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

Diplomová práce

Systém na správu a řízení vnitrofiremního vzdělávání

Plzeň, 2014 Martin Sýkora

Originál zadání

Zanalyzujte potřeby a vytvořte informační systém pro správu a řízení vnitrofiremního vzdělávání. Systém umožní organizaci interních i externích školení, vypisování realizací, žádosti o realizace atd. Součástí práce bude také napojení na externí kalendář formou volitelného pluginu. Součástí bude také zapojení datových zdrojů pro školení, například prezentace, příklady, ale i streamování videa a audionahrávek. Velká data budou uložena vhodnou technologií. Autor navrhne použitelné a moderní prostředí na platformě Java EE 6 s vyžitím moderních prvků GUI a UI frameworků. Při zpracovávání práce bude kladen velký důraz na snadnost použití, integraci s externími systémy (např. kalendář) a informační hodnotu řešení.

1. Analyzujte požadavky kladené na informační systém správy a řízení vnitrofiremního vzdělávání.
2. Navrhněte GUI s možnostmi streamování a moderními prvky.
3. Navrhněte business logiku aplikace s využitím Java EE návrhových vzorů.
4. Ověřte naplnění požadavků vybraných na základě dohody se zákazníkem praktickou realizací na platformě Java EE 6.
5. Navrhněte rozhraní pluginu pro kalendář.
6. Ověřte funkčnost pluginu kalendáře praktickou realizací pro MS Exchange (Outlook kalendář) a Google (Gmail kalendář).
7. Zhodnoťte dosažené výsledky.

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

V Plzni dne 23. června 2014

.....

Martin Sýkora

Poděkování

Na tomto místě bych rád poděkoval vedoucímu této diplomové práce, panu Ing. Jiřímu Jansovi, za jeho pomoc, odborné vedení, věcné připomínky a cenné rady, bez nichž by tato práce nemohla vzniknout. Dále bych rád poděkoval panu Ing. Petru Přibylovi, jenž se také účastnil pravidelných schůzek a přispíval opět svými postřehy a cennými radami. V neposlední řadě pak děkuji panu Ing. Pavlu Královi, Ph.D., který se laskavě zhostil role garanta této práce.

Abstract

The system for control and management of internal education

The aim of this thesis is to get familiar with modern technologies used for creating web applications based on Java EE 6 and to subsequently use newly acquired knowledge to analyze needs and to create information system for control and management of internal education, that allows connecting to the third party calendar services and streaming of multimedia content for training courses, which will result in replacement of an outdated system that is used now by the new one.

In the first part of this thesis, technologies, with which it was necessary to get thoroughly familiar and which were afterwards used to create application, their theoretical background and capabilities, are presented. In the next part I deal with implementation details of most important parts of application and summary of results.

The result of this thesis is the complete application, which meets the requirements of modernity of the used technologies, design patterns and graphical design and UX.

Abstrakt

Systém na správu a řízení vnitrofiremního vzdělávání

Cílem této práce je seznámit se s moderními technologiemi tvorby webových aplikací v Java EE 6 a následně využít nově získané poznatky k zanalyzování potřeb a vytvoření informačního systému pro správu a řízení vnitrofiremního vzdělávání, který umožní napojení na kalendáře třetích stran a streamování multimediálního obsahu školení a v budoucnu tak nahradí zastaralý systém, který je doposud využíván.

V první části práce jsou prezentovány technologie, s kterými bylo nutné se důkladně seznámit a které byly následně využity při tvorbě této práce, jejich teoretické pozadí a možnosti. V další části se již zabývám implementačními detaily nejpodstatnějších částí aplikace a shrnutím výsledků.

Výsledkem této práce je aplikace, která splňuje požadavky na modernost jak použitých technologií a návrhových vzorů, tak i design a uživatelskou přívětivost.

Obsah

1 Úvod

Firma CCA Group a.s. v Plzni již dlouhá léta využívá svůj systém pro správu a řízení vnitrofiremního vzdělávání nazvaný Ramses Akademie. Tento rozsáhlý systém s obrovským množstvím funkcí se ale za dobu své existence stal zastaralým a z hlediska uživatelské přívětivosti nevyhovujícím. Z těchto důvodů vznikla potřeba systém inovovat za pomoci moderních technologií a postupů, což vyústilo až v zadání této práce.

Cílem této práce je tedy vytvořit základ nového systému, který bude splňovat požadavky zadavatele na moderní aplikaci využívající UI framework s možností streamování multimediálních souborů a zvýšení přehlednosti a UX (uživatelského zážitku z aplikace).

V teoretické části této práce se čtenář může seznámit s návrhem aplikace a s technologiemi, které jsou využity v samotné realizaci aplikace. Probírány jsou zde jak jednotlivé prvky, z kterých jsou sestaveny HTML stránky, tak technologie, které stojí za plněním a úpravami dat pro tyto stránky, technologie pro kompilování a sestavení celé aplikace, moduly kalendářů, atd. Tato kapitola obsahuje důležité teoretické poznatky, bez nichž by nebylo možné aplikaci vyvíjet.

V praktické části se poté zabývám samotným vývojem aplikace a zejména implementačními detaily nejdůležitějších částí aplikace, mezi které se řadí zejména propojení na Google a Outlook kalendář a streamování multimédií. V této části se také zabývám některými nastaveními serveru Glassfish, zejména zabezpečením. V neposlední řadě také popisuji vývoj implementaci pomocí návrhového vzoru Plugin.

2 Learning Managament System

Learning Management System je aplikace, často webová, jejímž cílem je udržovat databázi studentů, kteří mají přístup k e-learningovým kurzům, poskytovat jim tyto kurzy a ukládat jejich výsledky. Dále obvykle poskytuje možnost plánování kurzů, možnost aby se uživatelé na tyto kurzy mohli hlásit a pak je absolvovat. Naproti tomu je možné se setkat se systémy LCMS (Learning Content Management System), které jsou primárně zaměřeny na vytváření obsahu a kurzů pro LMS.

3 Technologie

Technologie, které byly použity v této práci a které popisuji níže, byly zvoleny firmou CCA, proto nebylo potřeba zkoumat a porovnávat další.

3.1 HTML5

HTML5 je značkovací jazyk určený pro vytváření webových stránek a jako takový obsahuje sadu značek, které popisují a strukturují obsah vytvářeného dokumentu.

V roce 1998 zastavila organizace World Wide Web Consortium (W3C), která se stará o webové standardy, vývoj HTML, které se v té době nacházelo ve verzi 4.01, na úkor jeho nástupce XHTML, v nějž byly vkládány velké naděje jako do budoucnosti webu. XHTML se ale při vývoji své druhé verze ukázalo být slepou větví.

HTML5 je tak ve své páté revizi znovuoživením standardu HTML, o které se postarala Web Hypertext Application Technology Working Group (WHATWG), „*která se zaměřila na to, co chybí, ve smyslu věcí, které chtěli vývojáři udělat*“. [1 str. 5] Díky této skupině se tak ve specifikaci objevily, krom jiného, dva tagy, které jsou velmi důležité pro tuto diplomovou práci a to *<video>* a *<audio>* [1].

Množství formátů, které lze přehrávat, je ale zatím značně omezené. Video je možné přehrávat pouze ve formátech H.264, WebM a Ogg Theora, audio ve formátech MP3, WAV a Ogg Vorbis. Navíc ne všechny prohlížeče podporují všechny formáty (*viz* Příloha B – Podpora multimediálních formátů prohlížeči). Je ale možné vložit do tagu více zdrojů, každý v jiném formátu a prohlížeč zobrazí ten formát, který je schopen přehrát.

Audio a video bylo možné vkládat a přehrávat i v dřívějších revizích (za pomoci tagu <*object*>), k přehrávání byl ale za potřebí plugin.

Kód 3-1 HTML kód vkládající do stránky přehrávač audio souboru

```
<audio controls>
    <source src="{cesta k souboru}.mp3" type="audio/mpeg" />
</audio>
```

Obrázek 3-1 Ovládací panel přehrávání audia v Google Chrome

3.2 CSS3

CSS [1], neboli kaskádové styly, se používají pro formátování HTML stránek. Existují ještě další dva způsoby, ovšem ty jsou zastaralé a nejsou vhodné pro dynamicky generované stránky. Dříve se používal buď přímý zápis, kdy byla do nějakého HTML tagu přidána vlastnost *style*, pomocí níž se obsah daného tagu formátoval. Tento způsob vedl k přebujelému a nepřehlednému HTML kódu. Druhou možností bylo vložit do hlavičky stránky formátovací pravidla uzavřená v tagu <style></style>. Tento způsob už je výrazně lepší než ten předchozí, leč musí být vložen do každé stránky, a tím opět zbytečně narůstá množství přenášených dat. Od obou těchto metod je v současnosti upuštěno nebo minimálně upouštěno. Moderním způsobem je pro formátování stránek používat externí CSS soubory, které jsou snadno udržovatelné.

Externí CSS soubor, stejně jako styl uzavřený v tagu <style> má danou syntaxi, kdy se určuje prvek, ke kterému se styl vztahuje, a ve složených závorkách jsou poté vlastnosti, které mají tomuto prvku být přiřazeny (*viz* kód 3-2).

[1] Cascading Style Sheets

Kód 3-2 Stylování elementu h1

```
h1 {
    font-size: 16px;
    color: red;
}
```

Kaskádové styly a jejich třetí verze jsou plně podporovány HTML5 a přidávají některé dlouho očekávané funkce a nové možnosti stylování jako jsou barevné přechody, animace, stínování textu, apod.

Informace pro tuto kapitolu byly čerpány z [2].

3.3 Maven

Maven je nástroj založený na bázi pluginů[2], který má usnadnit a automatizovat buildování především Javovských aplikací. Lze jej ale využít i u jiných programovacích jazyků. Snahou tohoto projektu je usnadnit proces sestavení na maximální možnou míru a poskytnout jednotný systém pro sestavování projektů za pomoci POM (Project Object Model) souboru, který popisuje nejen jak se má projekt sestavit, jaké mají při tomto sestavení být použity pluginy, ale také daný projekt popisuje co do názvu, verze, apod. a udává jeho závislosti na externích knihovnách (tzv. dependencies). Díky POM může dále poskytovat další užitečné informace, jakou jsou seznamy změn, výsledky unit testů, apod.

Soubor *pom.xml* se nachází v kořenovém adresáři projektu a v případě zakládání Mavenového projektu v některém z IDE (např. NetBeans) se vytvoří celá adresářová struktura.

Maven dynamicky stahuje potřebné knihovny uvedené v dependecy ze svého centrálního repozitáře a ukládá je do místní kopie. V této místní kopii se pak nacházejí i artefakty projektů, které jsme si napsali a Mavenem sestavili my sami. Díky této funkci už nemusí programátor hledat knihovny na webu a sám je k projektu přidávat.

Informace pro tuto kapitolu byly čerpány z [3].

[2] Zásuvných modulů

3.4 Glassfish 4

Glassfish je open-source[3] aplikační server (resp. webový kontejner) vyvinutý společností Sun Microsystems v roce 2005 (v současné době patří pod Oracle Corporation). Jedná se o projekt, který vytváří referenční implementaci Java EE a s tím spojených technologií (JSF *viz* 3.5, JPA *viz* 3.11, EJB *viz* 3.12, EL *viz* 3.6, CDI *viz* 3.9, atd.).

3.5 Java Server Faces 2.2

Java Server Faces (dále JSF) je webový framework, který si klade za cíl zjednodušit a zrychlit vývoj a údržbu webových enterprise aplikací, především ale jejich User Interface (UI)[4], za použití standardizovaných komponent, které jsou součástí Java Enterprise Edition. V současnosti se nachází ve své druhé verzi (dáno specifikací JSR 314) a nabízí nepřeberné množství nástrojů k vytvoření příjemného a moderního uživatelského prostředí a usnadnění jeho vývoje. Jednou z hlavních devíz tohoto frameworku je možnost používat šablony, díky nimž je jednak urychlen vývoj a jednak je možno držet jednotný styl a uspořádání uživatelského rozhraní v celé aplikaci. Dále nabízí nástroje na validaci vstupů, navigaci apod. Krom jiného se JSF může pochlubit zabudovanou podporou Ajaxu, integrací s Expression Language a možnostmi internacionalizace.

„JSF allows developers to think in terms of components, events, managed beans, and their interactions, instead of requests, responses, and markup language.“ [4 str. 277]

Framework lze dále rozšiřovat pomocí komponentních knihoven třetích stran, které nabízejí řešení mnoha palčivých problémů. Jedná se například o knihovny IceFaces, RichFaces nebo Primefaces. O posledně zmiňované knihovně se více rozepisuji v kapitole 3.7.

---

[3] Software, k němuž jsou dostupné zdrojové kódy, které je možno dále upravovat podle vlastních potřeb a omezení licence

[4] Uživatelské rozhraní

3.5.1 Facelets

V JSF je jako zobrazovací technologie využíván jazyk Facelets, který je součástí specifikace JSF a zároveň je upřednostňovanou zobrazovací technologií oproti JSP (Java Server Pages), které byly využívány dříve, avšak ty nepodporují všechny nové funkce JSF. Facelets využívá pro tvorbu stránek jazyk XHTML, podporuje Expression Language (EL) – (*viz* kapitola 3.6 Expression Language) a zajišťuje šablonování (za pomoci knihovny tagů s prefixem „ui:“).

3.5.2 Beans

JSF je postaven na návrhovém vzoru MVC (Model – View – Controller), jehož snahou je oddělit prezentační a business logiku aplikace. Z tohoto důvodu se používají tzv. Beany, které mají toto oddělení zajistit.

Beana je prostá Javovská třída, která pouze dodržuje určité konvence a za pomoci getterů a setterů zpřístupňuje své parametry. K takovéto beaně pak mohou přistupovat jak JSF stránky, tak i další beany a třídy.

Pro pojmenování a deklaraci bean se používají anotace *@Named* a *@ManagedBean*. Hlavní rozdíl mezi *@Named* a *@ManagedBean* je v tom, že *@Named* beany jsou spravovány serverem (kontejnerem) přes CDI framework a díky tomu jsou dostupné pro všechny další, kontejnerem spravované artefakty, včetně JSF *@ManagedBean*, zatímco *@ManagedBean* je spravována JSF frameworkem a dalším ManagedBeanám je dostupná pouze přes *@ManagedProperty*. Obecně se doporučuje použití CDI (*@Named*) bean kvůli jejich rozsáhlejším možnostem, zároveň se očekává, že v Java EE 8 budou *@ManagedBeany* označeny za zastaralé.

Podle [5] se jedná o historický omyl, že pro beany využitelné v JSF existují tyto dva mechanismy a doporučuje použití CDI bean, pokud aplikace nemusí výslovně běžet na jiném, než Java EE 6 aplikačním serveru, např. Tomcatu.

CDI beany se používají stejně jako managed beany, jediný rozdíl je v jejich anotaci.

3.5.3 Bean Scopes

Při vytváření beany je potřeba specifikovat její Scope, který určuje její životní cyklus a přístupnost JSF stránkám a dalším beanám. Obecně existují tyto typy rámců.

- Request Scope *(@RequestScoped*)

  Udává, že instance beany bude existovat tak dlouho, jako HTTP požadavek.

- Session Scope (@*SessionScoped*)

  Stav objektů s tímto rámcem je mezi jednotlivými požadavky uživatele udržován, dokud není zrušena session.

- Application Scope (*@ApplicationScoped*)

  Toto je rámec s nejdelší životností. Beany s tímto scopem jsou inicializovány pouze jednou za celou dobu běhu aplikace a pro každý požadavek bude vrácena právě tato jedna instance.

- Conversation (*@ConversationScoped*)
- Dependent (*@Dependent*)

  Jedná se o základní scope. Nebude-li scope deklarován, automaticky je přiřazen tento a beana tak získává scope té beany, kam je injektována.

Dále je možno narazit na:

- View Scope
- Custom Scopes

3.5.4 Value Expression

Value Expression je jeden ze dvou výrazů, které nabízí Expression Language (*viz* kapitola 3.6). JSF stránky jej využívají nejen k zobrazení parametrů bean, ale také k nastavení těchto parametrů, čímž se zásadně liší od JSP stránek, které mohou hodnotu pouze zobrazit. Výraz vždy začíná křížkem (#) a je ohraničen složenými závorkami (*viz* kód 3-3). Uvnitř těchto závorek je již název beany, případně jméno proměnné (při procházení seznamu) a jméno parametru.

Po vyrenderování stránky se volá metoda getName objektu item, čímž získáme hodnotu tohoto parametru.

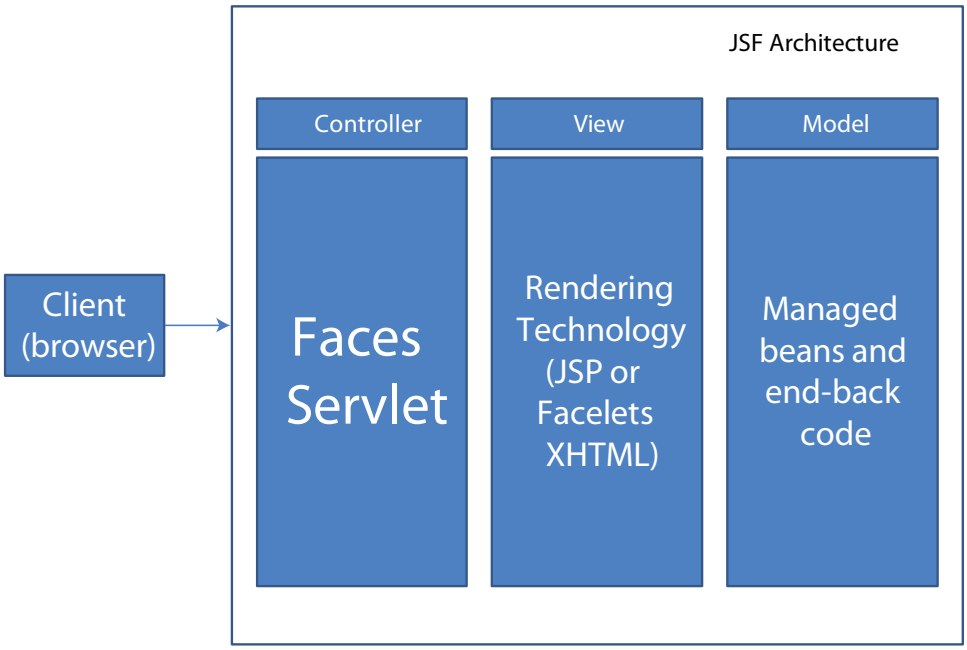
Kód 3-3 Value Expression

```
#{item.name}
```

3.5.5 Architektura

Jak již bylo zmíněno výše, architektura JSF je založena na návrhovém vzoru MVC Model 2 (*viz* obrázek 3-2). Zkratka MVC označuje Model – View – Controller. Tento vzor plně odděluje logiku aplikace od její zobrazovací vrstvy a nad vším dohlíží kontroler, který na základě požadavku uživatele určuje, co se má zobrazit a případně kde vzít data, kterými je toto zobrazení naplněno. Jako kontroler v tomto frameworku figuruje pouze Faces Servlet, který předává požadavky na jednotlivé stránky a ty, ve spolupráci s managed beanami, zobrazují a ukládají potřebná data.

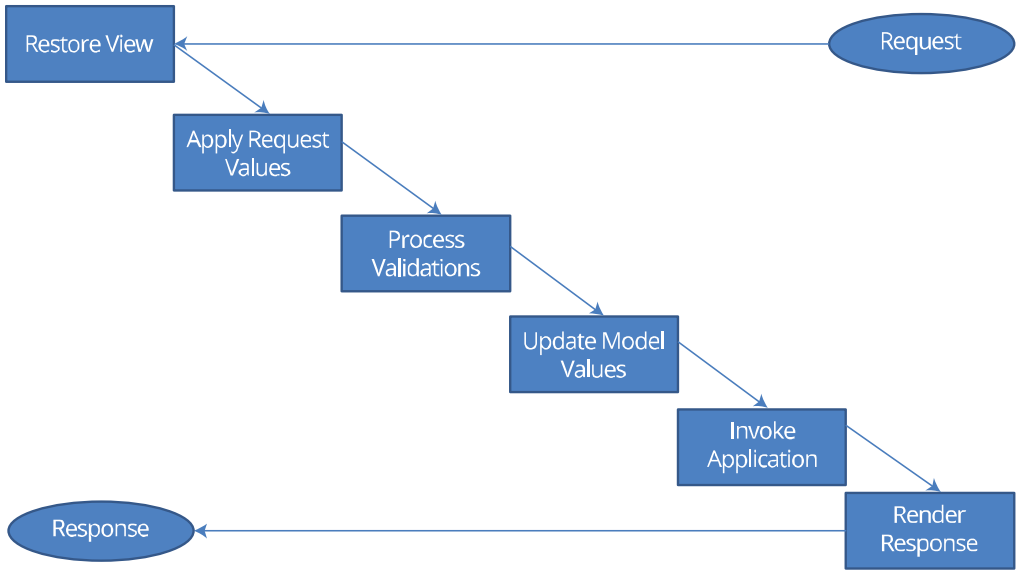
Obrázek 3-2 JSF Architektura [6]



3.5.6 Životní cyklus požadavku

Životní cyklus zpracování požadavku v JSF má šest fází (*viz* obrázek 3-3), které však nejsou sekvenční a mohou být přeskakovány například v případě, že se objeví nějaká validační chyba. V takovém případě může být, například, z fáze Process Validations přeskočeno rovnou na Render Response.

Obrázek 3-3 Fáze zpracování požadavku v JSF [6]



3.5.7 Konfigurace

Ke konfiguraci JSF se používají dva soubory, *web.xml* a *faces-config.xml*. V současné době je použití *faces-config.xml* nepovinné, protože většina metadat může být nastavena pomocí anotací přímo v backing-beanách. *Web.xml* je standardní deployment deskriptor [5], který obsahuje definici Faces Servletu (*viz* kód 3-4). Na začátku si čtenář může povšimnout nastavení kontextového parametru javax.faces.PROJECT_STAGE na Development (dalšími možnostmi jsou Production, SystemTest a UnitTest), což zajišťuje verbalistický výpis zpráv při chybě, čímž je usnadněno ladění.

Dále je vidět, že servlet je namapován na url-pattern „*.xhtml“, což znamená, že každý požadavek na xhtml soubor bude směrován na Faces Servlet. Tím se docílí toho, že může být připraven JSF kontext před zobrazením samotné JSF stránky.

Kód 3-4 Definice a mapování Faces Servletu v deployment deskriptoru

```
<context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
</context-param>
```

[5] Určuje kontejneru, jak se má aplikace deployovat (nachází se v {*kořenový_adresář/WEB-INF/*})

```
<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>
```

Informace pro tuto kapitolu byly čerpány ze zdrojů [4], [6] a [7].

3.6 Expression Language

Expression Language (dále EL) představuje důležitou součást Java EE, jelikož umožňuje prezentační vrstvě komunikovat s aplikační logikou.

Jak již bylo naznačeno v kapitole 3.5, EL pro JSP a JSF se liší. Nejenže se liší zápis, JSP používá značku dolaru ($) a JSF hashmark (#), ale liší se i možnostmi, kterými obě implementace disponují. V JSF je možné používat výrazy nejen pro čtení, ale i pro zápis hodnot do backing beany, v JSP lze pouze číst.

Níže je uveden příklad backing beany (kód 3-5), která obsahuje parametry k vypsání, případně zápisu a příklad výpisu hodnoty v JSF stránce (kód 3-6). Čtenář si může povšimnout, že ve výrazu je použito jméno beany, které je definováno u anotace (*@Named*), následuje parametr „student“, což je parametr této beany, který obsahuje data o konkrétním studentovi a parametr „name“, který vrací jméno studenta.

Když je stránka vyrenderována volá se metoda getStudent (objektu loggedUser), která vrátí objekt studenta a nad ním se zavolá metoda getName, která vrátí jméno daného studenta.

Zápis hodnot probíhá v případě, že stránka obsahuje pole pro vkládání, tak jako v kódu 3-7 a je odeslána na server. V takovém případě by se zavolala metoda setName, čímž by došlo k zápisu hodnoty do objektu.

Kód 3-5 Backing Bean

```
@Named(value = "loggedUser")
@SessionScoped
public class LoggedUser implements Serializable {
    private Student student;

    public LoggedUser() {
    }
    //… getters & setters…
}
```

Kód 3-6 Výstup v JSF stránce

```
<h:outputText id="userName" value="#{loggedUser.student.name}" />
```

Kód 3-7 Vstup v JSF stránce

```
<h:inputText id="userName" value="#{loggedUser.student.name}" />
```

Pomocí Expression Language lze také volat veřejné nestatické metody jednotlivých bean, ale v takovém případě se již nejedná o Value Expression (*viz* Value Expression), ale o Method Expression. Princip jejich fungování je však stejný. Opět se používají identifikátory bean a funkcí. Mohou se však volat pouze z komponent, které mají parametry Action nebo ActionListener (commandLink, commandButton) nebo valueChangeListener (inputText, SelectOneMenu).

3.7 PrimeFaces

Knihovna komponent rozšiřující JSF, byla jednou z prvních, které podporovaly JSF 2.0. Obsahuje sadu více než 100 komponent, které využívají jQuery, jQueryUI a podporuje HTML5. Primefaces využívají Ajax API z JSF. Tato knihovna nevyžaduje žádnou konfiguraci, v projektu se pouze přidá mavenová dependency a je možné ji začít používat.

Na stránkách PrimeFaces lze najít obsáhlou dokumentaci a showcase, který předvádí všechny komponenty této knihovny i s ukázkou kódu, který je potřeba k dosažení stejného výsledku.

3.8 PrettyFaces

PrettyFaces je open-source knihovna, která se stará o přepis URL na tzv. hezká nebo též sémantická URL. Jedná se o URL, která jsou věcná a popisná a i nezkušený uživatel podle nich dokáže odhadnout, co na stránce najde. Nesémantická URL naopak obsahují různé parametry, které si běžný uživatel není schopen zapamatovat a mnohdy ani určit, k čemu se používají. Sémantická URL jsou velmi vhodná pro SEO [6], leč výsledek této práce není primárně určen do veřejného internetu, a tak je tato výhoda zanedbatelná. Zanedbatelná není ale další výhoda, která z hezkých URL plyne, a tou je uživatelská přívětivost a tím pádem zvýšení uživatelského zážitku a komfortu.

PrettyFaces se přidá pomocí Mavenové dependency (*viz* kód 3-8) a konfiguruje se pomocí souboru *pretty-config.xml*, který je umístěn ve WEB-INF adresáři aplikace.

Kód 3-8 PrettyFaces Maven Dependency

```
<dependency>
    <groupId>com.ocpsoft</groupId>
    <artifactId>prettyfaces-jsf2</artifactId>
    <version>${latest.prettyfaces.version}</version>
</dependency>
```

3.9 ORM

Object-Relational Mapping je technologie, díky níž je možné, jak název napovídá, mapovat relační databáze na Javovské objekty. V podstatě se jedná o mapování entit na tabulky a parametrů na sloupce těchto tabulek.

Entita v podstatě není nic jiného, než Plain Old Java Object, který obsahuje anotaci *@Entity*, díky níž se od POJO odlišuje. Entita obsahuje atributy, s nimiž lze pracovat pomocí getterů a setterů a každý z těchto atributů představuje sloupec v tabulce, na níž je entita namapována.

3.10 JPA

Java Persistence API (JPA), je API, které má na starost persistenci dat. Jeho součástí je objektově relační mapování [8].

[6] Search Engine Optimization

3.10.1 JPQL

Java Persistence Query Language, je dotazovací jazyk, který je součástí Java Persistence API a ve své podstatě je rozšířením EJB QL (Enterprise Java Beans Query Language), k němuž přidává řadu funkcí. Svou syntaxí velmi připomíná SQL a používá se pro vyhledávání v entitách.

3.10.2 Named Queries

Jedná se o způsob, jak v entitě definovat určitý dotaz, který je možno pojmenovat a pak jej pomocí tohoto jména používat. Tyto dotazy jsou označeny anotací *@NamedQuery* a bývají cachovány.

3.11 EJB

Enterprise Java Beans jsou serverové komponenty, které se starají především o bussines logiku a datové transakce. Existují tři druhy Enterprise Java Bean. Jsou to:

- Session Beans
- Entity Beans
- MessageDriven Beans

3.12 Návrhové vzory

Nejdůležitější návrhové vzory, které byly použity při vývoji aplikace, jsou uvedeny v této kapitole.

3.12.1 Plugin

Jedná se o moderní návrhový vzor, který v součinnosti s CDI dovoluje snadnou rozšiřitelnost aplikace o nové funkce, bez nutnosti ji znovu kompilovat. Aplikace má závislost pouze na rozhraní, které implementují pluginy. Toto rozhraní musí být samostatně zabaleno v JAR archivu, stejně jako každá implementace tohoto rozhraní. V době zavádění či spouštění aplikace se pak kontejner postará o nalezení implementací tohoto rozhraní a umožní je využívat v aplikaci.

Pro vývoj pluginu je možné využít tři různé strategie. Jsou to:

- Strategie „Strategie“
- Strategie „Bridge“

- Strategie „Template“

Strategií, která je nejvhodnější pro tento projekt, se zabývám později v realizační části. Informace pro tuto kapitolu čerpány z [9].

3.12.2 Session Facade

Session Facade je navrhový vzor, který je tvořen Session beanami, které zapouzdřují a odstiňují implementační detaily od klienta, jemuž poskytují pouze potřebná rozhraní. Díky tomuto návrhovému vzoru je klient oddělen od bussines logiky zpracování persistentních operací a je mu místo toho nabídnuta řada služeb, které tyto operace „zakrývají“.

3.13 Kalendáře

Jedním z hlavních požadavků zadavatele byla integrace systému s kalendářem firmy Google a programu Outlook od Microsoftu. Tímto propojením bude dosaženo velmi dobrého uživatelského zážitku, protože uživatel si již nebude muset události, na něž je přihlášen, přepisovat do kalendáře ručně, vše proběhne automaticky. To samé platí o změnách v událostech provedených garantem vzdělávání. Tato kapitola pojednává o možnostech požadovaných kalendářů.

3.13.1 Google kalendář

Americká společnost Google Inc. nabízí uživatelům v rámci jednotného Google účtu mnoho rozličných služeb. Jednou z těchto služeb je i kalendář, který je, z mého pohledu, ve spojení s mobilními zařízeními, jednou z nejužitečnějších internetových služeb. Jedná se kalendář, který krom standardních funkcí, které se od kalendáře očekávají, nabízí také možnosti zvát na události další uživatele a vyžadovat od nich potvrzení či zamítnutí účasti, sdílet svůj kalendář s jinou osobou, případně hlídání průniku volných časů při plánování události. Ve spojení s mobilním zařízením se systémem Android a funkcí Google Now, pak může být uživatel včas upozorněn, že by se měl vydat na událost, kterou má v kalendáři. Toto upozornění zohledňuje uživatelovu polohu a předpokládaný čas cesty na událost, za předpokladu, že tato má uvedeno místo svého konání.

Google k většině (ne-li všem) svých služeb poskytuje API[7], je tedy možné napsat aplikaci, která bude se službami pomocí API komunikovat a využívat tak jejich možnosti. API kalendáře umožňuje vše, co je vyžadováno v zadání této práce a ještě mnohem více. Je tedy možno události vytvářet, editovat a mazat, obdržet seznam všech událostí, přesouvat události mezi různými kalendáři jednoho uživatele atd.

Aby ale bylo možné s těmito daty uživatele pracovat, musí tento nejprve udělit oprávnění naší aplikaci k přístupu k těmto datům. K autorizaci a autentizaci používá Google OAuth 2.0 protokol [10].

OAuth 2.0

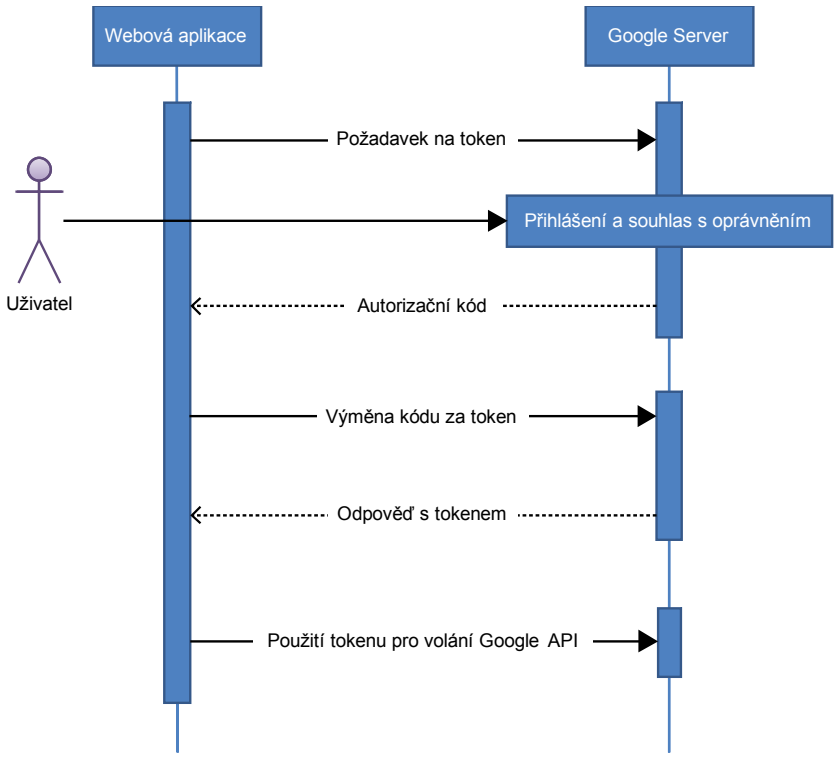
Chce-li se uživatel přihlásit v aplikaci do Google kalendáře tak, aby jej tato aplikace mohla využívat, musí provést několik jednoduchých kroků. Aplikace nejprve provede požadavek na autorizační kód. To přesměruje uživatele na stránku Google, kde se nejprve přihlásí do svého Google účtu. V případě, že je již přihlášen, se tento krok přeskočí. Poté je přesměrován na stránku, kde může, ale nemusí, odsouhlasit oprávnění, o která aplikace žádá. V případě, že tak učiní, je aplikaci vrácen autorizační kód na předem nastavené URL. Aplikace tento kód poté vymění s Google Serverem za přístupový token, který poté slouží jako přístupové heslo aplikace k datům uživatele. Celý tento proces je graficky znázorněn na obrázku 3-4.

Přístupový token má ale velmi krátkou dobu platnosti, standardně jednu hodinu, poté musí uživatel projít celý proces znovu. Pro aplikace, které vyžadují delší časový úsek je zaveden ještě tzv. Refresh Token, jehož platnost končí až ve chvíli, kdy jej uživatel sám odvolá.

Protokol OAuth tak umožňuje uživatelům sdílet svá data a informace z nějaké webové služby s aplikací, a přitom nevydávat svá přihlašovací jména ani hesla k této službě aplikaci, která tato data chce využívat.

[7] Application Programming Interface

Obrázek 3-4 Scénář autentizace a autorizace pomocí OAuth 2.0 [11]



3.13.2 Outlook kalendář

V podnikové sféře jedna z nejrozšířenějších forem kalendáře, díky možnosti jeho propojení s Microsoft Exchange Serverem, který umožňuje sdílení kalendářů, kontaktů apod. Samotný Outlook je však komplexní aplikace a kalendář je jen jednou z jejích součástí. Vzhledem k možnosti propojení s Exchange Serverem je také e-mailovým klientem, umožňuje spravovat kontakty apod.

Outlook umožňuje importování událostí do kalendáře pomocí souborů ve formátu iCalendar.

iCalendar

iCalendar (iCal) je datový formát v plain textu, určený pro výměnu a reprezentaci kalendářových dat. Je definován standardem RFC 5545 a aktualizován standardy RFC 5546 a RFC 6868. Jedná se o hojně využívaný formát, který lze použít např. i s Google kalendářem. Soubory tohoto formátu mají příponu *ics*.

Formát souboru je pevně dán standardem, který jej specifikuje. Na následujícím příkladu (viz kód 3-9) můžete vidět obvyklý formát takovéto zprávy. Tato zpráva

je tvořena jednotlivými elementy a jejich vlastnostmi. Hlavním elementem je objekt iCalendar, jehož první a poslední řádka musí obsahovat párový oddělovač VCALENDAR s patřičným označením začátku a konce tzn., první řádka musí obsahovat text *BEGIN:VCALENDAR* a poslední řádka musí být tvořena textem *END:VCALENDAR*. Zbytek souboru, který se nachází mezi těmito oddělovači, se nazývá iCalBody. Tělo objektu iCalendar je tvořeno povinnými parametry tohoto objektu (version, prodid), nepovinnými parametry (method, calscale) a jednou nebo více kalendářovými komponentami [12].

Kalendářových komponent je celá řada, ale pro účely této práce si vystačíme s komponentou události (VEVENT). Tato komponenta má, stejně jako objekt iCalendar, řadu parametrů (některé z nich povinné, některé ne), které tuto komponentu popisují. Komponenta opět musí být uvozena textem *BEGIN:VEVENT* a *END:VEVENT*.

Kód 3-9 Formát souboru iCalendar

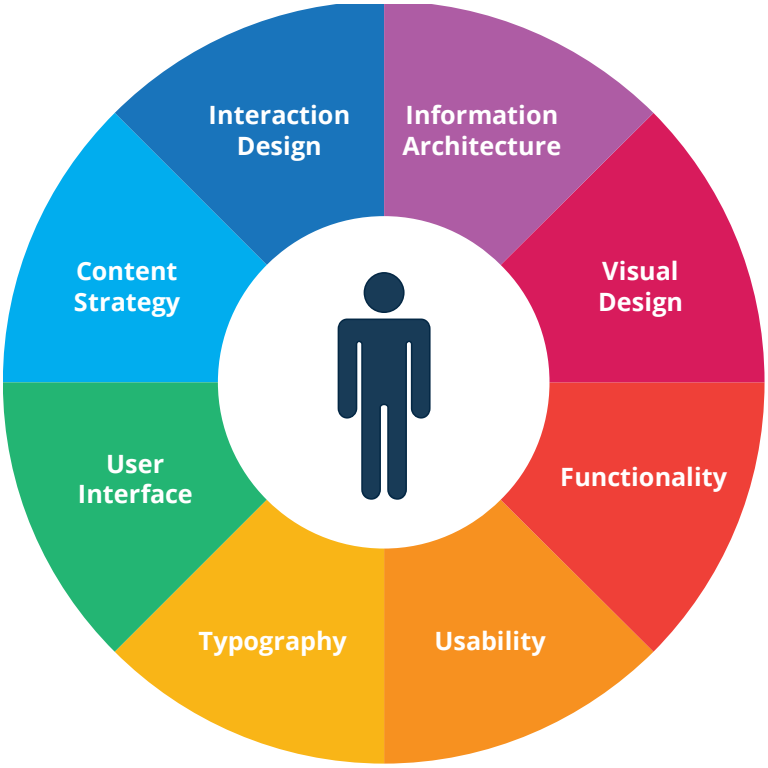
```
BEGIN:VCALENDAR
VERSION:2.0
PRODID:ID
METHOD:REQUEST/CANCEL
BEGIN:VEVENT
DTSTAMP:Datum a čas vytvoření souboru
DTSTART:Datum a čas začátku události
DTEND:Datum a čas konce události
SUMMARY:Název události
UID:ID události
END:VEVENT
END:VCALENDAR
```

3.14 UX a design

V současné době je na vrcholu popularity tzv. Flat Design, tedy plochý design. Jedná se o minimalistický design, který využívá především jednoduché tvary a barvy. Tento design je v široké míře přijat mnoha firmami zabývajícími se vývojem a prodejem operačních systémů, spotřební elektroniky apod., jmenovitě např. Apple, Google, či Microsoft.

Na obrázku 3-5 si můžete povšimnout, které oblasti je potřeba rozvíjet, aby bylo dosaženo dobrého UX. Velkou roli hraje nejen vizuální design, ale také funkcionalita, použitelnost a mnohdy opomíjená typografie.

Obrázek 3-5 Oblasti tvořící UX [13]



Dobře zpracované UX může být faktorem, který rozhoduje o úspěchu a neúspěchu produktu. Proto je potřeba věnovat této části značné úsilí a design neopomíjet.

4 Dokument vize

4.1 Popis a cíle projektu

V současné době je ve firmě CCA Group a.s. využíván e-learnignový systém vlastní výroby (Ramses Akademie), který však při svém věku začíná být značně zastaralý a nesplňuje již všechny požadavky na něj kladené.

Cílem projektu je modernizovat původní e-learningový systém Ramses Akademie za použití moderních technologií a postupů tak, aby byla zvýšena jeho přehlednost, UX a obohacen o nové funkce (streaming, propojení na kalendáře, e-mailová upozornění apod.).

4.2 Seznam funkčních požadavků

Systém bude umožňovat kompletní správu vnitrofiremního vzdělávání, kdy uživatelé mají různé role (Garant vzdělávání, Vedoucí pracovník, Student apod.) a na základě těchto rolí budou mít různé možnosti, jak se systémem pracovat. Další možností bude provádět online kurzy za pomoci streamovaných dat (videa, prezentace, audio nahrávky atd.)

4.2.1 Garant vzdělávání

Garant vzdělávání bude mít možnost vypisovat školení na základě požadavků studentů a vedoucích, vypršení platnosti některých školení (BOZP, školení řidičů apod.) a kvalifikačních plánů. Na takto vypsané školení se již mohou přihlašovat studenti a v případě dostatečné obsazenosti se školení uskuteční. Garant má také možnost školení zrušit v případě, že se např. přihlásí málo studentů. Garant dále zajišťuje výběr školení, spravuje katalog školení, apod.

4.2.2 Vedoucí pracovník

Vedoucí pracovník schvaluje a přiděluje školení svým podřízeným (role studenti) dle jejich vhodnosti pro současnou nebo budoucí pozici podřízeného. Vedoucí pracovník je ale sám také studentem. Vedoucí bude mít k dispozici kalendář se školeními (propojení Gmail, Outlook) a přehled o absolvovaných školeních svých podřízených, včetně jejich případného výsledku v testu. Další možností je zvolit podřízenému novou funkci (dovednost – toto probíhá ústně) a dle toho mu přiřadit

školení, která ho pro danou funkci připraví. O průběhu plnění tohoto plánu bude mít přehled.

4.2.3 Student

Student má možnost se hlásit na jakákoliv školení, avšak o jejich vhodnosti z hlediska osobního rozvoje podřízeného rozhodne vedoucí daného studenta – schválením přihlášky na školení. V případě schválení bude student upozorněn e-mailem a akce se mu vloží do kalendáře.

4.2.4 Školitel

Školitel bude naplňovat výukové materiály školení. Školitel obdrží tištěnou prezenční listinu k podpisu (studenti i školitel) a doplnění dle reálné účasti a odškolí akci. Po skončení školení studenti anonymně ohodnotí kvalitu školení a školitele.

Školitel bude vyplňovat prezenci a případné výsledky dané školící akce.

4.3 Seznam mimofunkčních požadavků

- Vývoj na Java EE 6
- Běh na Glassfish 4.0

4.4 Stakeholders

Jméno	Role	Reprezentant
Ing. Jiří Jansa	Zadavatel projektu / Vedoucí / Uživatel	Zákazník
Ing. Petr Přibyl	Lektor	Uživatel
Pavel Škarban	Garant vzdělávání	Uživatel
Bc. Martin Sýkora	Dodavatel produktu	Dodavatel

4.5 Plán projektu

Pravidelné schůzky každé úterý v 15:30 v Krátká 3, Plzeň

4.6 Zadání projektu

Zanalyzujte potřeby a vytvořte informační systém pro správu a řízení vnitrofiremního vzdělávání. Systém umožní organizaci interních i externích školení, vypisování realizací, žádosti o realizace atd. Součástí práce bude také napojení na externí kalendář formou volitelného pluginu. Součástí bude také zapojení datových zdrojů pro školení, například prezentace, příklady, ale i streamování videa a audio nahrávek. Velká data budou uložena vhodnou technologií. Autor navrhne použitelné a moderní prostředí na platformě Java EE 6 s vyžitím moderních prvků GUI z UI frameworků. Při zpracovávání práce bude kladen velký důraz na snadnost použití, integraci s externími systémy (např. kalendář) a informační hodnotu řešení.

1. Analyzujte požadavky kladené na informační systém správy a řízení vnitrofiremního vzdělávání.
2. Navrhněte GUI s možnostmi streamování a moderními prvky.
3. Navrhněte business logiku aplikace s využitím Java EE návrhových vzorů.
4. Ověřte naplnění požadavků vybraných na základě dohody se zákazníkem praktickou realizací na platformě Java EE 6.
5. Navrhněte rozhraní pluginu pro kalendář.
6. Ověřte funkčnost pluginu kalendáře praktickou realizací pro MS Exchange (Outlook kalendář) a Google (Gmail kalendář).
7. Zhodnoťte dosažené výsledky.

5 Analýza

Hlavními požadavky zadavatele byla přehlednost aplikace, moderní UI a UX, propojení s kalendáři pomocí pluginu a přehrávání multimedií, kde funkčnost má mnohem větší váhu než design a UX. Těmto požadavkům byl podřízen celý návrh aplikace, která se stavěla na zelené louce.

Základní analýza požadavků kladených na informační systém byla provedena a zdokumentována v rámci dokumentu vize projektu, který je uveden v předchozí kapitole.

Technologie použité při tvorbě tohoto projektu byly určeny zadavatelem práce, případně byly doporučeny při osobní konzultaci. Technologie, které byly zvoleny pro tento projekt, byly vybírány s ohledem na celkový koncept aplikace.

6 Návrh aplikace

6.1 Datový model

Vzhledem k tomu, že aplikace musí pro svou funkčnost udržovat data o studentech, kurzech a podobně, rozhodl jsem se pro tato data využít klasickou relační databázi, konkrétně MySQL ve verzi 5.6.11. Mohl jsem volit i další databáze, konkrétně například PostgreSQL nebo databázi od firmy Oracle, ale jednak je MySQL pro účely tohoto projektu zcela dostačující a jednak mi bylo, vzhledem k mým dřívějším zkušenostem s touto databází, také doporučeno vedoucím této práce.

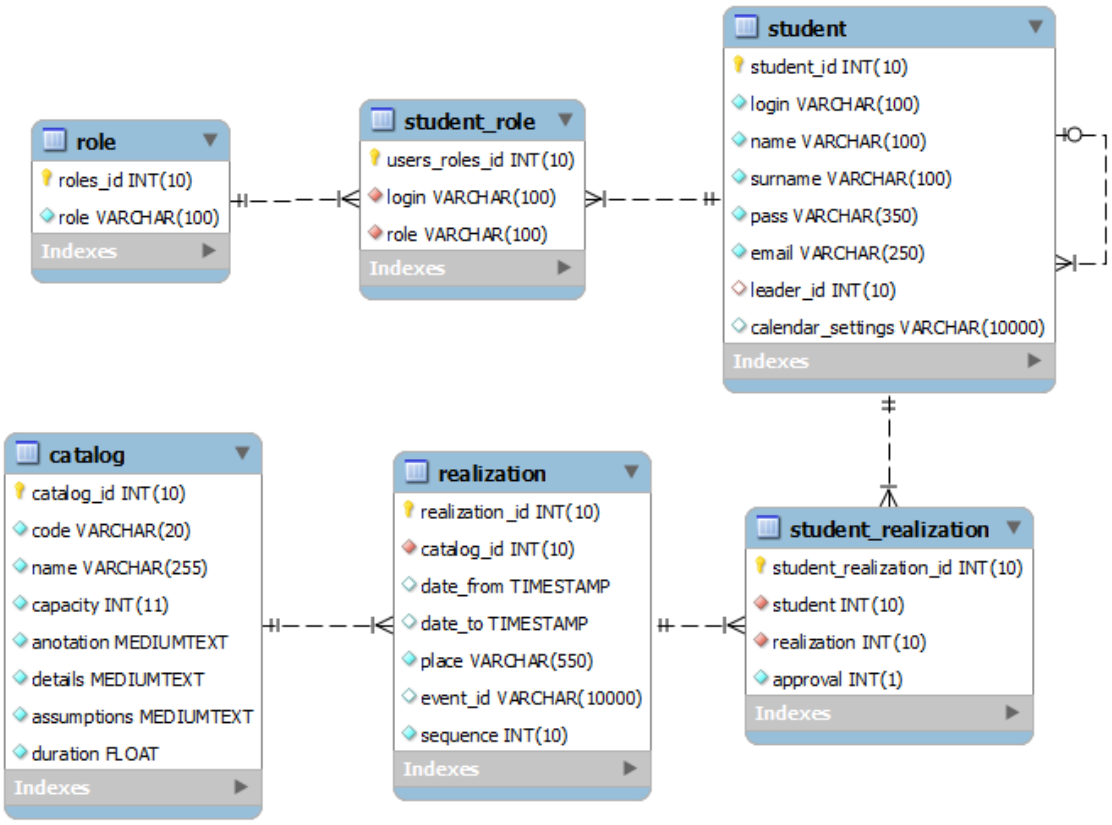
Datový model sestává ze šesti tabulek, které udržují všechna data potřebná k fungování aplikace (*viz* obrázek 6-1). V tabulce *role* jsou umístěny záznamy možných rolí, které mohou být přiřazeny uživateli systému.

Tabulka *student_role* možná překvapivě neudržuje id záznamů z tabulek *role* a *student*, ale hodnoty login a role. Je tomu tak kvůli Glassfish Security Realm, kdy nastavení Glassfishe jasně vyžadovalo hodnoty těchto polí namísto indexů.

Tabulka *student* obsahuje záznamy jednotlivých studentů (uživatelů) aplikace. Z pohledu systému je každý uživatel zároveň studentem, i když jeho role je např. lektor nebo administrátor. V této tabulce je udržováno nastavení kalendářů pro každého uživatele.

Tabulky *catalog* a *realization* pak obsahují záznamy jednotlivých školení, jejichž realizace lze provést a realizace, které jsou vypsány. Tabulka *student_realization* pak udržuje záznamy o přihlášení studentů na jednotlivá školení (realizace).

Obrázek 6-1 Datový model Ramses Akademie 2.0



6.2 UX a design

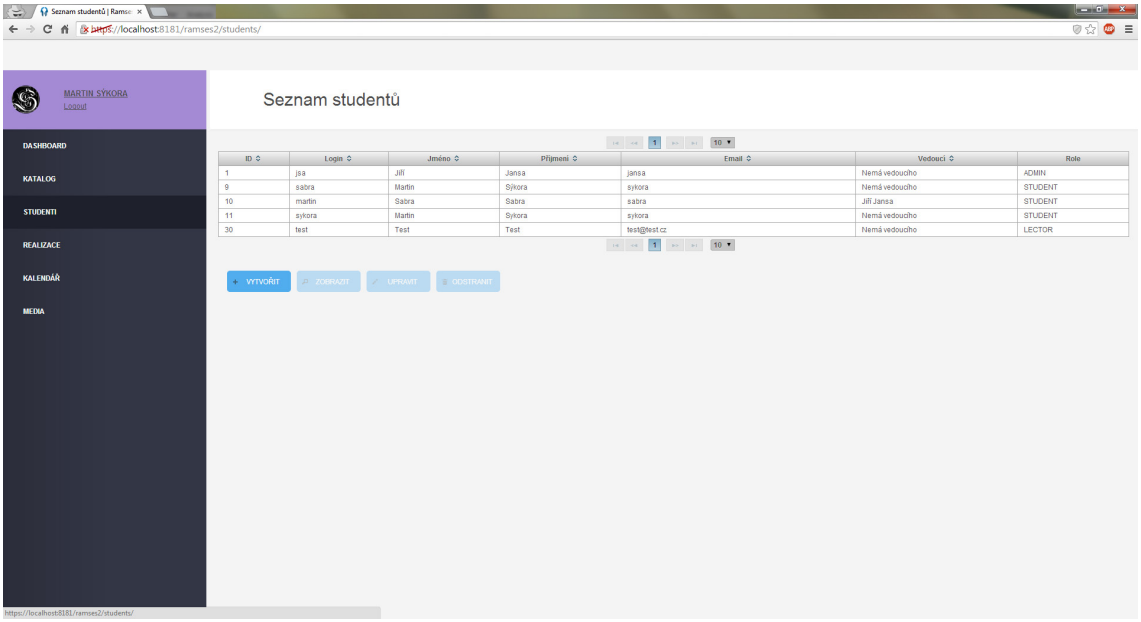
Vzhledem k účelu aplikace jsem se rozhodl pro klasický moderní dashboardový layout, který je bez problému rozšiřitelný o další prvky, které by mohly být v budoucnu požadovány. Grafický návrh začal nejprve tvorbou wireframe (*viz* Příloha B – Wireframe aplikace) a po odsouhlasení vedoucím práce se převedl do kaskádových stylů, využívajících moderní prvky jako přechody (gradient) a další.

Při kódování CSS stylů byl využit preprocesor Sass (zkráceně Syntactically Awesome Style Sheets) [14], který zpřehledňuje a zjednodušuje zápis kaskádových stylů a dovoluje i jejich rozdělení na menší celky, které se pomocí direktiv @import později vkládají do hlavního souboru. Překlad souborů SASS na CSS probíhá v prostředí NetBeans automaticky při každém uložení.

Jelikož rozhraní nebylo nejvyšší prioritou tohoto projektu, není zcela dokončeno a bylo by potřeba jej ještě výrazně vylepšit. Pro základní funkčnost aplikace ale stačí a oproti současnému stavu je, dle mého názoru, velkým krokem kupředu.

Na obrázku 6-2 můžete vidět, jak vypadá nový design aplikace, který se od wireframe z *Příloha B – Wireframe aplikace* mírně liší. Tato odlišnost je způsobena výběrem komponentního frameworku, který obsahuje vlastní styl a kompletní změna tohoto stylu by byla neúměrně časově náročná. Změna je tudíž jen částečná.

Obrázek 6-2 Design nové aplikace



7 Realizace aplikace

V první fázi realizace tohoto projektu jsem vytvářel základní prototyp, který byl schopen vypisovat, měnit a mazat data v databázi. Nejprve jsem tedy vytvořil datový model, podle nějž byly vytvořeny jednotlivé tabulky v databázi. Poté bylo potřeba nastavit Glassfish, tak aby byl schopen komunikovat s databází (*viz* kapitola 7.1 dole). Následně byly, pomocí vývojového prostředí NetBeans, vygenerovány třídy entit z tabulek v databázi.

V dalším kroku byly vytvořeny bezestavové EJB třídy, které obsluhují požadavky backing bean na persistenci, samotné backing beany a HTML stránky. Tímto byl dokončen první prototyp aplikace, na němž jsem se naučil základní práci

s technologiemi, které byly následně využity pro další rozšíření aplikace. Tato rozšíření jsou popsána v následujících kapitolách.

7.1 Nastavení Glassfish serveru

Aby bylo možné využívat JTA[8] bylo potřeba v Glassfishi nastavit parametry potřebné pro spojení s databází. Za tímto účelem byl tedy v kontejneru založen tzv. JDBC Connection Pool. Jelikož vytváření a udržování spojení s databází je drahé, používá se Connection pool, který udržuje již vytvořená spojení. Tato spojení jsou potom využívána stále dokola, tudíž není potřeba vytvářet nová. V případě, že jsou všechna spojení obsazená a je potřeba další, je vytvořeno a zařazeno do poolu. Tím jsou šetřeny systémové prostředky a zrychluje se i přístup k datům v databázi. V nastavení Connection Poolu se také nachází URL databáze, přístupové jméno a heslo, a driver, který se používá pro komunikaci, v tomto případě *com.mysql.jdbc.Driver*.

Takto vytvořený pool ale JPA ještě nemůže využívat. Je potřeba v Glassfishi vytvořit Resource (zdroj), který má přiřazeno JNDI[9] jméno, v tomto případě *jdbc/ramsesDatasource*, a odkazuje se na právě vytvořený pool. Zdroj s tímto jménem se poté nastaví v souboru persistence.xml.

JavaMail Session využívá stejný princip, kdy na serveru je nastaven zdroj pod nějakým JNDI jménem, v tomto případě konkrétně *mailSession* a třída, která tento zdroj využívá si jej mapuje pomocí anotace *@Resource* (*viz* kód 7-1).

Kód 7-1 Mapování zdroje z Glassfishe na proměnnou v objektu

```
@Resource(name = "mailSession")
private Session mailSession;
```

Nastavení jak Connection poolu, tak i JavaMail Session, je nakonfigurováno v souboru glassfish-resources.xml. Toto nastavení by se mělo automaticky přenést do

[8] Java Transaction API

[9] Java Naming and Directory Interface – Jmenná a adresářová služba, díky níž lze pomocí jména najít data a objekty

nastavení Glassfish serveru při zavádění aplikace, bohužel se tak při mém testování nikdy nestalo.

7.2 Security

Webové aplikace postavené na bázi Java Server Faces umožňují dva způsoby přihlašování pomocí formuláře. Je to přihlašování za využití ManagedBeany a přihlašování pomocí j_security_check. Druhý zmíněný přístup jsem využil při tvorbě této aplikace, protože nebylo potřeba vytvářet beanu a zabývat se její vnitřní logikou. J_security_check je akce pro formuláře, kterou definuje Java EE Security, umožňuje tak kontejneru autentikovat uživatele.

V případě, že se uživatel snaží vstoupit do zabezpečeného prostoru a není přihlášen, je mu automaticky zobrazen přihlašovací formulář. Tento formulář musí být tvořen běžnými HTML tagy, protože faceletové formuláře si samy generují formulářovou akci i ID prvků. Pro funkci j_security_check je však nutné, aby formulář měl akci nastavenu na j_security_check a vstupní pole pro uživatelské jméno/login a heslo mělo nastaveno property name na j_username a j_password (*viz* kód 7-2).

Kód 7-2 Formulář j_security_check

```
<form method="POST" action="j_security_check">
    <input id="loginName" type="text" name="j_username"
        placeholder="#{bundle.LoginName}" />
    <input id="loginPass" type="password" name="j_password"
        placeholder="#{bundle.LoginPass}" />
    <input type="submit" value="#{bundle.LoginButton}" class="button" />
</form>
```

Kód 7-3 Nastavení zabezpečení ve web.xml

```
<security-constraint>
    <display-name>RamsesAcademy</display-name>
    <web-resource-collection>
        <web-resource-name>WholeWeb</web-resource-name>
        <description>Přístup na celý web</description>
        <url-pattern>/*</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
```

```
        <auth-constraint>
            <description/>
            <role-name>student</role-name>
            <role-name>lector</role-name>
            <role-name>garant</role-name>
            <role-name>leader</role-name>
            <role-name>admin</role-name>
        </auth-constraint>
        <user-data-constraint>
            <description/>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
    </security-constraint>
    <security-constraint>
        <display-name>ResourcesConstraint</display-name>
        <web-resource-collection>
            <web-resource-name>resources</web-resource-name>
            <description/>
            <url-pattern>/javax.faces.resource/*</url-pattern>
        </web-resource-collection>
    </security-constraint>
    <login-config>
        <auth-method>FORM</auth-method>
        <realm-name>jdbc-realm</realm-name>
        <form-login-config>
            <form-login-page>/login.xhtml</form-login-page>
            <form-error-page>/login_error.xhtml</form-error-page>
        </form-login-config>
    </login-config>
```

V příkladu výše (*viz* kód 7-3) je možné vidět nastavení zabezpečení aplikace. V části *security-constraint* jsou dva důležité elementy. Jsou jimi *web-resource-collection*, kde je nastaveno schéma URL a HTTP metody, které mají být chráněny, a níže je element *auth-constraint*, který obsahuje seznam rolí, které na URL uvedené ve web-resource-collection mají přístup po úspěšném přihlášení.

V části login-config je pak možné vidět nastavení autentizační metody a zdroj dat, který jsou k této autentizaci využívána. V tomto případě se jedná o jdbc-realm, což jméno bezpečnostní oblasti nastavené v Glassfish serveru, využívající *jdbc/RamsesDatasource* k přístupu k databázi.

7.3 Plugin

V kapitole 3.13.1 jsem uvedl několik strategií, které je možno využít při návrhu pluginu. Z těchto strategií se jako nejvhodnější jevila strategie „Strategie“, která je velmi podobná návrhovému vzoru „Strategie“ od Gang of Four [15]. Tato strategie umožňuje definovat jednotné rozhraní, které poté implementují jednotlivé pluginy, jenž se však mohou zcela zásadně lišit v postupu, kterým dosáhnou podobného cíle než jiný plugin v této rodině. Tento případ se tudíž hodí pro náš problém, kdy potřebujeme provést stejnou akci, avšak s jiným výsledkem (Google vs Outlook).

K implementaci této strategie, stejně jako zbylých dvou, které je možno při programování pluginu využít, je tedy potřeba vytvořit rozhraní (API), které bude sloužit jako ústřední spojovací bod mezi aplikací a pluginy. Toto rozhraní by mělo být co nejstabilnější a velmi dobře promyšlené. Jakmile je totiž nad ním implementován nějaký plugin, nelze ho již měnit bez patřičné změny v tomto pluginu. Součástí API mohou být kromě samotného rozhraní také další třídy potřebné pro komunikaci mezi pluginy a aplikací, na zpracování dat, apod. Každý plugin potom toto rozhraní implementuje a pomocí něj komunikuje s aplikací (*viz* obrázek 7-1). Jak API, tak každá implementace musí být zabalena do samostatného JARu[10]. Aplikaci je poté možno, pomocí těchto archivů s implementací daného rozhraní, snadno rozšiřovat o nové funkce.

Jak aplikace, tak i pluginy vyžadují modul s API, aby je bylo možné zkompilovat. Pro aplikaci je to zároveň také jediná závislost. Aplikace totiž na samotných pluginech nesmí být závislá a musí být možné ji zkompilovat a spustit i bez těchto implementací daného rozhraní. Pluginy se do aplikace přidávají až při jejím sestavování a aplikace bude plně funkční i bez přídavných modulů, samozřejmě však bez funkcí, které poskytují chybějící pluginy. Pro snadnost sestavování aplikace s pluginy se však tyto přidávají jako dependency do *pom.xml*.

O nalezení a přiřazení pluginů k API se stará kontejner během zavádění nebo spouštění aplikace. Pluginy nemusejí být nikde nastavovány a aplikace o nich nemusí nic vědět.

[10] Java Archive

Kód 7-4 Nepřímá injekce rozhraní pluginu a průchod všech implementací

```
@Inject @Any
Instance<CalendarPlugin> calendarPlugin;
…
for (CalendarPlugin calendar : calendarPlugin) {
    …
}
```

Pomocí anotace (*@Inject @Any*) ve výše uvedeném příkladu (kód 7-4) jsou nepřímo injektovány všechny implementace rozhraní CalendarPlugin, které kontejner nalezne během zavádění aplikace. Jelikož rozhraní *Instance* dědí od rozhraní *java.lang.Iterable*, je možné všechny načtené pluginy procházet pomocí *for-each* smyčky.

Aby byly pluginy funkční je potřeba, aby stroj, na němž bude aplikace spuštěna, měl přístup k internetu.

7.3.1 Nastavení pluginů

Každý plugin vyžaduje jiné parametry pro nastavení a aby požadované parametry mohla zjistit i aplikace, implementují všechny pluginy metodu getKeys. Metoda getKeys tedy vrací sadu objektů Key, což je třída, která sestává ze dvou parametrů: id (identifikátor klíče) a desc (popis tohoto klíče). Tato sada je vytvořena v konstruktoru každého pluginu.

Jelikož pluginů implementujících rozhraní kalendáře může být neomezené množství, nelze mít pro každý parametr připraven speciální sloupec v tabulce databáze, protože by to odporovalo smyslu pluginu. Řešením je tedy zavést jeden sloupec, který bude dostatečně prostorný a bude udržovat nastavení všech kalendářů. Kalendáře se samozřejmě nastavují pro každého uživatele, sloupec s nastavením je tedy v tabulce *student* (*viz* obrázek 6-1).

Nastavení v databázi se uchovává v podobě (*viz* kód 7-5), kterou lze snadno, pomocí regulárních výrazů a funkcí pro práci s řetězci, převést na mapu *Map<String, Map<String, String>>*. V tomto formátu je nastavení načteno do aplikace a jednotlivé pluginy si vždy vyberou nastavení, které přísluší právě jim, pomocí svého ID. V případě

změny nastavení se řetězec pro uložení do databáze získá voláním funkce „toString“ nad mapou.

O převod z a do serializovaného tvaru se stará třída ConfigurationManager, která je součástí API. K serializaci nastavení bylo využito funkce toString, která mapu převede do formátu, který můžete vidět níže. Bylo tedy potřeba vytvořit funkci, která dokáže tento řetězec deserializovat. V této funkci bylo využito regulárních výrazů, pomocí nichž byl celý řetězec rozdělen tak, aby z něho mohla být opět vytvořena mapa *Map<String, Map<String, String>>*.

Kód 7-5 Serializovaná hodnota nastavení pluginů

```
{id_pluginu={klíč=hodnota, …}, id_pluginu={…}, …}
```

Podobným způsobem je následně řešeno i ukládání ID jednotlivých událostí (školení).

7.3.2 Převod událostí

Aby bylo možné pluginům, které mají různou funkčnost předávat událost z aplikace k dalšímu zpracování, bylo potřeba vytvořit jednotnou strukturu, které budou rozumět jak pluginy, tak samotná aplikace. Z tohoto důvodu byla v balíku s rozhraním pluginů vytvořena třída CalendarEvent, do níž se převádějí události z aplikace před odesláním pluginům a ty si ji následně převedou na události specifické pro svou implementaci. Tato třída je blíže popsána v následující kapitole.

7.3.3 Balík cca.ramses2.calendar.api

V tomto balíku se nacházejí všechny třídy, které jsou nutné pro správné fungování pluginu.

CalendarConfiguration

Třída, která udržuje nastavení kalendáře, s kterým se aktuálně pracuje a který je předáván pluginům pro jejich činnost. Při inicializaci načte od všech nalezených pluginů seznam jejich požadovaných nastavení a vytvoří z něj mapu, jejíž hodnoty jsou prozatím prázdné. Tyto hodnoty se, po zavolání funkce *merge*, naplní hodnotami nastavení uživatele, které jsou načteny z databáze a deserializovány.

CalendarEvent

Tato třída je standardní POJO[11] a slouží k převodu událostí z formátu, který zná pouze aplikace na jednotný formát pro všechny pluginy. Pro snadnost použití obsahuje konstruktor s parametry a standardní sadu getterů a setterů pro všechny parametry třídy.

CalendarPlugin

Rozhraní, které musí implementovat každý plugin. Definuje metody pro přidání, úpravu a mazání událostí a dále metody pro získání ID jednotlivých pluginů, jejich nastavení, požadovaných pro správnou funkčnost, a uživatelsky přívětivý popis. Diagram implementace tohoto rozhraní můžete vidět na obrázku 7-1.

ConfigurationManager

Třída, která se stará o serializaci a deserializaci nastavení kalendářů pro jednotlivé uživatele.

Key

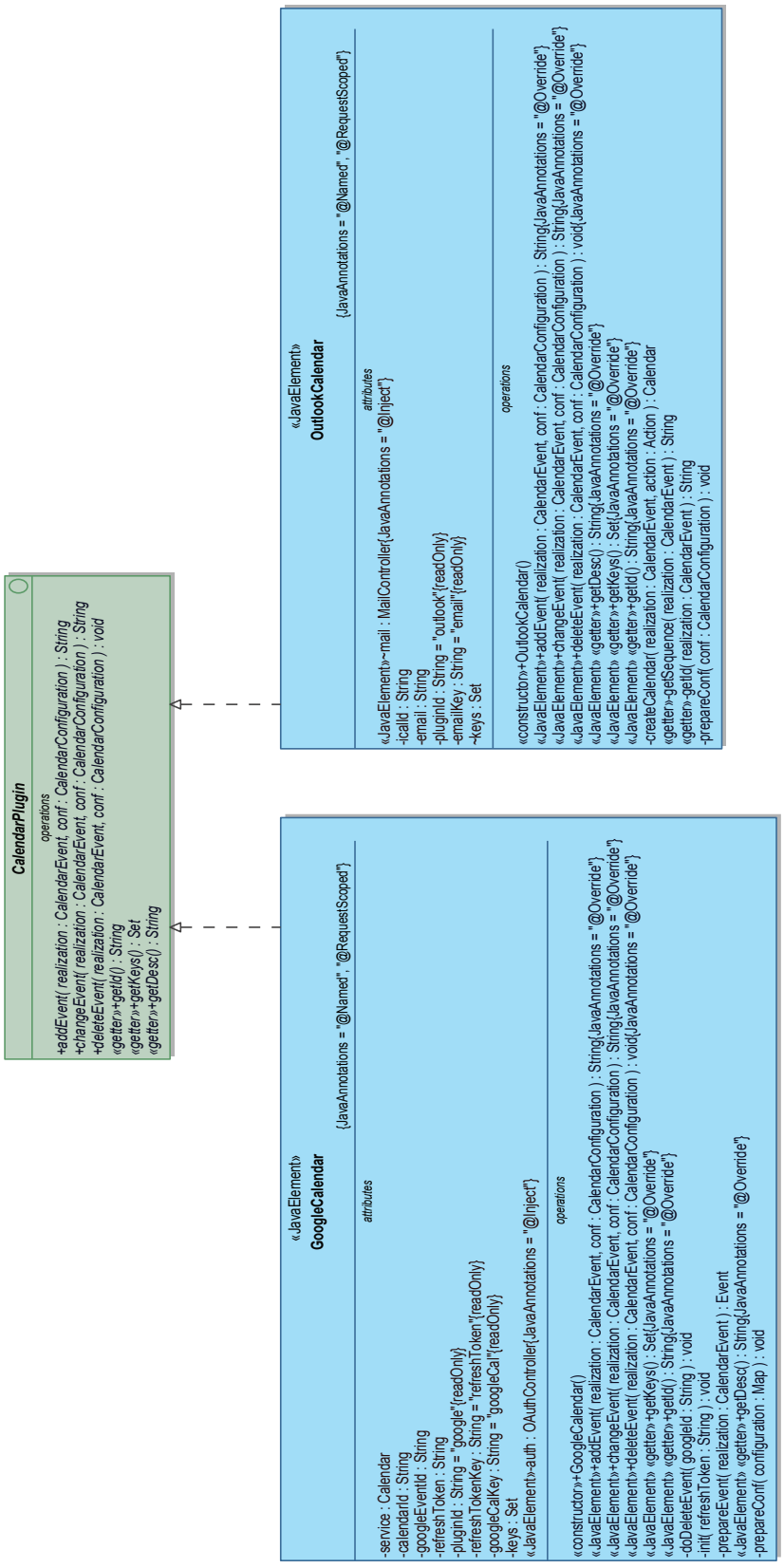
Opět standardní POJO třída, do níž pluginy vkládají ID a popis klíčů nastavení, které vyžadují. Seznam těchto klíčů je využit k vytvoření prázdného nastavení a jeho následnému naplnění uživatelem v hlavní aplikaci.

OAuthController

Tato třída zajišťuje autorizaci uživatelů do Google kalendáře pomocí OAuth protokolu. Obsahuje funkce pro načítání credentials, přesměrování na autorizační URL i odvolání obnovovacího tokenu.

[11] Plain Old Java Object

Obrázek 7-1 Diagram implementace rozhraní pluginu



7.4 Google kalendář

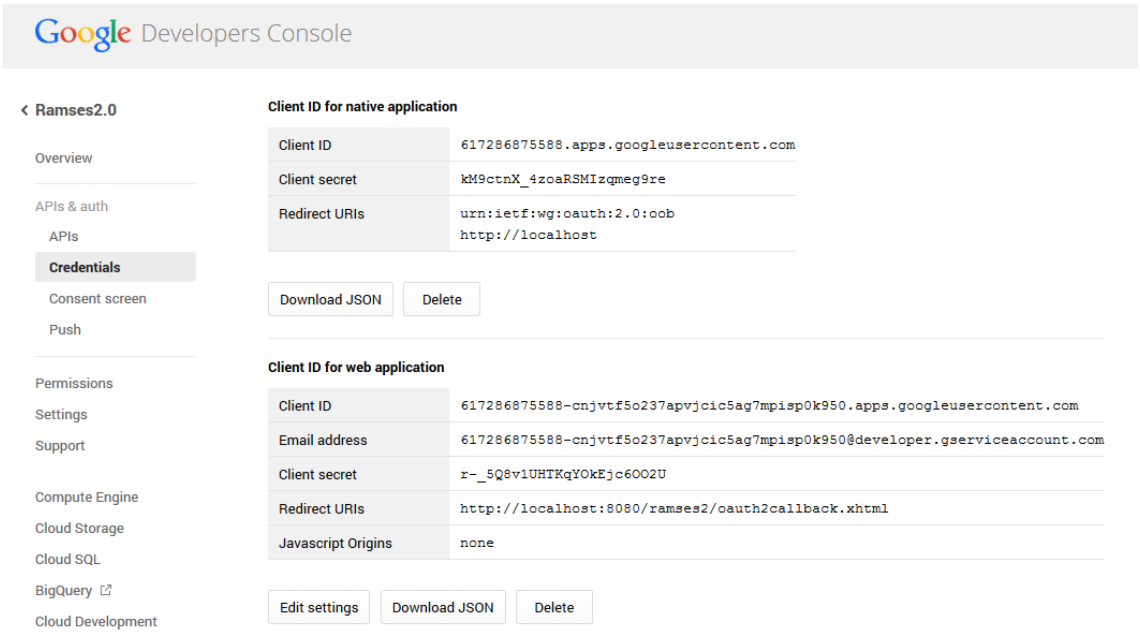
Implementace tohoto pluginu byla jedním z nejsložitějších a zároveň nejzásadnějších úkolů této práce. Prvním závažným problémem byla špatná dokumentace Google, která mnohdy uváděla zastaralé a již nefunkční postupy pro používání jejich API, které se v případě kalendáře nachází ve verzi 3. Druhým problémem bylo používání OAuth 2.0, kde opět sehrála roli špatná dokumentace, ale oba problémy se podařilo zdárně vyřešit.

Prvním krokem, který bylo potřeba uskutečnit, aby bylo možné používat Google API, je registrovat aplikaci v Google Developers Console (dála GDC). Aplikaci je možno přidat buď do nějakého stávajícího projektu GDC nebo nový projekt vytvořit. Pro tuto aplikaci bylo potřeba projekt vytvořit. Poté bylo nutné v GDC v části API aktivovat ta API, která chceme v projektu využívat (v tomto případě tedy Calendar API). Na využívání API se vztahují kvóty, konkrétně na kalendář je to 100.000 požadavků denně. Tuto kvótu je však možno navýšit, ovšem touto možností jsem se hlouběji nezaobíral, jelikož přidělená kvóta byla, dle mého názoru, více než dostatečná.

Při vytvoření projektu v Google Developers Console obdrží tento vygenerované ID. Pro projekt je dále nutné vytvořit tzv. credentials (doklady), pomocí nichž se naše aplikace „autorizuje“ projektu v Google Developers Console. Nastavuje se zde také tzv. callback URL, tedy URL, kam má být směrována odpověď s autorizačním kódem uživatele, který bude následně aplikací vyměněn za přístupový a obnovovací token pro daného uživatele (pro diagram autorizačního procesu *viz* obrázek 3-4). JSON[12] soubor s těmito údaji je poté potřeba stáhnout a uložit do pluginu, který z něj bude tato data extrahovat a používat pro komunikaci se servery Google. Pro ilustraci nastavení credentials v GDC *viz* obrázek 7-2.

[12] JavaScript Object Notation

Obrázek 7-2 Nastavení credentials v Google Developers Console



Poté již bylo možno pro tento plugin vytvořit nový projekt a do dependencies vložit závislosti na knihovně s rozhraním pluginů a knihovnách Google Calendar API a Google APIs Client. Tyto knihovny poté umožní využívat všechny funkce API potřebné pro přidání, odebrání a změnu událostí z uživatelského kalendáře a samozřejmě implementovat rozhraní pluginu.

Ještě předtím, než je možné začít plugin využívat, je ale nutné, aby již uživatel měl access nebo refresh token a mohl tak API Googlu využívat. Bez tohoto tokenu by to nebylo možné. Refresh token lze získat v nastavení uživatele v části Google kalendáře, kde je potřeba kliknout na tlačítko „přihlásit“. Autorizační proces pak uživatele přesměruje na stránku přihlášení Google (pokud není přihlášen, jinak se tento krok přeskočí) a následně se mu zobrazí formulář pro přijetí oprávnění dané aplikace (*viz* obrázek 7-3). Po přijetí je přesměrován zpět na výchozí bod a zbytek procesu proběhne v pozadí. Více informací o implementaci tohoto autorizačního procesu je uvedeno v následující kapitole. Když je uživatel autorizován, může plugin začít plně využívat.

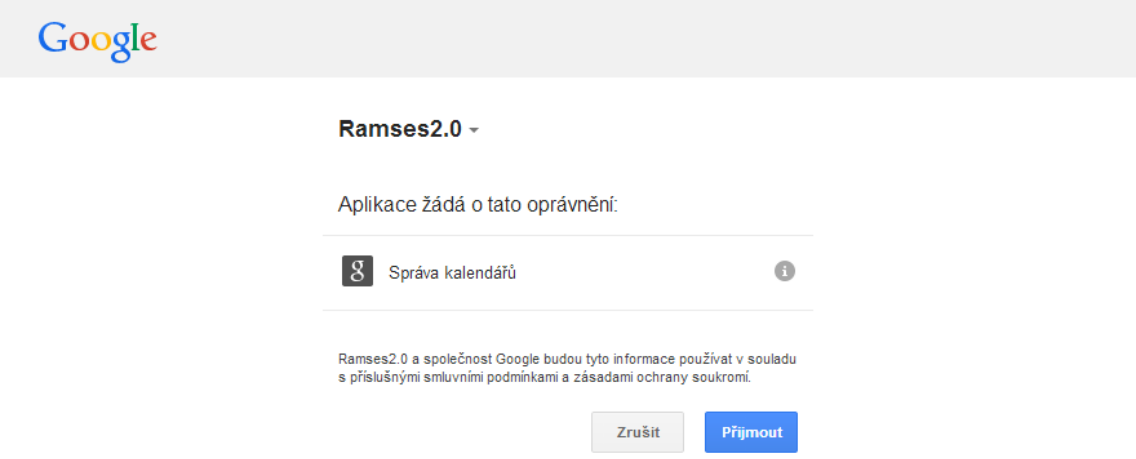
Každá funkce pluginu, kromě funkce pro vymazání události z kalendáře, vrací ID události, které bylo vygenerováno Google kalendářem, a pod nímž je událost v tomto kalendáři evidována. ID události je složeno ze znaků base32hex kódování, tzn. malých

písmen a – v a čísel 0 – 9 (definováno RFC2938). ID musí být dlouhé 5 – 1024 znaků a v kalendáři musí být unikátní.

V případě, že je událost z kalendáře „vymazána“ nedojde k jejímu úplnému vymazání, ale je pouze skryta. Takováto situace může způsobit řadu problémů, jelikož může dojít ke kolizi ID událostí, což vrátí chybu. Může dojít například k tomu, že uživatel se v aplikaci přihlásí na realizaci nějakého školení, plugin mu tedy toto školení zapíše do kalendáře. Uživatel se následně odhlásí a plugin „smaže“ tuto událost z kalendáře. V případě, že by se uživatel pokusil znovu přihlásit, došlo by ke kolizi ID, protože plugin by se snažil do kalendáře vložit událost s ID, které v kalendáři již je, i když skryté. Je tedy vždy potřeba kontrolovat, zda se událost s tímto ID již v kalendáři nenachází a v případě, že ano, ji pouze „updatovat“. V případě, že událost ještě neexistuje, je možno ji bez problému vytvořit.

Další problém může nastat, přihlásí-li se uživatel na událost a přihlášení do Google kalendáře provede až poté. V případě, že by garant tuto událost následně upravil, například změnil datum, snažil by se plugin změnit neexistující událost v kalendáři přihlášeného uživatele. I v takovém případě tedy bylo nutné kontrolovat, zda již událost existuje či nikoliv a dle toho provést vložení nové události nebo aktualizaci již zaznamenané.

Obrázek 7-3 Autorizační obrazovka pro přístup ke Google kalendáři uživatele



S tímto kalendářem je, jako s jediným, také potřeba počítat v aplikaci, resp. v části nastavení kalendářů v hlavní aplikaci. Jelikož se jedná o velmi specifickou

implementaci, musí na ni být připravena jak backing beana, tak i JSF stránka. Tento plugin totiž nelze nastavovat „ručně“, protože v pozadí probíhá autorizační proces, který ve výsledku vrátí nebo nevrátí (závisí na uživatelově akci) přístupový a refresh token ve formě HTTP odpovědi, který by měl být uchován v databázi. Není tedy cesty, aby se k těmto údajům běžný uživatel dostal a tokeny kopíroval do input polí formuláře.

Všechny ostatní pluginy je ale možno používat bez jakýchkoliv zásahů do hlavní aplikace, pokud tedy nepotřebují speciální přístup k řešení nastavení stejně jako tento plugin.

7.4.1 OAuth 2.0

Implementace tohoto protokolu byla zcela zásadní pro fungování pluginu pro Google kalendář. Je mu tedy vyhrazená celá tato kapitola. Jak tento protokol funguje je teoreticky rozvedeno v kapitole *3.14.1 Google kalendář* a v této kapitole se zabývám samotnou implementací. OAuth protokol se používá pro získání přístupových tokenů ke kalendáři uživatele, bez nichž by aplikace nemohla zapisovat a měnit zapsané události.

Protokol funguje tak, že uživatel je přesměrován na nějaké URL, kde se přihlásí do Google účtu a potvrdí aplikaci oprávnění, o která žádá. O vytvoření tohoto URL se stará GoogleAuthorizationCodeFlow (*viz* kód 7-6). Nejprve je vytvořena instance této třídy, která je součástí Google Client API knihovny, kdy při vytváření jsou nastaveny všechny důležité parametry. V ukázkovém kódu si můžete povšimnout volání funkce getClientSecrets(), která vrací načtené credentials neboli doklady, které byly získány z Google Developers Console a o kterých jsem se zmiňoval v předchozí kapitole. Dále se nastavují tzv. scopes, které určují, k jakým zdrojům bude mít aplikace přístup (za předpokladu, že jej uživatel umožní).

Kód 7-6 Sestavení URL

```
GoogleAuthorizationCodeFlow authFlow = GoogleAuthorizationCodeFlow.Builder(
        HTTP_TRANSPORT, JSON_FACTORY, getClientSecrets(),
        Arrays.asList(CalendarScopes.CALENDAR))
        .setAccessType("offline")
        .setApprovalPrompt("auto").build();
String url = authFlow.newAuthorizationUrl()
```

```
        .setRedirectUri(REDIRECT_URI).build();
```

Po přijetí oprávnění je uživatel přesměrován na adresu, která je uvedena v REDIRECT_URI. Na této adrese je zachycen autorizační kód, který je obratem vyměněn za přístupový token (*viz* kód 7-7) a v tomto případě také za obnovovací token. Uživatel toto přesměrování ani nepostřehne a je ihned vrácen na výchozí stránku nastavení svého účtu, odkud požadoval přihlášení. Přístupový token, který aplikace obdrží, má ale velmi krátkou životnost a po jejím vypršení by uživatel tuto aplikaci musel znovu autorizovat. Nastaví-li se tedy v GoogleAuthorizationCodeFlow typ přístupu na „offline“, jak můžete vidět v kódu výše, obdrží aplikace také obnovovací token. Tento token umožňuje dlouhodobý přístup aplikace k uživatelským datům.

Kód 7-7 Výměna autorizačního kódu za tokeny

```
String authCode = request.getParameter("code");
GoogleAuthorizationCodeTokenRequest requestToken =
            authFlow.newTokenRequest(authCode);
GoogleTokenResponse response = requestToken.execute();
```

Uživatel se také může rozhodnout, že aplikaci zruší oprávnění přístupu ke svým datům v Google kalendáři. V takovém případě by musel refresh token smazat z nastavení svého účtu v aplikaci. Tuto možnost samozřejmě má, ale dobrým přístupem by bylo zrušit toto oprávnění i v účtu Google. Aby tuto možnost nemusel složitě hledat v nastavení účtu Google, má tuto možnost v aplikaci. V nastavení svého uživatelského účtu v Ramses Akademii kde se do Google účtu přihlašoval, se v případě, že aplikace má k dispozici uživatelův refresh Token, změní tlačítko na odhlašovací, které zruší platnost tokenu nejen v aplikaci, ale i v Google.

7.5 Outlook kalendář

Jak již bylo zmíněno výše, v kapitole 3.14.2, Outlook umožňuje práci se soubory formátu iCalendar, který je dán standardem RFC 5545. Této možnosti bylo využito při návrhu a realizaci pluginu pro tento kalendář. Druhou možností by byla snaha o přímé propojení s Microsoft Exchange, leč bez přístupu k Exchange serveru nemohl probíhat vývoj ani testování. Zadavateli však plně vyhovovala možnost použití iCal souborů, protože tuto možnost ve firmě znají a využívají.

Pluginu je, při volání každé funkce obsluhující kalendář, předloženo deserializované nastavení všech kalendářů, z nějž si vybere své nastavení a dále objekt třídy CalendarEvent, která je součástí API pluginů a obsahuje údaje o události.

Plugin si poté pomocí své funkce *createCalendar* převádí data z CalendarEvent na iCal formát za pomoci knihovny iCal4j [16]. Knihovna iCal4j vytvoří objekt Calendar, do nějž se poté vkládají potřebné hodnoty.

Při vytváření iCal souboru je, krom jiného, nutné definovat dvě věci, a sice akci, to znamená, jestli se událost vytváří, upravuje nebo ruší a sekvenční číslo této události (celočíselná hodnota), které udává, kolikátá změna v pořadí se u této události provedla. Sekvenční číslo musí být vždy alespoň o 1 větší než při předchozím přidání či úpravě události. Podle akce se poté do iCal souboru přidává označení metody a případně další parametry, které určují, co se s událostí má provést u cíle. Metody mohou být REQUEST (přidání/úprava) a CANCEL (zrušení). V případě metody CANCEL je potřeba samotné události přidat ještě „Status: CANCELED“. Formát iCal souboru můžete vidět v kódu 7-8.

Kód 7-8 Formát souboru iCal pro přidání události

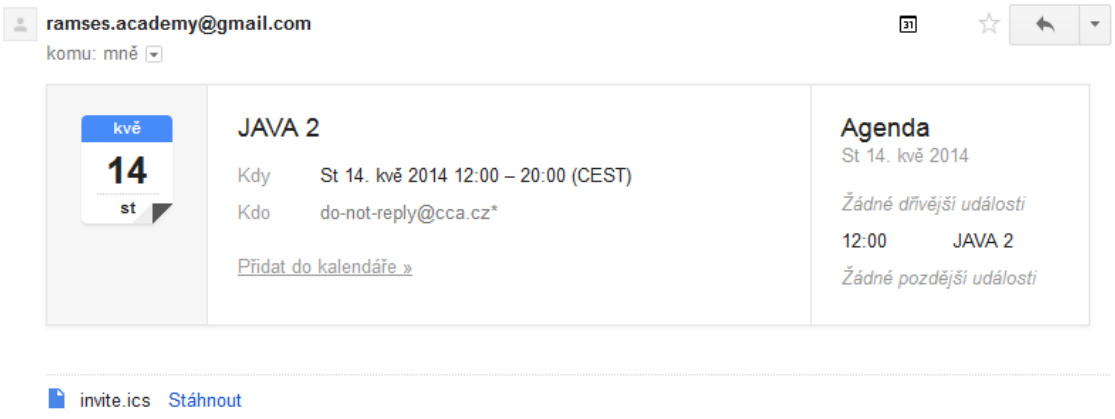
```
BEGIN:VCALENDAR
PRODID:Ramses Akademie
VERSION:2.0
METHOD:REQUEST
BEGIN:VEVENT
DTSTAMP:20140421T143226Z
DTSTART:20140422T180000
DTEND:20140422T200000
SUMMARY:JAVA 3
SEQUENCE:5
UID:20140422T180000-9
ORGANIZER:mailto:do-not-reply@cca.cz
END:VEVENT
END:VCALENDAR
```

Další potřebnou věcí je ID události, které jednoznačně určuje událost v kalendáři uživatelů, ale to je již obsaženo v objektu CalendarEvent, a pokud ne, vytvoří se nové za použití předem daného pravidla. Díky ID události a sekvenčnímu číslu je možné

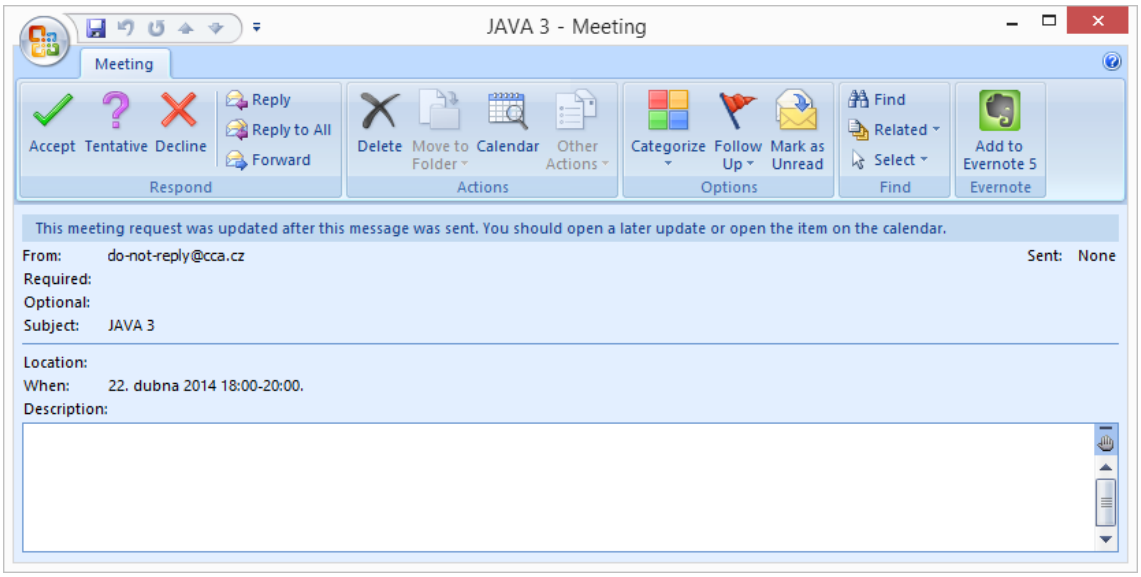
provádět v kalendářích uživatelů změny a rušení událostí. Funkce pro přidání nebo změnu události toto ID vždy vrací volající funkci, která si jej, v případě, že před odesláním ještě neexistovalo, uloží do databáze. V databázi je u každé události uchováváno vždy jedno ID pro každý plugin. Stejné ID události je u různých uživatelů možné použít bez problémů, ušetří se tak místo v DB, protože jinak by bylo potřeba uchovávat speciální ID pro každou událost pro každého uživatele.

Je také nutné, aby byl v iCal uveden e-mail organizátora, jinak se pozvánka nezobrazuje správně. Výsledek správného zobrazení iCal souboru v Gmailu můžete vidět na obrázku 7-4 a v MS Outlook na obrázku 7-5.

Obrázek 7-4 Zobrazení příchozího mailu s iCal souborem v Gmailu



Obrázek 7-5 Okno MS Outlook 2007 pro přidání nebo odmítnutí události



Když je objekt Calendar naplněn všemi důležitými daty, je, společně s e-mailem příjemce, předán objektu mailController. MailController sestaví e-mailovou zprávu, objekt Calendar přiloží jako přílohu a následně odešle. K odeslání e-mailu se využívá JavaMail API v součinnosti s účtem Google mailu. Nastavení a přístup k tomuto účtu je zapsáno v JavaMail Session, která je uložená v „Resource“ Glassfish serveru a mailController k ní přistupuje pomocí anotace *@Resource*, která podle jména Session toto nastavení najde a načte. Pak již nic nebrání odeslání zprávy do schránky příjemce.

7.6 Ukládání multimediálních souborů

Přesto, že zadavatel práce počítal s ukládáním multimediálních souborů na disk, původní domněnkou při analýze aplikace bylo, že pro ukládání velkých souborů se bude používat Hadoop nebo MongoDB. V průběhu zkoumání těchto technologií jsem však zjistil, že mohou být, pomalejší, než streamování z běžného disku.

Další možností, která tedy připadala v úvahu, bylo použití Java Content Repository, což je technologie specifikovaná v dokumentech JSR170 a JSR283. Implementací této technologie je obsahový repositář Apache Jackrabbit, či JBOSS modeShape. Tyto repositáře umožňují ukládání hierarchických dat, jejich verzování, zamykání, sdílení, fulltextové vyhledávání apod. I tato možnost však byla, po poradě s vedoucím této práce, opuštěna kvůli časové náročnosti.

Konečným řešením tedy bylo ukládání na disk a streamování obsahu odtud. Aby bylo možné data z disku využívat, bylo potřeba jej namapovat do Glassfishe do souboru glassfish-web.xml jako další property (*viz* kód 7-9). Hodnota této property se skládá ze dvou částí a sice *dir* a *from*. Část *from* udává vzor URL, jehož volání se bude mapovat na *dir*, který udává cestu k adresáři, v němž se nachází obsah.

Kód 7-9 Přidání adresáře s medii do Glassfishe

```
<property name="alternatedocroot_1" value="from=/output/* dir=[adresář]"/>
```

Za předpokladu, že aplikace je nasazena na adrese http://localhost/ramses2/ bude pak požadavek s url http://localhost/ramses2/output/INT-JAVA2 namapován na zdroj, který je na adrese [adresář]/output/INT-JAVA2. Část from tedy udává URL pattern,

který je potřeba dodržovat i v aplikaci při načítání a streamování souborů, aby došlo ke správnému mapování těchto požadavků.

7.7 Streamování multimediálních souborů

Streamování multimediálních souborů má dvě fáze. Tou první je načtení souborů, které se budou přehrávat a tou druhou je již samotné přehrávání.

Nejprve je ale potřeba mít soubory, které chceme streamovat. Jelikož každé školení z katalogu bude mít svůj obsah, je potřeba každému vytvořit na disku, v adresáři, který je definován v property s názvem alternatedocroot_1 v nastavení glassfish-web.xml, složku pojmenovanou kódem školení a naplnit ji obsahem. Jediným omezením, na které je při nahrávání obsahu nutné pamatovat je fakt, že videa lze přehrávat pouze ve formátu MP4 a název souboru zatím nesmí obsahovat mezery.

Poté je potřeba vytvořit stránku, která bude pro přehrávání médií připravena. Tato stránka před samotným vyrenderováním požádá třídu MediaController o seznam médií z umístění, které je uvedeno v parametru URL stránky. MediaController projde dané umístění a všechny nalezené soubory uloží do mapy, jejímž klíčem je cesta k souboru a hodnotou pak přípona tohoto souboru. Stránka pak tuto mapu obdrží a projde všechny její položky. Podle přípony souboru se pak ve stránce rozhoduje, v jakém tagu (zda <video>, <audio>, či <p:media>), bude cesta k tomuto souboru vypsána. Tím je docíleno toho, že každý soubor je ve správném HTML tagu a že například PDF soubor není zobrazován tagem <video>, což by nebylo možné.

Při žádosti uživatele o začátek přehrávání souboru je tento požadavek odeslán na URL servletu který jej zachytí a zpracuje. Z URL získá název a cestu k požadovanému souboru. Poté již jen po bytech čte BufferedInputStream daného souboru a dokud je co číst, tak jej vypisuje na OutputStream HTTP odpovědi. Na konci streamování je potřeba jak input, tak output stream uzavřít.

MediaServlet se náchazel ve dvou verzích, kdy první verze umožňovala bezproblémové streamování souborů, jak je popsáno výše, avšak po přehrání videa již nebylo možné jej přehrát znovu a během přehrávání nebylo možné posouvání v čase. Při zkoumání proč tomu tak je jsem zjistil, že je potřeba klientovi sdělit, že server dokáže zpracovávat bytové rozsahy. Toho se dosáhne tím, že server do odpovědi vloží

hlavičku Accept-Ranges s hodnotou „byte“. Klient poté může žádat jen určitou část souboru (za pomoci hlavičky Range) a server mu, v případě že je rozsah souboru který žádá platný, odpoví. Tato odpověď pak obsahuje stavový kód 206 Partial Content.

Během hledání této odpovědi jsem narazil na FileServlet, napsaný Bauke Scholtzem (BalusC), který je distribuován jako svobodný software, jehož streamovací část byla prakticky shodná se mnou dříve napsaným servletem, avšak rozšiřoval jej o možnost požadování různých bytových rozsahů. Tento servlet byl tedy testován a shledán naprosto funkčním a po konzultaci s vedoucím práce, který s jeho použitím souhlasil, byl ve výsledné práci ponechán. Toto je tedy druhá verze servletu pro streamování multimedálního obsahu [17].

8 Závěr

Cílem této práce bylo seznámit se s technologiemi Java EE 6 a s jejich pomocí vytvořit aplikaci pro správu a řízení vnitrofiremního vzdělávání. Během práce jsem se naučil nejen mnoho moderních přístupů při návrhu a vývoji podnikových aplikací, ale získal jsem i mnoho cenných poznatků v oblasti streamování multimediálních souborů a integraci s externími službami a programy různých společností.

Práce byla zaměřena na praktickou realizaci požadavků vybraných na základě dohody se zadavatelem, především pak na vytvoření moderní, moderně vypadající aplikace, která zvýší přehlednost a použitelnost původního systému a umožní přehrávání multimediálního obsahu a integraci s kalendáři (Google, Outlook). Tyto požadavky se podařilo naplnit, a přesto, že osobně jsem si kladl za cíl větší rozsah funkčnosti aplikace (jak dokládá dokument vize), než byla původní dohoda, zadavatel nahlížel na věc více realisticky a dokázal přesněji odhadnout a požadovat rozsah výsledné aplikace.

Na začátku práce bylo potřeba seznámit se s velkým množstvím nových technologií a zvyknout si na některé „neprůhledné“ postupy frameworků, což nebylo úplně snadné, nicméně postupem času jsem si byl v těchto technologiích stále jistější.

V první části této práce se čtenář seznámí s technologiemi a některými návrhovými vzory, které byly využity při tvorbě této práce. Ve druhé části popisuji, jak vznikala celá aplikace a jak byly implementovány jednotlivé nejdůležitější požadavky zadavatele. Výslednou aplikaci by ale jistě bylo možno ještě vylepšit a návrhy na některá taková vylepšení jsou uvedena v následující kapitole (*viz* Možná vylepšení)

8.1 Možná vylepšení

Aplikace mohla být v mnoha ohledech lepší a poskytovat uživatelům více možností. Téměř všechna tato vylepšení ale z časových důvodů nebylo možné implementovat a snaha tedy byla zaměřena na zásadní funkčnost a splnění zadání, jehož hlavním cílem byla integrace s kalendáři a streamování multimediálního obsahu.

Do budoucna by aplikace měla využívat rozdělení uživatelů do rolí. V současné chvíli může uživatel dělat v aplikaci cokoliv, ať je jeho role jakákoliv. Toto vylepšení tedy spočívá v určení, kam která role má či nemá přístup. Toho se dosáhne nastavením

těchto přístupových oprávnění v souboru web.xml, který se nachází v adresáři WEB-INF. Současné nastavení zajišťuje, že se na stránky nedostane osoba, která není v databázi nebo nemá přiřazenu žádnou roli.

Dále by bylo vhodné zvýšit bezpečnost hesel uchovávaných v databázi pomocí tzv. saltu[13]. Glassfishový JDBC Realm práci se salty neumožňuje a musel by se tedy implementovat vlastní SecurityRealm.

Dále by bylo možné rozšířit přehledy podřízených pracovníků o statistiky jejich studia, přiřazování na školení apod.

Dalším vylepšením, které by bylo vhodné v budoucnu implementovat, by bylo využití Jackrabbitu jako obsahového repositáře, který by uchovával nahrané multimediální soubory.

S výše uvedeným vylepšením se pojí i to následující, a sice upload souborů pomocí GUI. V současné chvíli je potřeba soubory ukládat na disk pomocí nějakého souborového manažeru. Do budoucna by bylo dobré, aby toto umožňovala sama aplikace.

Dále by bylo pěkné, aby si uživatel mohl zvolit, do kterého Google kalendáře se mu budou události zapisovat. V současné chvíli je nastaven pouze výchozí kalendář, ale aplikace je na možnost výběru vlastního kalendáře připravena (obsahuje funkci vracející seznam všech kalendářů, které uživatel má).

[13] Náhodný řetězec přidávaný k heslu před hashováním, aby se zabránilo slovníkovým a dalším útokům (pro každé heslo by měl být jiný)

Přehled zkratek

W3C	World Wide Web Consortium
WHATWG	Web Hypertext Application Technology Working Group
POM	Project Object Model
JSF	Java Server Faces
EL	Expression Language
EJB	Enterprise Java Beans
CDI	Contexts and Dependency Injection
JPA	Java Persistence API
JTA	Java Transaction API
JPQL	Java Persistence Query Language
ORM	Object-Relational Mapping
JNDI	Java Naming and Directory Interface
SEO	Search Engine Optimization
GDC	Google Developers Console
GUI	Graphical User Interface
UI	User Interface
UX	User eXperience
POJO	Plain Old Java Object

Citovaná literatura

[1] **MacDonald, Matthew.** *HTML5: The Missing Manual.* 2. vydání. Sebastopol, CA : O'Reilly Media, Inc., 2013. ISBN 978-1-4493-6326-0.

[2] **Clark, Richard, a další.** *Beginning HTML5 and CSS 3.* místo neznámé : Apress, 2012. ISBN 978-1-4302-2875-2.

[3] Maven - What is Maven? *Apache Maven Project.* [Online] http://maven.apache.org/what-is-maven.html.

[4] **Goncalves, Antonio.** *Beginning Java™ EE 6 Platform with GlassFish™ 3.* 2. vydání. 2010. ISBN 978-1-4302-2890-5.

[5] **Geary, David a Horstmann, Cay.** *Core JavaServer™ Faces.* 3. vydání. místo neznámé : Prentice Hall, 2010. ISBN 978-0-13-701289-3.

[6] **Saleh, Hazem, Christensen, Allan Lykke a Wadia, Zubin.** *Pro JSF and HTML5.* 2. vydání. místo neznámé : Apress, 2013. ISBN 978-1-4302-5011-1.

[7] *The Java EE 6 Tutorial.* [Online] Oracle Corporation. http://docs.oracle.com/javaee/6/tutorial/doc/docinfo.html.

[8] **Keith, Mike a Schincariol, Merrick.** *Pro JPA 2.* 2. vydání. místo neznámé : Apress, 2013. ISBN 978-1-4302-4927-6.

[9] **Bien, Adam.** *Real World Java EE Patterns - Rethinking Best Practices.* 2. vydání. místo neznámé : press.adam-bien.com, 2012. ISBN 978-1-300-14931-6.

[10] OAuth 2.0 - OAuth. *OAuth Community Site.* [Online] http://oauth.net/2/.

[11] Using OAuth 2.0 to Access Google APIs. *Google Developers.* [Online] [Citace: 10. Březen 2014.] https://developers.google.com/accounts/docs/OAuth2.

[12] *RFC 5545 - Internet Calendaring and Scheduling Core Object Specification (iCalendar).* [Online] Září 2009. http://tools.ietf.org/html/rfc5545.

[13] **Harvey, Amy.** User Experience: What Is It And Why Should I Care? *Usability Geek.* [Online] 2. Červenec 2013. http://usabilitygeek.com/user-experience/.

[14] **Catlin, Hampton, Weizenbaum, Nathan a Eppstein, Chris.** *Sass: Syntactically Awesome Style Sheets.* [Online] http://sass-lang.com/.

[15] **Carr, Richard.** Gang of Four Design Patterns. *BlackWasp Software Development Home.* [Online] http://www.blackwasp.co.uk/GofPatterns.aspx.

[16] **Fortuna, Ben.** *iCal4j Wiki.* [Online] http://wiki.modularity.net.au/ical4j/index.php.

[17] **Scholtz, Bauke.** *The BalusC Code.* [Online] http://balusc.blogspot.com/.

[18] **Alur, Deepak, Crupi, John a Malks, Dan.** *Core J2EE Patterns - Best Practices and Design Strategies.* [Online] 29. Leden 2006. [Citace: 05. Únor 2014.] http://www.corej2eepatterns.com/.

[19] **Fowler, Martin, a další.** *Patterns of Enterprise Application Architecture.* 1. vydání. místo neznámé : Addison-Wesley Professional, 2002. ISBN 978-0321127426.

[20] **Freeman, Eric, a další.** *Head First Design Patterns.* místo neznámé : O'Reilly Media, 2009. ISBN 978-0-596-55656-3.

Seznam použitých obrázků

Seznam úryvků kódu

Příloha A – Podpora multimediálních formátů prohlížeči

Audio

Prohlížeč	MP3	Wav	Ogg
Internet Explorer	ANO	NE	NE
Chrome	ANO	ANO	ANO
Firefox	NE (od verze 21 ANO)	ANO	ANO
Safari	ANO	ANO	NE
Opera	NE	ANO	ANO

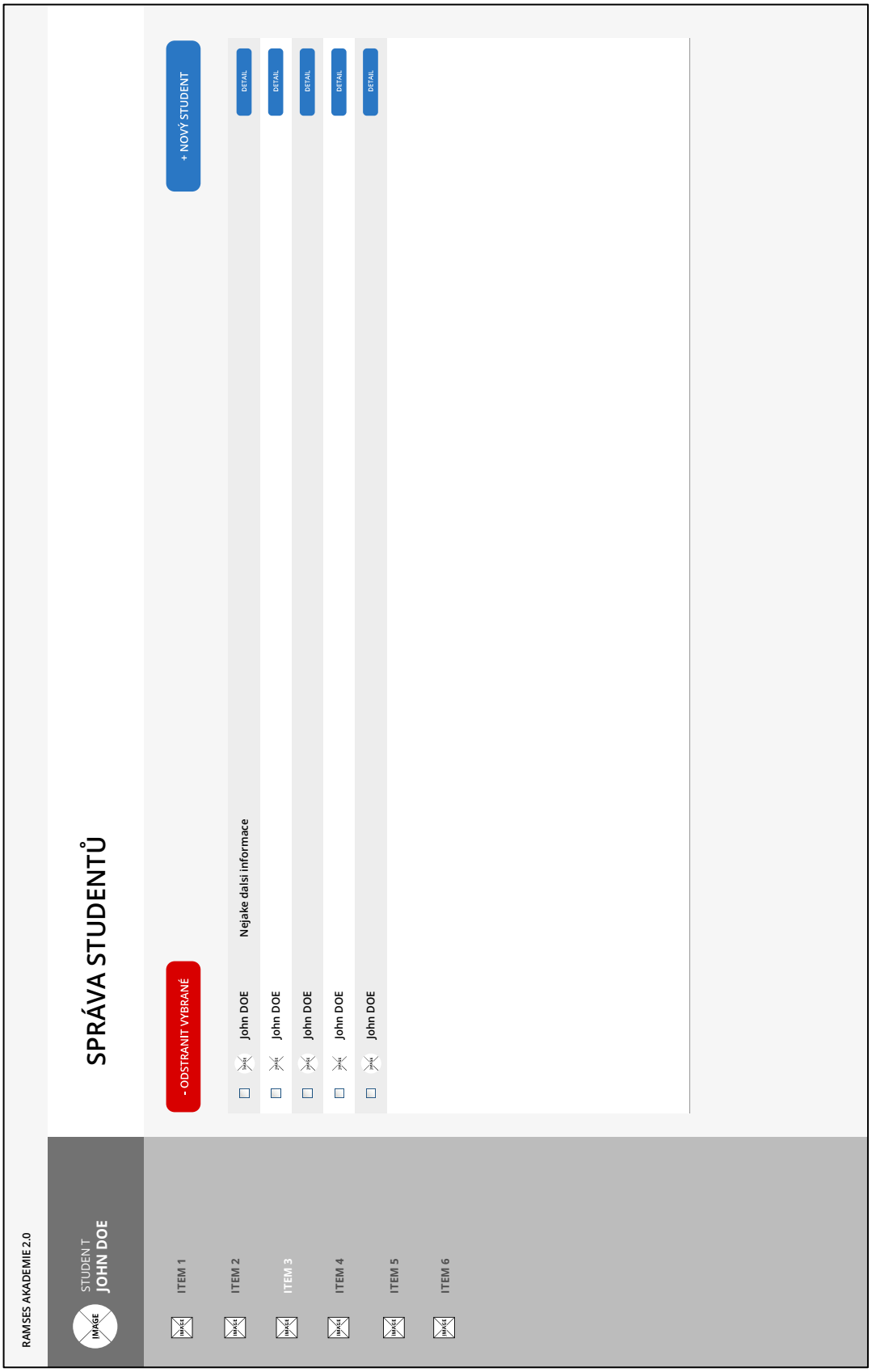
(zdroj: http://www.w3schools.com/html/html5_audio.asp)

Video

Prohlížeč	MP4	WebM	Ogg
Internet Explorer	ANO	NE	NE
Chrome	ANO	ANO	ANO
Firefox	NE (od verze 21 ANO)	ANO	ANO
Safari	ANO	NE	NE
Opera	NE	ANO	ANO

(zdroj: http://www.w3schools.com/html/html5_video.asp)

Příloha B – Wireframe aplikace



Příloha C – Uživatelský manuál

Ramses
Akademie 2.0

Uživatelský manuál

Obsah

1 Úvod

Aplikace byla vyvíjena na operačním systému Microsoft Windows 7 x64 ve vývojovém prostředí NetBeans 8.0, jehož součástí je aplikační server Glassfish 4.0. Dále byla využita databáze MySQL 5.6.11 z balíku XAMPP.

2 Instalace

2.1 Systémové požadavky

Počítač, na němž má být aplikace spuštěna musí mít nainstalováno minimálně Java Runtime Environment, Glassfish Server 4.0 a MySQL 5.6.11. Dopručeno je však vývojové prostředí NetBeans 8.0 s JDK 7 a Glassfish serverem 4.0, který se nainstaluje společně s IDE a databáze MySQL 5.6.11. Jako operační systém doporučuji Windows 7.

2.2 Nastavení MySQL databáze

MySQL databáze by měla běžet na localhostu na portu 3306 a uživatel root by měl mít nastaveno heslo „root“ (bez uvozovek). Poběží-li na jiném portu nebo bude-li nastaven jiný uživatel či heslo, je potřeba tyto údaje změnit v souboru glassfish-resources.xml (v části jdbc-connection-pool), který se nachází ve složce setup kořenového adresáře projektu.

Do databáze se následně importují data pomocí přiloženého souboru db.sql, který založí potřebnou databázi s tabulkami a naplní je základními daty. Databáze by měla být neustále spuštěna. Bude-li offline, nebude možné spustit aplikaci.

2.3 Nastavení serveru Glassfish

Nastavení serveru Glassfish je závislé na tom, jak bude aplikace na tento server poprvé zaváděna. Pro zavádění jsou dvě možnosti, buďto pomocí NetBeans nebo nahrání sestavené aplikace přímo do Glassfishe. Většina potřebných nastavení je připravena v projektu a Glassfish by je měl umět přečíst a použít v době zavádění, bohužel však tato funkce při testování nefungovala. Naproti tomu NetBeans je schopen toto nastavení do Glassfishe přenést a ruční nastavovaní je tak zredukováno na minimum.

2.3.1 Zavádění pomocí NetBeans

V tomto případě stačí nastavit Security Realm, který se bude starat o autentikaci uživatelů. Postup je následovný.

1. Spustit Glassfish server
2. Ve webovém prohlížeči otevřít adresu http://localhost:4848/, čímž se dostaneme do administrátorské konzole
3. Přihlášení do administrátorské konzole
4. V levém sloupci vybrat postupně kategorie Configurations → server-config → Security → Realms (*viz* obrázek 4-1)
5. Vytvořit nový realm s následujícím nastavením

Name	jdbc-realm
Class name	com.sun.enterprise.security.ee.auth.realm.jdbc.JDBCRealm
JASS Context	jdbcRealm
JNDI	jdbc/ramsesDatasource
User Table	student
User Name Column	login
Password Column	pass
Group Table	student_role
Group Name Column	role
Pasword Encryption Algorithm	AES
Charset	UTF-8

2.3.2 Zavádění pomocí Glassfish Admin Console

V případě zavádění aplikace pomocí Glassfishové Admin Console, je potřeba nastavit kromě výše uvedeného security realmu (viz 2.3.1) ještě zdroje, které budou využívány k přístupu k databázi a odesílání e-mailů. Postup je následovný:

1. Spustit Glassfish server
2. Ve webovém prohlížeči otevřít adresu http://localhost:4848/, čímž se dostaneme do administrátorské konzole

3. Přihlášení do administrátorské konzole
4. V levém sloupci vybrat postupně kategorie Resources → JDBC → JDBC Connection Pools
5. Založit nový pool s následujícím nastavením

Pool Name	jdbc/realmPool
Resource Type	javax.sql.ConnectionPoolDataSource
Datasource Classname	com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource

V druhém kroku pak vložit Additional Properties

URL	jdbc:mysql://localhost:3306/ramses?zeroDateTimeBehavior=convertToNull
User	Root
Password	Root
driverClass	com.mysql.jdbc.Driver

6. V levém sloupci vybrat postupně kategorie Resources → JDBC → JDBC Resources
7. Založit nový Resource s následujícím nastavením

JNDI Name	jdbc/ramsesDatasource
Pool Name	jdbc/realmPool

8. V levém sloupci vybrat postupně kategorie Resources → JavaMail Sessions
9. Založit novou JavaMail Session s následujícím nastavením

JNDI Name	mailSession
Mail Host	smtp.gmail.com
Default User	ramses.academy
Default Sender Address	ramses.academy@gmail.com
Store Protocol	imap
Store Protocol Class	com.sun.mail.imap.IMAPStore
Transport Protocol	smtp

Transport Protocol Class	com.sun.mail.smtp.SMTPTransport
+Additional Properties	
mail.smtp.port	465
mail.smtp.socketFactory.port	465
mail.smtp.socketFactory.class	javax.net.ssl.SSLSocketFactory
mail.smtp.auth	true
mail.smtp.password	CCARamsesAcademy20

Všechna tato nastavení jsou automaticky přenesena na server, pokud se aplikace zavádí pomocí NetBeans. V opačném případě je nutné je založit ručně.

2.4 Buildování aplikace

Nejjednodušším způsobem zkompilování a sestavení aplikace je otevřít ji ve vývojovém prostředí NetBeans a stisknout tlačítko Build. Vzhledem k tomu, že je tato aplikace závislá na rozhraní pluginů, které by mělo být přeložené v lokálním mavenovém repositáři, je potřeba nejprve provést sestavení tohoto rozhraní. Stav, v němž je aplikace distribuována, vyžaduje také implementace pluginů. Tyto buď mohou být odstraněny ze souboru pom.xml nebo musí být také přeloženy. Postup je tedy následovný.

1. Otevřít všechny dostupné projekty
    - calendar-api
    - calendar-google
    - calendar-outlook
    - ramses2
2. Všechny projekty postupně sestavit v tom pořadí, ve kterém jsou uvedeny výše

Sastevená aplikace ve formátu WAR archivu, se bude nacházet ve složce s projektem, v adresáři „target“.

2.5 Deployment aplikace

Jak již bylo řečeno výše, existují dvě možnosti, jak aplikaci zavést na server. Obě tyto možnosti blíže popíši v následujících kapitolách.

Aplikaci je nutné spouštět na adrese http://localhost:8080/ramses2 případně https://localhost:8181/ramses2. Toto omezení je zavedeno z důvodu, že při autorizaci aplikace pro Google kalendář se bude na tuto adresu vracet přístupový token.

2.5.1 Deployment pomocí NetBeans

Je asi nejjednodušším řešením, za předpokladu, že je nejprve provedeno sestavení všech rozhraní a pluginů, na nichž je hlavní aplikace závislá. Je-li tedy aplikace sestavena je možno ji spustit tlačítkem „Run“, které ji zavede do Glassfishe a nastaví všechny potřebné zdroje. Před spuštěním je také potřeba zkontrolovat nastavení projektu, resp. část Run, zda je server nastaven na GlassFish 4.0, verze Javy na Java EE 6 a Context Path na „/ramses2“.

2.5.2 Deployment pomocí Glassfish Admin Console

Po přihlášení do Admin Console je možno aplikaci nahrát na server (za předpokladu, že tento byl nejprve nastaven podle kapitoly 2.3.2) následujícím způsobem:

1. V levém sloupci vybrat kategorii Applications
2. Stisknout tlačítko Deploy
3. Vybrat na disku WAR archiv aplikace
4. Stisknout OK

2.6 Adresář pro multimediální soubory

Na disku je potřeba vytvořit adresář, kde se budou uchovávat multimediální soubory určené ke streamování. Tento adresář může být kdekoliv na disku a jeho umístění je potřeba zanést do nastavení v souboru glassfish-web.xml, který se nachází ve složce WEB-INF aplikace. Cestu k tomuto adresáři je potřeba zapsat do části dir (*viz* příklad níže). V tomto adresáři však musí být další adresář s názvem „output“ a v něm adresáře pojmenované kódy školení, jejichž obsah mají uchovávat (např. INT-JAVA1). Nebude-li tato adresářová struktura dodržena, nebude možné přehrávat multimédia.

```
<property name="alternatedocroot_1" value="from=/output/* dir=G:/School/_DP" />
```

3 Používání aplikace

Po spuštění je uživatel vyzván k přihlášení. Aplikace, tak jak je distribuována, má v databázi záznam jen pro jediného uživatele s uživatelským jménem „admin“ a heslem „admin“. Použitím tohoto jména a hesla by měl být umožněn vstup do aplikace. Vyskytne-li se při přihlášení problém, doporučuji zkontrolovat přihlašovací údaje, dále zda je spuštěna databáze a nakonec důkladně zkontrolovat nastavení, která byla ručně zadávána do Glassfishe.

3.1 Uživatelský účet

Po přihlášení je v možné vidět v levém sloupci nahoře jméno právě přihlášeného uživatel. Po kliknutí na toto jméno je uživatel přesměrován na nastavení svého profilu, kde si může nastavit email, na který mu budou chodit iCal soubory, a kde se může přihlásit se do Google kalendáře.

3.2 Přihlášení na realizaci

Je-li vypsána nějaká realizace, jejíž datum je v budoucnosti, je možné se na ní na stránce realizací přihlásit pomocí tlačítko u této realizace. Získal-li přihlášený uživatel obnovovací token pro Google kalendář, bude mu tato realizace přidána do kalendáře. Pakliže má vyplněn e-mail, na nějž chce zasílat iCal soubory, bude mu poslán i tento soubor. Při odhlášení bude událost z Google kalendáře vymazána a do e-mailu přijde iCal soubor rušící událost v Outlook kalendáři.

Po přihlášení na realizaci se tato zobrazí v dashboardu přihlášeného uživatele s odkazem na multimediální obsah školení, v případě, že toto školení nějaký má.

4 Obrazová příloha

4-1 Cesta k nastavení Security Realm