi, která oproti minulé zpravidla doznala změn. Vzhledem k rychle probíhajícímu vývoji, jsem po dokončené funkcionalitě v jedné z částí projektu musel před samotným odesláním své verze projektu do gitu zkontrolovat, jestli zde není nová revize projektu, abych mohl nahrát svou verzi do repositáře.

Projekt byl vytvářen v komunitní edici Liferay v JSF. Pro grafické rozhraní webových stránek se použily komponenty Primefaces, které nabízí nejvíce komponent z podporovaných grafických knihoven pro Liferay. Primefaces jsou postaveny na HTML5, jQuery a CSS3. Do stránek tak bude možné přidávat obsah, který nebude závislý na přítomnosti flash playeru.

3.1 Tvorba stránky

Na ukázku z projektu jsem vybral část, která se týkala zobrazení informací z předmětu pro daný stupeň. Konkrétně se jedná o informace o tématických celcích. Na obrázku č.3 je snímek obrazovky, který ukazuje výslednou stránku. Pro jednotlivé údaje tématického celku se použily rozbalovací panely s ovládacími prvky v hlavičce. Obsah každého panelu byl pro přehlednost odlišen jiným odstínem fialové podle úrovně zanoření, která mohla nabývat maximálně hodnoty 2.



Obrázek 3: Stránka s tématickými celky

3.1.1 Managed bean

Data pro portál se získávala z MySQL databáze. Takto získaná data poté naplnila entity definované v souboru `service.xml`. Pro ukázání funkcionality z ukázky bylo potřeba definovat entity: `VazbaSmMpf`, `MetodyPostupyFormy`, `StandardMnozina`. Entita `VazbaSmMpf` představuje vazební tabulku mezi zbylými zmíněnými entitami. V ní se definoval finder s názvem StandardMnozina na sloupec s id StandardMnoziny. Po spuštění Service Builderu se vygenerovala ve třídě VazbaSmMpfPersistenceImpl metoda `findByStandardMnozina(long id)`. Následně se v další z vygenerovaných tříd VazbaSmMpfLocalServiceImpl implementovala metoda `getMpfByIdAndTyp(Long tcId, mpfTyp)`, ve které se vytváří uživatelem vytvořené služby. V této metodě persistentní třídy - v tomto příkladě `findByStandardMnozina(long id)`.

Pomocí třídy `PredmetSkvpViewBean` se přistupovalo k datům pro předmět. Aby bylo zaručené, že ze stránky se bude volat tato třída, je dobrým zvykem ji pojmenovat předáním hodnoty do parametru `name` anotace `@ManagedBean`. Bez zadání vlastního pojmenování třídy se bere jako výchozí její název `predmetSkvpViewBean`.

Ze stránky se volá metoda `getMpf(Long tcId, String mpfTyp)`, která předává parametry tcId a mpfTyp do stejnojmenné metody v servisní třídě VazbaSmMpfLocalServiceUtil. Ta volá metodu `getMpfByIdAndTyp(long id, string typ)` implementovanou v VazbaSmMpfLocalServiceImpl. V ní se prochází seznamem třídy `MetodyPostupyFormy`, který vrací metoda `findByStandardMnozina(long id)` vygenerovaná ve třídě VazbaSmMpfPersistence, a do nového seznamu se přidává třída, jejíž hodnota atributu typ se rovná hodnotě řetězce `typ`.

```
@ManagedBean(name = "predmetSkvpView")
@ViewScoped
public class PredmetSkvpViewBean extends AbstractStav {
...
    public List<MetodyPostupyFormy> getMpf(Long tcId, String mpfTyp) {
        List<MetodyPostupyFormy> mpf = Collections.emptyList();

        try {
            mpf = VazbaSmMpfLocalServiceUtil.getMpf(tcId, mpfTyp);
        } catch (SystemException e) {
            LOG.error(e.getMessage(), e);
        }

        return mpf;
    }
...
}
```

Výpis 7: Ukázka části kódu třídy predmetSkvpViewBean

3.1.2 Lokalizace

Liferay poskytuje vlastní mechanismus internacionalizace pomocí vestavěného klíčového slova i18n, tento způsob byl také použit v projektu. Do konfiguračního souboru liferay-hook.xml (v JSF faces-config.xml) se přidá název souboru s konfiguracemi pro portál do `<portal-properties/>` a název lokalizovaného souboru do `<language-properties/>`. Lokalizovaný text se získává přes proměnnou i18n, kterou není třeba uvádět jako u JSF. To jsou jediné odlišnosti oproti konfiguraci internacionalizace v JSF.

3.1.3 Kód stránky

Pro zobrazení metod, postupů a forem tématického celku jsem použil panelGrid, který uspořádává prvky do mřížky o 3 sloupcích v tomto případě. Sloupce jsou vloženy do hlavičky `<f:facet name="header">`. Text se do sloupců hlavičky získává z lokalizovaného souboru. V každém ze sloupců je seznam `<p:dataList>`, který zobrazí zprávu předanou parametru emptyMessage, když dostaneme prázdnou kolekci z `predmetSkvpView` (výpis č.7). Pokud se z metody `getMpf(tc.id, 'METODA')` vrátí neprázdný seznam, tak `<h:outputText>` zobrazí název metody pro daný tématický celek, kterou přebírá parametrem `value`. Parametr escape je příznak, který při hodnotě false, zobrazí hodnotu komponenty jako text. Při hodnotě true se uplatňují pravidla pro speciální

znaky. Pro zobrazení popisu metody tématického celku je použit outputPanel, který parametrem rendered přebírá pravdivostní hodnotu z podmínky, zda popis metody obsahuje text. To určuje, jestli se zobrazí vnořený `<h:outputText>`. Nemělo by význam zbytečně zobrazovat prázdný text.

```
<p:panelGrid
    id="evsrs-skvp-mpf-table"
    style="width: 100%;">
    <f:facet name="header">
        <p:row>
            <p:column>#{i18n['evsrs-skvp-mpf-metody']}</p:column>
            <p:column>#{i18n['evsrs-skvp-mpf-postupy']}</p:column>
            <p:column>#{i18n['evsrs-skvp-mpf-formy']}</p:column>
        </p:row>
    </f:facet>
    <p:row>
        <p:column style="vertical-align: top; width: 33%;">
            <p:dataList
                    id="evsrs-skvp-mpf-metody-list"
                    value="#{predmetSkvpView.getMpf(tc.id, 'METODA')}"
                    var="mpf"
                    emptyMessage="#{i18n['evsrs-ziadne-zaznamy']}">
                    <h:outputText value="#{mpf.nazov}" escape="false" />
                    <p:outputPanel rendered="#{not empty mpf.popis}">
                        <h:outputText value="#{mpf.popis}" escape="false" />
                    </p:outputPanel>
                </p:dataList>
            </p:column>
            ...
    </p:row>
</p:panelGrid>
...
```

Výpis 8: Ukázka části kódu stránky s prvky Primefaces pro výstup tématického celku

3.2 Tiskové výstupy

K vygenerování textové podoby ŠkVP byly použity: Freemarker, DocBook, SAXParser, FOP. Výsledný pdf dokument poté slouží k archivaci po dobu 5 let.

Celý proces tvorby pdf dokumentu začal tvorbou šablony ve Freemarkeru, která vygenerovala textový výstup. Ten sloužil jako vstup pro DocBook, ze kterého vznikl přes SAXParser XML soubor (výpis č.11). Pro další použití se musel tento soubor následně převést pomocí FOPu do PDF. FOP je formátovací nástroj, který ze stromové reprezentace dat (XML) vytváří soubory určené k tisku.

3.2.1 Příprava dat

Data pro šablonu Freemarkeru ke generování textového výstupu metod, postupů a forem tématického celku se připravují do mapy, kde se k hodnotám přistupuje přes námi definovaný klíč. Do hlavní mapy `map`, ze které přebírá Freemarker data, se přidává mapa,

Tematické celky

celek1 (1)		
popis celku		
Výkonové štandardy	**Obsahové štandardy**	
• vyk. standard1	• obsah. standard	
Metódy	**Postupy**	**Formy**
• fixačná - metóda opakovania a precvičovania vedomostí a spôsobilostí - ústne opakovanie učiva žiakom		
Literatúra	**Pomôcky**	
kniha		

Obrázek 4: Vygenerovaný výstup pomocí Docbooku

která obsahuje seznam metod, postupů a forem. Do vnořené mapy `mpf` se vkládají výsledky z dotazů funkce `getMpf()`, která vrací konkrétní seznam pro tématický celek podle jeho id a hodnoty stringu určující, zda se jedná o metody, postupy, nebo formy.

```
Map<String, Object> map = new HashMap<>();
Map<Long, Map<String, List<MetodyPostupyFormy>>> metodyPostupyFormyMap = new
    LinkedHashMap<>();
...
Map<String, List<MetodyPostupyFormy>> mpf = new LinkedHashMap<>();
mpf.put(GeneratorConstants.FORMA, VazbaSmMpfLocalServiceUtil.getMpf(standardMnozina.getId
    (), GeneratorConstants.FORMA));
mpf.put(GeneratorConstants.POSTUP, VazbaSmMpfLocalServiceUtil.getMpf(standardMnozina.
    getId(), GeneratorConstants.POSTUP));
mpf.put(GeneratorConstants.METODA, VazbaSmMpfLocalServiceUtil.getMpf(standardMnozina.
    getId(), GeneratorConstants.METODA));
metodyPostupyFormyMap.put(standardMnozina.getId(), mpf);
...
map.put(GeneratorConstants.METODY_POSTUPY_FORMY, metodyPostupyFormyMap);
```

Výpis 9: Příprava dat pro Freemarker

3.2.2 Šablona

Následující šablona přistupuje k připraveným datům v předchozí části. Z ukázky šablony se týkají Freemarkeru pouze jeho direktivy, které naplňují tabulku daty. Tagy, které tvoří vzhled tabulky patří k Docbooku, jehož část rozeberu o kapitolu níže.

Nejprve se do proměnné `metodyPostupyFormyMap` přiřadí mapa podle id tématického celku. Hlavička a tělo tabulky v DocBooku musí mít shodný počet sloupců, proto se před tvorbou tabulky kontroluje, zda alespoň jeden seznam není prázdný, protože nepotřebujeme generovat do dokumentu prázdnou tabulku (obrázek č.4). Do hlaviček

sloupců se vloží hodnoty ze souboru s lokalizovanými texty. Pro každý sloupec se zjišťuje, jestli není seznam prázdný. V případě, že seznam obsahuje nějaká data, tak se prochází v cyklu `<#foreach>` seznam a vkládá se text do sloupce. V uvedené ukázce je znázorněno vkládání textu pro jeden sloupec - formy. Ve zbylých sloupcích se provádí stejný postup naplňování textem.

```
...
<#assign metodyPostupyFormyMap = metodyPostupyFormy.get(tematickyCelek.id) >
                    <#if  metodyPostupyFormyMap.get('METODA')?has_content ||
                          metodyPostupyFormyMap.get('POSTUP')?has_content ||
                          metodyPostupyFormyMap.get('FORMA')?has_content >
<row>
   <entry namest="evsrs-vykonove-standardy" nameend="evsrs-obsahove-standardy">
      <informaltable  frame='none' pgwide="1">
      <tgroup cols='3'  align=' left '  colsep='1'  rowsep='1'>
         <colspec colname='metody'/>
         <colspec colname='postupy'/>
         <colspec colname='formy'/>
         <thead>
            <row>
               <entry  align="center">${i18n.get('evsrs-skvp-mpf-metody')}</entry>
               <entry  align="center">${i18n.get('evsrs-skvp-mpf-postupy')}</entry>
               <entry  align="center">${i18n.get('evsrs-skvp-mpf-formy')}</entry>
            </row>
         </thead>
         <tbody>
            <row>
               <entry>
               <#if  metodyPostupyFormyMap.get('METODA')?has_content >
                  <itemizedlist  mark='bullet'>
                     <#assign metody = metodyPostupyFormyMap.get('METODA') >
                     <#foreach metoda in metody>
                        <listitem>
                           <para>${metoda.nazov}</para>
                        </listitem>
                           </#foreach>
                        </itemizedlist>
                     </#if>
                     </entry>
                  ...
            </row>
         </tbody>
        </tgroup>
      </informaltable>
   </entry>
</row>
```

Výpis 10: Ukázka části šablony Freemarkeru

3.2.3 Výstup

Tabulka je tvořena pomocí DocBooku, která se skládá z další vnořené tabulky pro metody, postupy a formy. Vnější tabulka je definována pomocí dvou sloupců `evsrs-vykonove-standardy` a `evsrs-obsahove-standardy`, kde se pro název a popis celku oba sloučily. Ve vnější tabulce se musely pro účely sloučení sloupce pojmenovat pomocí hodnoty předané do atributu `<colspec colname=''/>`. Pro metody, postupy a formy bylo potřeba tabulku rozšířit o 1 sloupec. To se provedlo roztažením vnitřní tabulky přes oba pojmenované sloupce použitím `<entry namest="" a nameend="">`, které určuje hranice vnitřní tabulky.

```
<entry namest="evsrs−vykonove−standardy" nameend="evsrs−obsahove−standardy">
  <informaltable frame='none' pgwide="1">
    <tgroup cols='3' align=' left ' colsep='1' rowsep='1'>
      <colspec colname='metody'/>
      <colspec colname='postupy'/>
      <colspec colname='formy'/>
      <thead>
        <row>
          <entry align="center">Metódy</entry>
          <entry align="center">Postupy</entry>
          <entry align="center">Formy</entry>
        </row>
      </thead>
      <tbody>
        <row>
          <entry>
            <itemizedlist mark='bullet'>
              <listitem >
                <para>fixaćná − metóda opakovania a precvićovania vedomostí a spó
                    sobilostí − ústne opakovanie učiva žiakom</para>
              </listitem>
            </itemizedlist>
          </entry>
          <entry>
          </entry>
          <entry>
          </entry>
        </row>
      </tbody>
    </tgroup>
  </informaltable>
</entry>
```

Výpis 11: Ukázka XML souboru z vygenerovaného textu

4 Závěr

Na praxi jsem přicházel bez jakýchkoliv znalostí programování webových aplikací. Když jsem se po krátké době na zaučení dověděl, čeho se bude moje praxe týkat, nebyl jsem si jistý, zda tento úkol zvládnu. Práce na projektu kladla jak na mě, tak i na konzultanta mé praxe velkou zodpovědnost, který musel testovat kromě své i mou implementovanou část projektu. Postupem času jsem se naučil pracovat s jednotlivými technologiemi, jejichž znalost mi bude přínosem do budoucna - hlavně v používání verzovacích systémů. Za významnou zkušenost také považuji, že jsem se mohl podílet na vývoji v týmu reálného projektu, kde bylo důležité mezi sebou komunikovat, koordinovat svou práci a reagovat na požadavky zákazníka. V době psaní této práce byl projekt odevzdán na testování. Z dosavadních informací se na naši straně neobjevila žádná výrazná implementační chyba.

Vzhledem k vývoji informačních technologií je zřejmé, že bude aktuální informační systém za několik let zastaralý. V průběhu záruční doby 5 let, která je poskytována na dodávaný software, bude potřeba sledovat nové verze technologií použitých v tomto projektu a řešit aktualizace softwaru se zákazníkem.

5 Reference

[1] dokumentace Hibernate.
URL: <*http://hibernate.org*> [cit. 2015-4-28].

[2] Hibernate tutorial.
URL: <*http://www.tutorialspoint.com/hibernate/index.htm*> [cit. 2015-4-28].

[3] Specifikace Java Persistence API.
URL: <*http://docs.oracle.com/javaee/7/api/javax/persistence/package-summary.html*>
[cit. 2015-4-28].

[4] dokumentace Liferay Portal.
URL: <*http://www.liferay.com/products/liferay-portal/overview*> [cit. 2015-4-28].

[5] dokumentace Primefaces.
URL: <*http://www.primefaces.org*> [cit. 2015-4-28].

[6] dokumentace Git.
URL: <*http://git-scm.com/book/en/v2*> [cit. 2015-4-28].

[7] Krčmář, Petr, Jak se přihlašovat na SSH bez zadávání hesla.
URL: <*http://www.root.cz/clanky/jak-se-prihlasovat-na-ssh-bez-zadavani-hesla*>
[cit. 2015-4-28].

[8] dokumentace DocBook.
URL: <*http://docbook.cz*> [cit. 2015-4-28].

[9] Horáček, Petr, Java na Webu II. - Základní stavba.
URL: <*http://www.linuxsoft.cz/article.php?id_article=1970*> [cit. 2015-4-28].

[10] webové aplikace.
URL: <*https://kore.fi.muni.cz/wiki/index.php/Kategorie:Webové_aplikace*> [cit. 2015-4-28].

[11] Komponentní technologie.
URL: <*http://www.cs.vsb.cz/benes/vyuka/pte/texty/komponenty/index.html*>
[cit. 2015-4-28].

[12] Basic JSP Tutorial.
URL: <*http://www.tutorialspoint.com/jsp/*> [cit. 2015-4-28].

[13] dokumentace Apache Maven.
URL: <*https://maven.apache.org*> [cit. 2015-4-28].

[14] dokumentace Java EE 6.
URL: <*http://docs.oracle.com/javaee/6/tutorial/doc*> [cit. 2015-4-28].

[15] uzavřená smlouva s MŠVVaŠ SR.
URL: <*http://crdvo.uvo.gov.sk*> [cit. 2015-4-28].