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

Bakalářská práce

Prezenční vrstva EEG/ERP Portálu

Plzeň, 2013 Jakub Rinkes

Prohlášení

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

V Plzni dne 8.5.2013,

Jakub Rinkes

Poděkování

Touto cestou bych rád poděkoval všem, kteří mi pomáhali a podporovali mě během vytváření této práce.

Abstract

The EEG/ERP Portal is a web application for long term storage and management of electrophysiological experiments. It was implemented using mainly the following technologies - Spring MVC and Java Server Pages. These technologies are already outdated and their next extension is complicated. To solve this problem an alternative technology, Apache Wicket framework, was found after analysis of current technologies. This framework, which ensures simple extension and functional improvement of the portal, was successfully implemented.

Obsah

1.Úvod

Po roce vývoje se aplikace stává zastaralá, proto je její údržba náročnější a dochází ke složitým úpravám. Dokonce se stává, že použitý framework přestane být vyvíjen. Za posledních pár let takto vzniklo a zaniklo velké množství frameworků, málokterý se může chlubit životností delší než pět let.

Tématem této práce je výše uvedená problematika. EEG/ERP Portál (dále jen Portál) je webová aplikace, která je vyvíjena již několik let s použitím Spring MVC a Java Server Pages, které jsou dnes již zastaralé.

V teoretické části této práce dochází k seznámení s aktuálním stavem Portálu a analýze dnešní nabídky frameworků, kterými lze nahradit použitý Spring MVC a Java Server Pages. Od nového frameworku se očekává jednodušší údržba a rozšiřování, spolupráce s dnešními technologiemi a delší životnost.

V praktické části dochází k seznámení s nově zvoleným frameworkem, který bude implementován. Zvolen byl Apache Wicket, bude popsána jeho základní funkčnost. Při samotné implementaci bude docházet k přepsání webových stránek s použitím frameworku Wicket se snahou o zachování stejné funkčnosti nebo jejího vylepšení.

2. EEG/ERP Portál

Portál je webová aplikace výzkumné skupiny Katedry informatiky a výpočetní techniky na Západočeské univerzitě v Plzni. Výzkumná skupina se zabývá experimenty v oboru neuroinformatiky. Pomocí definovaných scénářů jsou osobám vystaveným specifickým vlivům měřeny evokované potenciály. Při těchto měřeních vzniká velké množství dat, které je nutné pro účely výzkumu správně ukládat k pozdější analýze a zpracování. Samotné měření provádí větší množství osob, které specifikace scénářů sdružují do skupin, a proto je nutné umožnit hromadný přístup k systému pro ukládání, analýzy a zpracovaní dat. Z tohoto důvodu byl Portál vytvořen a je používán.

2.1. Funkce

Aktuální funkce Portálu jsou [1]:

- Vytváření a správa uživatelských účtů
- Vytváření výzkumných skupin a jejich organizaci
- Ukládání, analýza a zpracování dat a metadat experimentů
- Sdílení dat a metadat mezi uživateli a výzkumnými skupinami
- Přidávání článků a jejich komentovaní pro uživatele a skupiny
- Vyhledávání v EEG databázi
- Historie pro administrátora skupiny nebo systému
- Přihlášení do Portálu pomocí sociálních sítí Facebook a LinkedIn

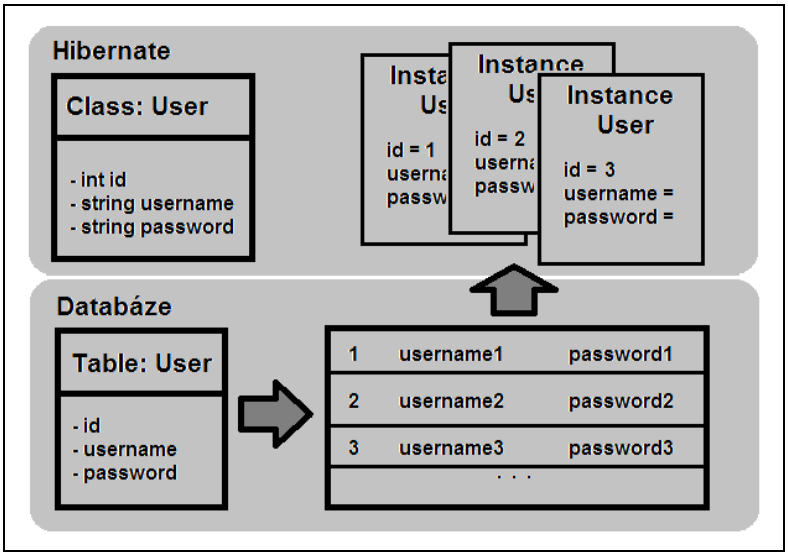
2.2. Architektura

Portál je webová aplikace s třívrstvou architekturou napsaná v jazyce Java Enterprise Edition. Vícevrstvá architektura slouží k rozdělení implementace aplikace do úrovní, jejichž oddělení umožňuje snadnější rozšíření a modifikaci bez ovlivnění ostatních úrovní.

2.2.1. Datová vrstva

Aplikační data jsou uložena v databázovém systému Oracle verze 11g. Přístup k datům v databázi je zajištěn nástrojem Hibernate [2] pro objektově-relační mapování (ORM).

ORM zajišťuje mapování dat z databáze do Javových objektů označovaných jako POJO (zkratka z anglického Plain Old Java Object), se kterými aplikace už může pracovat (obrázek č. 1).



Obrázek 1: Hibernate - princip mapování řádků tabulek do tříd.

2.2.2. Aplikační vrstva

Aplikační vrstva je implementována Spring frameworkem [3], který obsahuje volitelných 12 modulů, které usnadňují vývoj aplikací v Javě.

- Základní modul kontejner (z anglického Inversion of Control container) se stará o vytvoření objektů aplikace a jejich provázání podle závislostí mezi objekty. Patří mezi nejdůležitější moduly ve Springu.

- Modul Security byl použit pro zabezpečený přístup k datům.

- Modul Aspect-oriented programming (zkratka AOP) pracuje s aspekty a umožňuje použití anotací v aplikaci.

- Modul Social poskytuje předpřipravené napojení na internetové sociální sítě na internetu. Portál využívá pro přihlášení nebo registraci uživatelů sociální sítě Facebook a LinkedIn. LinkedIn z důvodu toho, že jiné výzkumné skupiny využívají tuto síť. Pomocí Portálu je možné publikovat články na sociální síť a získávat informace o skupinách.

- Modul MVC (Model-View-Controller) ve webové aplikaci zajišťuje vrstvu pro zobrazení a interakci mezi uživatelem a aplikací. Model obsahuje data pro zobrazení uživateli. Uživatel provede interakci, controller tuto akci zpracuje a předá modelu, který zpracuje data a předá je k zobrazení.

2.2.3. Zobrazovací vrstva

Vrstva zobrazení generuje výstup do HTML jazyka. O tento výstup se stará Spring MVC, který předzpracuje data a stará se o interakci s uživatelem a Java Server Page (zkratka JSP)[4], který obsahuje šablony, které definují výstup. JSP šablona obsahuje kód v jazyce HTML a určité značky které umožňují dynamické generování výstupu dle dat. Kromě šablon se používají ještě soubory s příponou „.tag“, které určují akce a funkcionalitu šablon pro zobrazení.

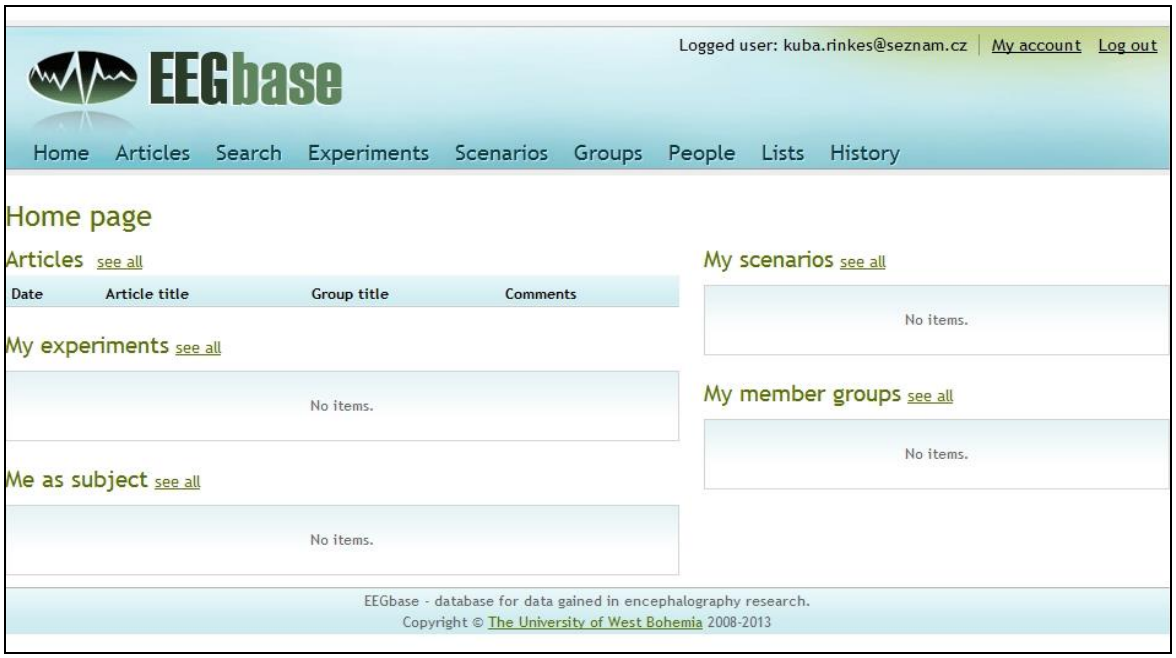
Touto akcí je například: podmínka pro zobrazení odkazu na jinou stránku, pokud je v datech správná hodnota.

2.3. Ostatní technologie

EEG/ERP Portál používá různé technologie a knihovny, které mu poskytují dodatečné vlastnosti. Pro práci s projektem a jeho překladem byla použita knihovna Maven verze 3. Webové služby knihovny Apache CXF poskytují rozhraní ostatním aplikacím, se kterými je portál propojen. Mezi tyto aplikace patří generátor struktur v sémantickém webu, rezervační systém laboratoře a speciální verze portálu pro mobilní zařízení.

2.4. Aktuální stav Portálu

Po hlubším prostudování zdrojových kódů bylo možné zhodnotit stav Portálu. Ze zdrojových kódů je vidět, že návrh architektury a implementace základní verze portálu byly dobře navrženy a provedeny. Postupem času byly do portálu dodělávány funkčnosti, které byly zadány jako seminární práce některých předmětů nebo jako kvalifikační práce. Některé tyto dodělávané funkcionality způsobily nepořádek v navržené architektuře. Portál (vzhled na obrázku č.2) je stabilní a je využíván a rozšiřován pro potřeby výzkumu.



Obrázek 2 : Ukázka webového rozhraní EEG/ERP Portálu.

3. Technologie pro webové rozhraní

Pro platformu jazyka Java existuje dnes mnoho technologií a frameworků, prostřednictvím kterých lze vytvořit webové rozhraní. Historicky první byly Java Applety. Po nich už přišly servlety a později Java Server Pages (zkratka JSP). Pro Portál, který je více webovou aplikací než jen webovou stránkou, hledáme framework, který nám usnadní práci při vývoji a bude se snáze modifikovat a rozšiřovat.

Aktuálně používané JSP jsou sice stále používaným nástrojem (pozn. společnost Google má pomocí JSP vytvořen jejich základní Google App Engine, na kterém staví všechny své aplikace), ale jejich modifikace a rozšiřováni je pracné a obtížné. Práce s AJAX[1] (Asynchroním JavaScriptem a XML) a jeho integrace je složitější.

3.1. Porovnání technologií

Frameworky se dělí do dvou větších skupin – component-based a request-based.

- Při použití **component-base frameworků** se pracuje s připravenými nebo vlastními komponentami. S jejich pomocí se vytváří webové stránky podobně, jako se vytvářela grafická rozhraní standartní desktopové aplikace.

- **Request-based frameworky** většinou pracují s požadavky. Požadavek je zpracován serverem, který vygeneruje výstupní HTML kód a zobrazí jej uživateli.

[1] AJAX je obecné označení pro technologie vývoje interaktivních webových aplikací, které mění obsah svých stránek bez nutnosti jejich opětovnému načtení.

3.1.1. Component-based frameworky

- Google Web Toolkit (zkratka GWT) [5]

GWT je framework vyvíjený společností Google. Využívá jazyk Java a komponent, ale jeho zpracování a generování webového rozhraní je unikátní. Používá svůj vlastní překladač pro generování kódu z jazyka Java do jazyka JavaScript[2], který už webový prohlížeč umí zobrazit. Komunikaci se serverem řeší voláním vzdálených procedur (z anglického Remote Call Procedure, zkratka RCP).

GWT ulehčuje práci s technologií AJAX a tím umožnuje vytvářet kvalitní dynamické webové rozhraní. Jeho předností je rychlost, s jakou dovoluje vyvíjet aplikace. Problém pak nastává v samotné implementaci, kde dochází ke kombinaci prezentace a aplikačního kódu, což znesnadňuje práci s CSS styly, a ve srovnání s ostatními frameworky vzniká znatelně větší množství zdrojového kódu. Aplikace v GWT se nechová jako aplikace, spíše jako skupina modulů, obvykle je jedna webová stránka jako jeden modul. Navíc překlad kódu Javy do JavaScriptu je poměrně zdlouhavý.

- Apache Wicket [6]

Autory Wicketu jsou Jonathan Locke a Miko Matsumura. První verze Wicketu byla uvolněna v roce 2005. Po dvou letech byl přijat pod organizaci Apache Software Foundation. Za jeho vývojem nestojí softwarový gigant, ale menší skupina vývojářů z celého světa. Tvorba webového rozhraní je velice podobná vytváření grafického rozhraní knihovnou Java Swing.

Stránka se skládá z komponent. Vzhled webového výstupu komponenty je popsán v HTML souboru, který má stejné pojmenování a nachází se ve stejném balíčku. Pokud je stránka vyžádána, je z objektu stránky a všech HTML souborů použitých komponent

[2] Java Script je interpretovaný programovací jazyk pro WWW stránky, často vkládaný přímo do HTML kódu stránky. Jsou jím obvykle ovládány různé interaktivní prvky GUI (tlačítka, textová políčka) nebo tvořeny animace a efekty obrázků.

vygenerován HTML kód a ten je následně zobrazen uživateli. Wicket využívá jQuery[3] pro práci s technologií AJAX. K vývoji stačí znát jazyk Java a mít základy HTML.

Jeho největší předností je využití cache, jednoduchý komponentový model pro vývoj. Oddělený HTML kód umožnuje jednodušší návrh a úpravu vzhledu a použití stylů, ale i odolnost webového rozhraní, aktivní vývoj a použití nových technologií. Wicket spolupracuje se Spring a Hibernate frameworkem bez větších implementačních nároků.

3.1.2. Request-based framework

- Grails [7]

Grails je zkrácený název pro technologii Groovy on Rails. Je to request-based framework, který využívá programovací jazyk Groovy [8], který vychází z jazyka Java. Framework ulehčuje práci vývojářů tím, že je úplně odstiňuje od jakékoliv konfigurace a nastavení pro webové rozhraní, pokud se spokojí se základním nastavením, které úplně postačuje. Využívá AJAX a jeho snadnou modifikaci pro bohatší webové rozhraní. Jeho model je hodně podobný architektuře Spring MVC a šablonám JSP. Grails má místo Java Server Page šablony Groovy Server Page. Groovy je spíše skriptovací jazyk, ale i tak dovoluje napojení na běžně užívané frameworky, jako jsou Spring a Hibernate.

Jeho největší přednosti jsou rychlost vývoje, velká podpora pluginů a knihoven s hotovými řešeními, snadné vytváření vlastních tagů pro šablony a jednoduché testování. Implementace v tomto frameworku používá jazyk Groovy.

[3] jQuery je JavaScriptová knihovna, která je zaměřena na interakci mezi JavaScriptem a HTML.

3.2. Výběr

Webové stránky rozdělíme podle typu používání na webové prezentace a webové aplikace. Za webové prezentace lze označit stránky, které mají určitým způsobem někoho nebo něco představit. Obvyklé jsou firemní nebo různé reklamní stránky. Webová aplikace je webová stránka s hlavním cílem umožnit uživatelům pracovat s informacemi, například e-shopy, knihovny, studijní agendy, internetové bankovnictví. Po webové aplikaci se obvykle požaduje chování jako po běžné desktopové aplikaci, není důležitý design ale spíš uživatelsky příjemné prostředí.

GWT framework je nástroj pro webové prezentace, protože umožnuje vytvářet dynamické stránky, ale pro použití v Portálu se nehodí. GWT patří mezi client-side[4] frameworky, což zkomplikuje napojení na stávající datovou vrstvu, jelikož pro komunikaci s datovou vrstvou bude nutné dopsat mnoho napojení pomocí RPC.

Grails spolu s Wicketem patří mezi server-side[5] frameworky a hodí se více na webové aplikace, ale Grails se nehodí pro použití v Portálu. Propojit Grails se stávající datovou vrstvou není nemožné, přestože jsou Grails napsané v jazyce Groovy. Takovéto propojení není doporučené, protože Grails používá pro práci s Hibernatem vlastní nástroj GROM a obvykle i datová vrstva bývá vytvořena pomocí Grails a jeho nástrojů. Použití Grails v případě Portálu by znamenalo kompletní přepis datové vrstvy, a tím větší rozsah implementace a pracnosti, pokud by měl být Grails implementován správně.

Vybrán bude Wicket. Při jeho použití se nahradí stávající vrstva zobrazení, nebude nutné měnit velkou část architektury, přímá spolupráce se Spring a Hibernatem umožní napojení na stávající datovou vrstvu bez větších zásahů. Odpadne i nutnost učit se jiný programovací jazyk nebo pracovat s novou technologií. Wicket pracuje pouze s Javou a HTML, což umožní rychle naučit ostatní vývojáře používat tento framework.

[4] Client-side Framework – zdrojový kód je prováděn v prohlížeči webových stránek na straně klienta.
[5] Server-side Framework – zdrojový kód je prováděn na straně serveru a prohlížeč dostane stránku k zobrazení.

4. Apache Wicket

„ Jednoduchost, Znovupoužitelnost, Bezpečnost, Efektivnost “

je filozofie, se kterou byl a je Wicket [6][9] vyvíjen. V roce 2004 byly desítky frameworků, umožňující programování webového rozhraní v jazyce Java. V roce 2005 se objevila první verze, která byla uvolněna k použití. V roce 2007 byl zařazen do Apache Software Fondation a byla mu zajištěna aktivní komunita vývojářů, aby projekt nezanikl. Dnes je Wicket aktuálně ve verzi 6.7.0+ a přibližně každé 4 týdny vychází nové verze s opravami nebo rozšířeními.

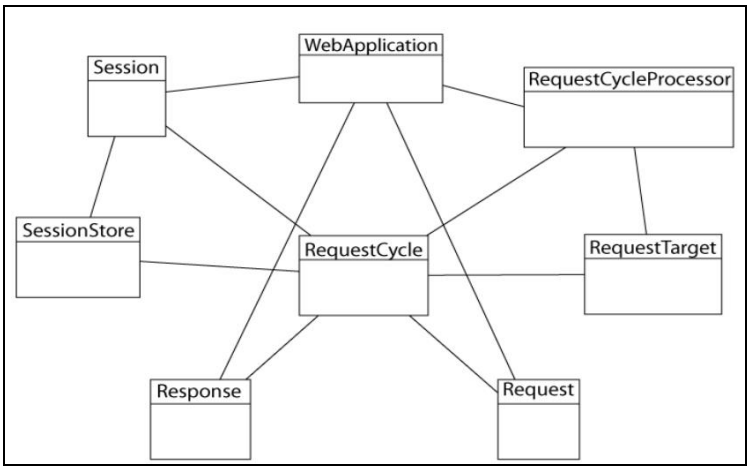
Hlavní myšlenkou je umožnit vývojářům naplno využít programovací jazyk Java místo spolupráce více jiných technologií a odstínit je od složitých procesů webové aplikace. Vývojář pracuje v Javě nad komponentami a Wicket zajistí práci s URL, relací a požadavky metodami GET nebo POST.

Wicket zjednodušuje nastavení webové aplikace. Ta je schopná běžet bez jediného dodatečného souboru s nastavením kromě základního popisu webové aplikace v souboru web.xml. Oddělení části s HTML od části v Javě usnadňuje vývoj a dovoluje rozdělení práce pro designéra webu a programátora.

Wicket už v základu umožňuje velice pokročilé funkce. Má vyřešený problém zpětného tlačítka v prohlížeči webových stránek stejně jako problém více oken nebo záložek. Nabízí jednoduché a flexibilní validace formulářů, automatickou konverzi dat z formuláře do datových typů programovacího jazyka nebo generovaní HTML validních stránek a možnost, jak ovlivnit programově HTML značky výstupu (například dodaní CSS tříd).

4.1. Jádro aplikace

Wicket jádro je skupina tříd, které díky vzájemné spolupráci obsluhují požadavky od uživatele a vrací mu odpovědi. Těchto základních tříd je 8 [10]: WicketApplication, Session, SessionStore, RequestCycle, RequestCycleProcessor, Request, RequestTarget a Response.



Obrázek 3: Graf tříd představující jádro Wicket frameworku.
[9] DASHORST, Martijn a Eelco HILLENIUS. Wicket in action.

RequestCycle, RequestCycleProcessor, Request, RequestTarget a Response jsou třídy obsluhující požadavky přicházející od uživatele a po odbavení požadavku vrací odpověď uživateli. Jejich vnitřní proces je možné ovlivnit podle potřeb.

Session a SessionStore jsou třídy odpovědné za uchování relace. Instance Session se stará o relaci mezi uživatelem a serverem po celou dobu komunikace. Díky možnosti vlastní implementace je možné rozšiřovat vlastnosti relace a její chování. SessionStore určuje, jaká data se budou v rámci session uchovávat. K instanci pro aktuální relaci se dostaneme přes statickou metodu *get()* v té třídě, která tuto instanci vytvořila.

WicketAplication je základní třída, která je potřeba pro chod aplikace. Každá běžící webová aplikace Wicketu má právě jednu instanci této třídy. Tato třída je odpovědná za to, jak aplikace funguje. Určuje nastavení pro zdroje, debug režim, zabezpečení, práci s výjimkami, vzhled a dodatečné vlastnosti výstupního HTML kódu, třídu použitou pro

relaci, domovskou stránku a propojení s ostatními technologiemi jako je Spring a Hibernate.

Aby nebylo nutné tvořit projekt pro Wicket aplikaci ručně, vytvořili autoři Wicketu pro ostatní uživatele generátor projektu pod nástrojem Maven. Na oficiálních stránkách je sekce Quickstart, kde je připraven příkaz pro Mavenu, který vygeneruje základní Wicket projekt obsahující HomePage.class, WicketApplication.class, web.xml. Tento projekt je po přeložení a umístění do webového serveru spustitelný.

4.2. Komponenty

Wicket obsahuje velké množství komponent pro různá použití, ale základní stavební komponenty pro tvorbu webového rozhraní mají stejnou architekturu. Každá z komponent obsahuje svůj model pro uchování dat a stavu, logiku pro obsluhu událostí, a pokud komponenta má generovat výstup pro uživatele, tak i popis jak takový výstup připravit.

4.2.1. Model

Každá komponenta obsahuje model, ten buď uchovává stav komponenty, nebo data, která komponenta zobrazuje. Model [11] je implementace rozhraní IModel:

```
public interface IModel {
        public Object getObject();
        public void setObject(final Object object);
}
```

Wicket obsahuje sadu připravených modelu, které se liší v chování.

- Model – model určený k uchování dat. Nemá žádné specifické vlastnosti, a pokud si komponenta řekne o objekt z modelu, je jí dán objekt, který je uložen v modelu.
- Property model – přistupuje k datům za běhu aplikace a objekt, který vrací, je vždy ten aktuální, který je do modelu vložen. Tímto chováním se řadí mezi dynamické modely. Do konstruktoru je kromě objektu, který do něj vkládáme,

vložen i tzv. property klíč. Ten určuje, který atribut objektu má model vracet jako data, není nutné hlídat *null* hodnotu.

Příklad: *Třída Person obsahuje 2 atributy – name, password. Při použití* ***new PropertyModel(new Person(), "name")*** *bude model vracet vždy jen aktuální atribut name.*

- CompoundProperty model – rozšířený PropertyModel. Pokud rodičovská komponenta obsahuje tento model a komponenta nemá přímo určený svůj model, vyhledá si v rodičovské třídě tento model a identifikátorem komponenty si řekne CompoundProperty modelu o atribut.

Příklad: *Opět použijeme třídu Person, ale její instanci vložíme do CompoundPropertyModelu a ten předáme rodičovské komponentě. Do rodičovské komponenty přidáme komponentu pro výpis řetězce Label a její identifikátor použijeme přesně stejný jako je název atributu tedy "name". Ve chvíli, kdy bude komponenta potřebovat data z modelu pro zobrazení a zjistí, že sama žádný model neobsahuje, řekne si o CompoudPropertyModel rodičovské komponenty. A z něj se pokusí identifikátorem získat atribut "name".*

- LoadableDetachable model – model je abstraktní implementace. Při vytváření jeho instance je potřeba doplnit tělo metody *load()*. Pokud komponenta požaduje data, model získá data voláním metody *load()* a předá je komponentě.

Příklad: vytvoření modelu a doplnění metody pro získání dat.

```
LoadableDetachableModel personListModel = new LoadableDetachableModel() {
        protected Object load() {
                return getPersonDao().findPersons();
         }
};
```

Po předání dat je model pomocí metody *detach()* automaticky zahodí. Díky těmto vlastnostem model neskladuje data a nezabírá paměť aplikace. Je na uvážení programátorů, kde potřebují data zobrazit a můžou je z důvodu paměťové úspory zahodit. Typickým příkladem jsou seznamy a tabulky.

- Resource modely – usnadňují práci s lokalizací. Tyto modely pracují s lokalizátorem. Wicket si při spuštění aplikace načte všechny soubory pro lokalizaci, které najde, nebo které mu určíme a jejich mapu si drží v paměti. ResourceModel nám zařídí, abychom dostali správný řetězec na základě klíče, pod kterým je řetězec uložen. ResourceModely reagují na nastavení jazykového prostředí (Locale) v aktuální relaci.

Příklad: WicketApplication.properties *soubor obsahuje: label.helloworld=Ahoj Světe, pokud vytvoříme* ***new ResourceModel("label.helloworld")****, tak model bude obsahovat řetězec „Ahoj Světe".*

Protože občas je potřeba kombinovat vlastnosti modelů, což by vyžadovalo napsání nového modelu, došlo k rozšíření modelů. Modely je možné vkládat do sebe a tím se budou kombinovat vlastnosti modelů. Velice často se kombinuje CompoundPropertyModel s LoadableDetachableModelem, aby se dosáhlo toho, že data se před použitím načtou aktuální a nebude potřeba v kódu určovat komponentám jejich modely přímo.

4.2.2. Komponenty

Webové stránky se tvoří z jednotlivých komponent nebo seskupením komponent. Proto je při vytváření stránek nutné se zamyslet, z jakých komponent stránku vytvoříme a do jakých struktur budeme komponenty skládat. Komponenty je možné do sebe libovolně vkládat, ale je třeba dát pozor, aby strom komponent byl správně popsán i HTML souborem, který k stránce musí také vzniknout.

Wicket obsahuje celou sadu základních komponent, navíc ještě mnoho komponent specifických či užitečných. Komponenty se dělí do 4 skupin:

- výstupní – výpis řetězce, víceřádkový výpis, atd.
- odkazy – odkazy na stránky, externí odkazy, atd.
- rozvržení – stránka, panel, fragment, atd.
- formuláře – textové pole, tlačítko, zaškrtávací políčko, atd.

Dále je možné vytvářet vlastní komponenty (využít dědičnost a překrytí). Aby se kód nemusel opakovat, je dobré si napsat vlastní specifickou komponentu a ušetřit délku kódu i čas programátora.

4.2.3. HTML výstup – Markup

Wicket při tvorbě stránek kromě komponent potřebuje i HTML strukturu, do které se bude komponenty snažit dosadit. Tyto soubory nebo kusy HTML kódu, které píšeme pro Wicket komponenty, budeme dále nazývat Markup [12].

Některé komponenty si generují výstupní HTML kód samy, pro některé je nutné daný kód dopsat. Mezi takové komponenty patří stránka, panel, fragment a další. Pokud se vytváří stránka nebo panel je nutné, aby byl vytvořen i stejně pojmenovaný HTML soubor.

Příklad vytvořené stránky a markupu:

MyPage.java

```
public class MyPage extends WebPage {
        public MyPage() {
                ... komponenty…
        }
}
```

MyPage.html

```
<html>
        <body>
                ... struktura markupu ...
        </body>
</html>
```

Proces přidání komponenty obsahuje kromě samotného přidání komponenty v Javě také přidání komponenty do Markupu.

Příklad:

MyPage.java

```
public class MyPage extends WebPage {
        public MyPage() {
                add(new Label("label", "Text"));
        }
}
```

MyPage.html

```
<html>
        <body>
                <span wicket:id="label"></span>
        </body>
</html>
```

Na příkladu je vidět přidání komponenty Label do stránky. Komponentě Label byl předán v konstruktoru identifikátor "label". Pomocí tohoto identifikátoru je nutné komponentu označit v markupu, aby Wicket věděl, kde bude daná komponenta ve struktuře umístěna.

Toto provázání komponenty s její značkou v markupu se provádí pomocí atributu **wicket:id**. V příkladu dostala značka <*span*> atribut **wicket:id** a mezi začáteční a ukončovací značku bude vložen výstup komponenty. Výstupní HTML kód bude vypadat takto:

```
<html>
        <body>
                <span wicket:id="label">Text</span>
        </body>
</html>
```

Protože se komponenty dají zanořovat do sebe, musí toto zanoření být i v markupu. Příklad zanoření dvou komponent Label do sebe:

Příklad:

MyPage.java

```
public class MyPage extends WebPage {
        public MyPage() {
                Label label1 = new Label("label1" "Text1");
                label1.add(new Label("label2", "Text2");
                add(label1);
        }
}
```

MyPage.html

```
<html>
        <body>
                <span wicket:id=”label”>
                        <span wicket:id=”label2”></span>
                </span>
        </body>
</html>
```

V příkladu vidíme, že zanoření komponent v Javě je popsáno i v markupu zanořením značky <*span*> s identifikátorem pro vnitřní label2 mezi párové značky pro vnější label1.

Wicket obsahuje pro tvorbu markupu i další značky nebo atributy [12]:

Wicket atributy:

wicket:id, wicket:message, wicket:enclosure, wicket:for, wicket:unknown, wicket:scope

Wicket značky:

wicket:link, wicket:panel, wicket:fragment, wicket:border, wicket:body, wicket:extend, wicket:child, wicket:message, wicket:remove, wicket:head, wicket:enclosure, wicket:container

Přesná funkce značek a použití je k nalezení v dokumentaci.

4.3. Formuláře a validace

4.3.1. Wicket a formuláře

V případě webových aplikací slouží pro zadávání vstupních dat do aplikace formuláře. Wicket obsahuje základní komponentu Form [9] neboli formulář. Tato komponenta sdružuje skupinu komponent pro zadávání vstupních dat. Komponenty pro vstupní data jsou ekvivalentem pro komponenty v základním HTML jazyce. Provázání vstupních komponent v markupu je potom stejné jako u běžných komponent provázaných pomocí **wicket:id**.

Příklad: *Jednoduchá webová stránka s formulářem pro přihlášení.*

LoginPage.java

```
public class LoginPage extends WebPage  {

        public LoginPage() {
                Form<Login> loginForm = new Form<Login>("form");
                loginForm.setModel(new CompoundPropertyModel(new Login());
                loginForm.add(new TextField<String>("username").setRequired(true));
                loginForm.add(new PasswordTextField("password").setRequired(true));
                add(loginForm);
        }
}
```

LoginPage.html

```
<html>
        <body>
                <form wicket:id="form">
                        <input type="text" wicket:id="username"/>
                        <input type="password" wicket:id="password"/>
                </form>
        </body>
</html>
```

V třídě LoginPage.java je ukázka, jak vznikne formulář a přidá se na stránku. V konstruktoru nejdřív vznikne objekt Form, který v sobě bude seskupovat komponenty pro formulář. Metodou setModel() mu předáme CompoundPropertyModel, který obsahuje prázdnou instanci objektu Login. Ten obsahuje atributy username pro uživatelské jméno a password pro heslo. Do tohoto objektu nám formulář uloží data ze vstupu. Další dva příkazy přidají do formuláře dvě vstupní pole. První je obyčejné textové pole a jeho obsahem bude uživatelské jméno. Druhé pole je speciální komponenta pro zadávání hesel. Jeho obsah je schován a ve vstupním poli se objevují jen hvězdičky. Poslední příkaz přidá hotový formulář do stránky. HTML markup pro tuto stránku je vidět pod samotnou třídou, obsahuje formulář s identifikátorem Form a v něm zanořeny dva vstupy s identifikátory username a password.

4.3.2. Zpracování dat formulářem

Hlavní funkcí formuláře je získat vstupní data z komponent. Této akci se obecně říká odeslání dat (z anglického submit). Wicket usnadňuje práci s formuláři při získávání dat ze vstupních komponent tím, že programátor zařídí pomocí modelů, kam mají vstupní komponenty vkládat data. Data z komponent se mohou vkládat do modelu formuláře nebo do modelu komponent. Při využití CompoundPropertyModelu ve formuláři není nutné určovat komponentám, kam mají data při odeslání vkládat. Je nutné dodržet, aby identifikátor pro komponentu byl stejný jako název atributu objektu v modelu.

Wicket při odeslání formuláře přečte vstupní hodnoty z komponent, provede konverzi na správný datový typ, který určuje generické nastavení komponent, a vloží načtená data do modelu komponent nebo do modelu formuláře, pokud je tak určeno. Po odeslání formuláře nastane stav, kdy data ze vstupu jsou naplněny v objektu v modelech komponent nebo v modelu formuláře, a můžeme s nimi dál pracovat.

Pro odeslání formuláře slouží specifické komponenty. Jsou tu SubmitLink, SubmitButton nebo AjaxSubmitLink a AjaxButton. Tyto komponenty obsahují metodu *onSubmit(),* kterou při překrytí můžeme kontrolovat, co se stane po zpracování dat formulářem (například uložení dat do databáze). Formulář obsahuje také metodu *onSubmit()*, kterou lze překrýt a tím kontrolovat, co se stane po zpracování dat.

4.3.3. Validace dat ve formuláři

Při práci s formulářem je požadováno, aby vstupní data byla automaticky kontrolována na správný datový typ, formát nebo splňovala určité podmínky. Při zpracování provádí formulář validaci vstupních dat validátory, které jsou ke komponentám přiřazeny.

Validátor je třída, která implementuje rozhraní IValidator:

```
public interface IValidator<T> extends IClusterable {
        void validate(IValidatable<T> validatable);
}
```

Wicket obsahuje mnoho předpřipravených validátorů pro různé datové typy, případně není složité požadovaný validátor naprogramovat. Pokud je požadováno, aby vstupní pole nebylo prázdné, byla tato vlastnost implementována přímo do formulářových komponent a stačí nastavit vlastnost „required“ na hodnotu *true* metodou *setRequired()*.

Komponenta obsahuje svůj seznam přidružených validátorů. Validátor se komponentě přidá metodou *add(IValidator<? super T> ...)*. Formulář během zpracování vstupů použije daný validátor a vyhodnotí, zda vstupní data jsou správná nebo ne. Pokud vyhodnotí, že nejsou správná, vznikne zpráva, kterou se bude snažit formulář předat uživateli. K tomuto účelu vznikl propracovaný systém zpráv, které Wicket umí zobrazovat. Nazývají se zprávy zpětné vazby (z anglického Feedback message).

4.3.4. Zprávy zpětné vazby

WicketApplication, Session a každá komponenta obsahuje metody (*info(), warn(), error(), atd*) pro vytvoření zprávy zpětné vazby [9]. Tyto zprávy slouží k zobrazení informací o zpracování formuláře nebo pro zobrazení informací uživateli. Pro jejich zobrazení na stránce je nutné, aby stránka obsahovala komponentu FeedbackMessagePanel. Tato komponenta sbírá zprávy z úložiště v Session, kde se všechny ukládají. Proces sběru a zobrazení je automatický a při každém načtení stránky probíhá kontrola, zda se mají některé zprávy zobrazit. Každý panel pro zobrazení může obsahovat filtry, které budou třídit zprávy. Pokud Wicket má zprávu, kterou by chtěl zobrazit, ale nemá k tomu na stránce příslušný panel, objeví se v konzoli výpis, že existuje určitá zpráva a nemůže ji zobrazit uživateli.

4.4. AJAX technologie

Technologie AJAX [9] umožňuje prohlížeči provést dotaz na server a z odpovědi aktualizovat stránku bez kompletního znovunačtení. Nedochází k překreslení celé stránky, a to umožňuje jednodušší ovládání aplikace.

Wicket má vlastní integraci technologie AJAX, která je aktuálně implementovaná jazykem jQuery. Wicket díky této integraci zvládá [9]:

- Posílání zpráv mezi klientským prohlížečem a serverem.
- Překreslování, skrývání a úpravu komponent.
- Spuštění JavaScript kódu poslaného v AJAX odpovědi.
- Dynamické přidávání JavaScript a CSS stylu do webových stránek.
- Opakovací nebo časované události.
- Debug režim umožňuje sledovat a ladit JavaScript kódu.

Wicket obsahuje připravené komponenty pro práci s AJAX technologií a abstraktní třídy pro přidání vlastních komponent nebo chování. Programátor bez znalostí této technologie je schopen připravit dynamické chování pro stránku a to vše díky tomu, že toto chování je obvykle napsáno v Javě a do technologie AJAX si to obvykle Wicket převede sám a připraví jQuery skripty.

4.5. Shrnutí Wicketu

Wicket je jednoznačně nástroj vhodný pro webové aplikace a je schopen se vyrovnat desktopové aplikaci díky integraci technologie AJAX a práci s jQuery. Existují dodatečné knihovny s integrací jQueryUI komponent pro Wicket. Wicket také obsahuje balík pro integraci s frameworky Spring a Hibernate a aktuálně se vývojáři Wicketu snaží o rozvoj frameworku a integraci nových technologií jako jsou HTML5, WebSocket[6] a další. Chyby ve frameworku jsou opravovány podle závažnosti. S problémem vám poradí nejen ostatní uživatelé frameworku, ale i jeho samotní vývojáři. Pravidelně vychází minoritní verze s úpravami.

[6] WebSocket je technologie umožňující obousměrnou komunikaci mezi klientem a serverem u webové aplikace.

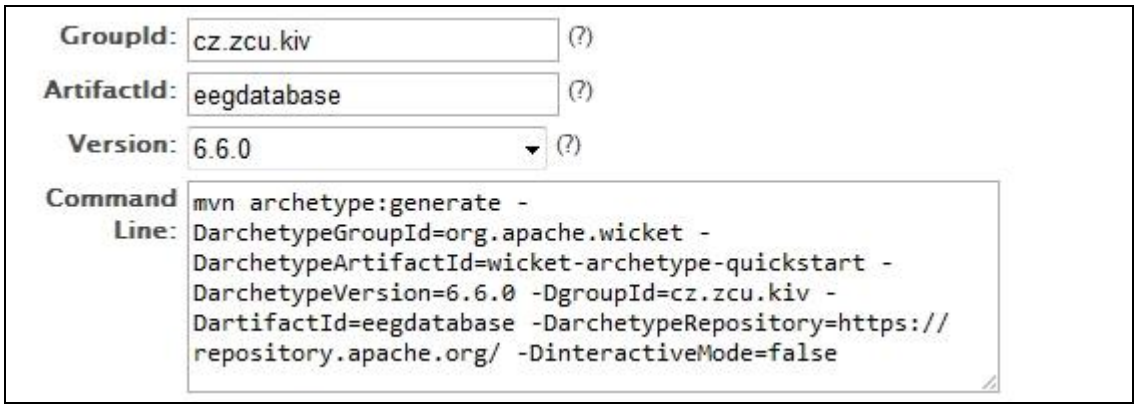
5.Implementace portálu ve Wicketu

Původní implementace Spring MVC frameworkem a šablonami JSP se nahradila novou vrstvou ve Wicketu. Protože se mění request base framework za component base framework, bude potřeba, aby se logika z controlerů tvořící jednu webovou stránku přenesla do nové logiky pro Wicket. Vzhled jako takový bude překopírován a využije se stávající design a struktura stránek.

Před samotnou implementací jednotlivých stránek je nutné záležitostí spojené se změnou frameworku. Mezi ně patří příprava hlavních tříd pro běh Wicket frameworku, jeho propojení s moduly Spring frameworku a napojení na Hibernate pro získání dat. Pro opakující se akce budou vytvořeny pomocné třídy a pro samotnou implementaci stránek abstraktní třídu jako předka, který už bude obsahovat záhlaví a zápatí stránky, které je na všech stránkách stejné.

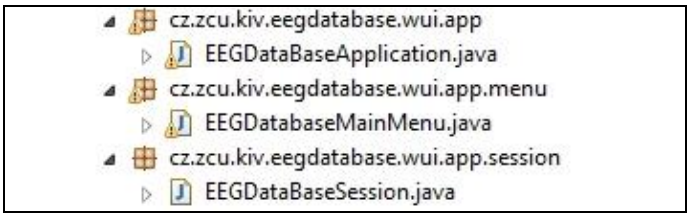
5.1. Wicket Core

Pro získání základních tříd nutných pro běh Wicketu byla použita funkce Quickstart na oficiálních webových stránkách Wicketu. Quickstart [13] připraví základní příkaz (obrázek 3) do nástroje Mavenu pro generování projektu, který už obsahuje základní třídy a konfigurační soubory nutné pro běh webové aplikace napsané ve Wicketu. Tento projekt bude obsahovat WicketApplication.class, HomePage.class, a web.xml. Tyto tři soubory byly použity pro novou implementaci portálu ve Wicketu.



Obrázek 4: Ukázka funkce Quickstart pro Wicket

V projektové struktuře vznikl balíček **cz.zcu.kiv.eegdatabase.wui**, do kterého se bude vytvářet kompletně nová implementace webového rozhraní pro Portál (obrázek 4). Třída **WicketApplication.class**, která slouží jako hlavní třída, byla umístěna do balíčku **cz.zcu.kiv.eegdatabase.wui.app**. Byla vytvořena vlastní implementace Session pojmenovaná EEGDatabaseSession, nachází se v balíčku **cz.zcu.kiv.eegdatabase.wui.app.session**. Do těchto dvou tříd bude později implementováno napojení na moduly Spring frameworku a zabezpečení.



Obrázek 5: Struktura baličku cz.zcu.kiv.eegdatabase.wui.app

V balíčku **cz.zcu.kiv.eegdatabase.wui.app.menu** se nachází enumeration třída popisující strukturu a jednotlivé položky v hlavním menu webové stránky (obrázek 5).

Obrázek 6: V rámečku označeno hlavní menu stránky

Ve složce **src/main/webapp/WEB-INF** se nacházely původní konfigurační soubory pro webovou aplikaci. Původní web.xml soubor byl při vývoji uchován pro pozdější potřeby. Nový web.xml pro Wicket sem byl umístěn a byl upraven podle původního, aby zůstala zachována konfigurace pro Spring, Hibernate, webové služby a ostatní frameworky, které portál využívá. Z konfigurace bylo odstraněno vše, co se týkalo původní vrstvy, kterou bude Wicket nahrazovat.

5.2. Spring

Spring se se svými moduly stará o jádro aplikace. Wicket podporuje práci se Springem pomocí vlastního modulu wicket-spring.

5.2.1. Spolupráce Wicketu a Springu

Tento modul obsahuje anotaci @SpringBean [14], která zajistí, že daný atribut bude inicializován instancí, jež je vytvořena v contextu Springu. Díky tomu lze ve Wicketu snadno používat Spring a jeho práci se závislostmi. Pro toto použití bylo potřeba upravit EEGDatabaseApplication třídu pro práci se Spring frameworkem.

Musela se implementovat rozhraní Springu:

```
public interface ApplicationContextAware extends Aware {
        void setApplicationContext(ApplicationContext applicationContext) throws BeansException;
}
```

A při vytváření instance třídy EEGDatabaseApplication v metodě *init()* bylo nutné přidat listener pro Spring, který bude pracovat s anotací @SpringBean:

```
getComponentInstantiationListeners().add(new SpringComponentInjector(this));
```

5.2.2. Konfigurace Springu

Pro načtení konfigurace a použití Springu sloužilo konfigurační nastavení ve web.xml. Toto nastavení se zkopírovalo do web.xml pro Wicket. Aby instance třídy EEGDatabaseApplication byla v contextu Springu a bylo možné pracovat se Springem, bylo nutné do web.xml přidat filtr pro práci se Springem a do konfigurace přidat nastavení pro vytvoření instance třídy EEGDatabaseApplication.

web.xml

```
...
<filter>
    <filter-name>myWicketFilter</filter-name>
    <filter-class>org.apache.wicket.protocol.http.WicketFilter</filter-class>
    <init-param>
      <param-name>applicationFactoryClassName</param-name>
      <param-value>org.apache.wicket.spring.SpringWebApplicationFactory</param-value>
    </init-param>
    ...
</filter>
…
```

applicationContext.xml

```
...
<bean id="wicketApplication " class="cz.zcu.kiv.eegdatabase.wui.app.EEGDataBaseApplication" />
...
```

Tímto bylo zajištěno, že při startu aplikace a po konfiguraci Springu bude připraveno vše i pro Wicket a budou doplňovány závislosti dle nastavení Springu nebo použití anotací @SpringBean.

5.2.3. @SpringBean nebo @Autowired

Spring obsahuje anotaci *@Autowired*, která se chová velice podobně jako anotace *@SpringBean*. Při práci s Wicket komponentami je důležité používat anotaci *@SpringBean*. *@SpringBean* doplňuje závislosti pouze do komponent. Anotaci *@Autowired* by v tomto případě nešlo použít, protože doplňuje závislosti pouze do instancí, které vytváří přímo Spring. Instance komponent a stránek si Wicket vytváří sám.

5.2.4. Úprava konfigurace pro Spring

Portál obsahoval větší množství konfiguračních souborů pro Spring a ostatní frameworky. Při implementaci Wicketu došlo k následujícím změnám v konfiguraci.

Vznikl obecný konfigurační soubor **applicationContext.xml** pro Spring. Do tohoto souboru byly přesunuty potřebné konfigurace, které se nacházely v souborech, jež už nebudou potřeba. Dále bude obsahovat konfiguraci objektů (tzv. bean[7]) vznikající při implementaci a nebudou mít svůj vlastní konfigurační soubor.

Soubor **controllers.xml** obsahoval konfiguraci controlerů pro modul Spring MVC, ale také obsahoval konfiguraci bean typu DAO pro práci Hibernatu a jiné beany. Tento konfigurační soubor byl promazán od všeho, co se netýkalo bean typu DAO, a byl přejmenován na **dao.xml**. Všechny beany, které byly pro konfiguraci MVC Springu, se smazaly. Všechny ostatní beany se přesunuly do **applicationContext.xml**.

[7] Bean je třída, která je instancována, zpravována a přiřazována Spring IoC modulem.

Ze souboru **security.xml** bylo použito nastavení pro autorizaci, která pracuje s modulem Spring Security. Soubor je možné odstranit z projektu, ale byl prozatím ponechán, dokud nebude implementace Wicketu kompletně hotová. Soubor **persistence.xml** je soubor pro nastavení Hibernatu a zůstal nezměněn. Ostatních konfiguračních souborů se implementace netýká.

5.2.5. Spring Security a Wicket

Původní verze portálu používala modul Spring Security pro zabezpečení a autorizaci uživatelských účtů. Jedním z požadavků při implementaci bylo, aby se modul Spring Security používal dál. Pro zabezpečení stránek byl proto použit Wicket modul Auth [15]. Tento modul spravuje zabezpečení stránek a komponent pomocí anotací. Autorizace uživatelů se dá snadno propojit s modulem Security od Springu. Pro použití modulu Auth se musely rozšířit základní třídy pro Wicket.

EEGDatabaseApplication třída musí mít předka AuthenticatedWebApplication, který si vynutí doplnit dvě metody.

```
protected abstract Class<? extends AbstractAuthenticatedWebSession> getWebSessionClass();
protected abstract Class<? extends WebPage> getSignInPageClass();
```

První metoda určuje, která třída reprezentuje zabezpečenou implementaci pro Session, aby modul věděl, kterou instanci má používat pro autorizovanou relaci. Druhá metoda vrací třídu, která slouží jako webová stránka pro autorizaci uživatele. Když bude chtít uživatel zobrazit některou webovou stránku a nebude autorizován, bude zobrazena nejdřív tato stránka pro autorizaci.

Do metody *init()* bylo doplněno nastavení pro autorizační strategii:

```
getSecuritySettings().setAuthorizationStrategy(new AnnotationsRoleAuthorizationStrategy(this));
getSecuritySettings().setUnauthorizedComponentInstantiationListener(this);
```

Implementace třídy Session musí mít předka AuthenticatedWebSession, který si vynutí doplnění metod pro autorizaci uživatele a získání práv a rolí uživatele.

```
public abstract boolean authenticate(final String username, final String password);
public abstract Roles getRoles();
```

Právě v těchto metodách dochází k propojení Spring Security a Wicket Auth. Wicket si autorizuje uživatele pomocí Spring Security, a pokud tato autorizace bude v pořádku, načtou se uživatelská oprávnění do relace. Použití anotací pro zabezpečení pak vypadá takto: *@AuthorizeInstantiation("ADMIN")*

Propojení Spring Security a Wicket Auth:

```
public abstract boolean authenticate(final String username, final String password) {
        boolean authenticated = false;
        try {
                Authentication authentication = authenticationManager.authenticate(new
                UsernamePasswordAuthenticationToken(username, password));
                SecurityContextHolder.getContext().setAuthentication(authentication);
                authenticated = authentication.isAuthenticated();
                this.setLoggedUser(facade.getPerson(username));
                this.createShoppingCart();
        } catch (AuthenticationException e) {
        error((String.format("User '%s' failed to login. Reason: %s", username, e.getMessage())));
        authenticated = false;
        }
    return authenticated;
}

public Roles getRoles() {
    Roles roles = new Roles();
    if (isSignedIn()) {
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        for (GrantedAuthority auth : authentication.getAuthorities())
          roles.add(auth.getAuthority());

    }
    return roles;
}
```

5.2.6. Spring Social a Wicket

V původní verzi Portálu bylo umožněno uživatelům se přihlašovat pomocí jejich účtů na sociálních sítích Facebook a LinkedIn. Tahle funkčnost musela být zachována. Přihlášení zajišťoval modul Spring Social předpřipraveným řešením ve spolupráci se Spring MVC modulem.

Aby bylo možné tuto funkčnost zachovat bez větších zásahů do systému, bylo rozhodnuto ponechat část modulu Spring MVC v Portále a využít ji nadále k autorizaci uživatelů. Jen bylo nutné propojit Spring Social a Wicket. Proces přihlášení pomocí Spring Social probíhá na čtyři fáze a propojení s Wicketem probíhá až ve čtvrté fázi.

Původní zjednodušený proces autorizace pomocí sociální sítě:

1. Uživatel provede volbu přihlášení pomocí sociální sítě.
2. Portál přesměruje uživatele na stránky sociální sítě pro samotné přihlášení.
3. Po přihlášení dojde k získání informací o uživateli ze sociální sítě.
4. Na základě získaných informací dojde k autorizaci a k přesměrování zpět na stránky Portálu.

K propojení dochází ve čtvrté fázi při přesměrování zpět na stránku Portálu **HomePage.class**, která provede autorizaci pro Wicket. Jakmile k tomu dochází, je relace ve Spring Security už autorizovaná. Toho po přesměrování na **HomePage.class** využijeme k autorizaci uživatele na úrovni zabezpečení Wicketu.
Při autorizaci uživatele pomocí normální registrace na Portálu dochází k autorizaci relace ve Spring Security přes autorizaci Wicketu. Při přihlášení přes sociální sítě dochází k jinému efektu. Při samotném přihlášení dojde k autorizaci relace ve Spring Security a až po přesměrování dojde k autorizaci relace na úrovni Wicketu.

Kompletní nastavení Spring Social modulu bylo přeneseno z ostatních konfiguračních souborů do soubor **socialContext.xml**.

Samotné propojení se provedlo jednoduchými úpravami v konfiguračních souborech pro Spring Social modul:

facebook.properties:

```
redirect.uri = http://147.228.64.172:8080/home-page
```

socialnContext:

```
<bean id="providerSignInController"
class="org.springframework.social.connect.web.ProviderSignInController">
    ...
    <property name="postSignInUrl" value="/home-page" />
</bean>
```

V obou nastaveních je důležitá ta část URL, která určuje stránku, na kterou se bude Portál přesměrovávat. Po autorizaci na sociální síti dojde k přesměrování na správnou stránku, která provede autorizaci Wicketu.

EEGDatabaseAplication.class:

```
mountPage("home-page", HomePage.class);
```

Toto nastavení určuje, která stránka bude na dané URL. **HomePage.class** obsahuje podmínku pro testování, zda nedošlo k přihlášení pomocí sociální sítě.

HomePage.class:

```
if (EEGDataBaseSession.get().authenticatedSocial()) {
    throw new RestartResponseAtInterceptPageException(WelcomePage.class);
}
```

EEGDatabaseSession.class si zařídí, že dojde k autorizaci relace pro Wicket, pokud se uživatel přihlásil přes sociální síť. Po autorizaci na úrovni relace ve Wicketu je uživatel přesměrován na uvítací stránku.

EEGDataBaseSession.class:

```
public boolean authenticatedSocial() {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        ...
    if (authentication.isAuthenticated()) {
      String username = "";
      if (authentication.getPrincipal() instanceof User)
        username = ((User) authentication.getPrincipal()).getUsername();
      else if (authentication.getPrincipal() instanceof Person)
        username = ((Person) authentication.getPrincipal()).getUsername();

      return signIn(username, SOCIAL_PASSWD);
    }
    return false;
}
```

5.3. Hibernate

Hibernate zajišťuje práci s databází pomocí mapování jednotlivých tabulek a jejich řádků do kolekcí tříd v Javě. Díky tomu lze pracovat s daty pomocí těchto tříd, stále však dochází k používání transakcí při práci s databází. O tyto transakce je možné se starat více způsoby.

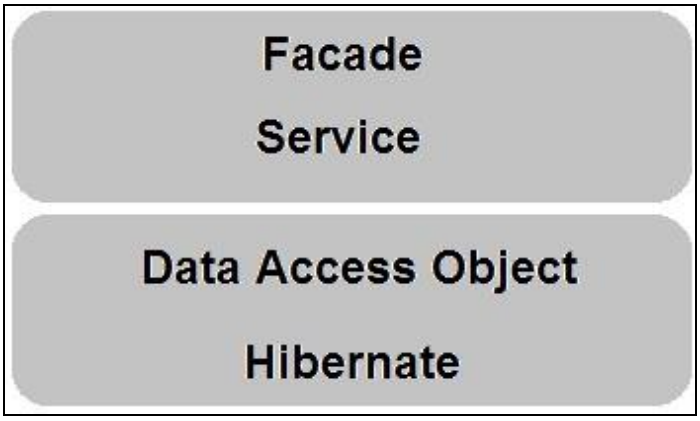
5.3.1. Původní řízení transakcí

O transakce se staral transakční manažer od Springu. Byl nastaven tak, aby pro všechna volání metod z balíčku **cz.zcu.kiv.eegdatabase.data.service** byla použita transakce a pro metody začínající výrazem „**get**“ byla transakce pouze pro čtení. Dále byly použity pro manipulaci s transakcemi anotace *@Transactional*. V kódu tyto anotace byly, ale konfigurace pro jejich použití chyběla, tudíž nebyly používány. Proto v tuto chvíli všechny ostatní transakce řešil Hibernate sám a pokud transakce nebyla použita, vytvářel transakce pro každou operaci nad databází. Toto chování není doporučováno, protože zahlcuje databázi malými transakcemi a ještě je Hibernate zaměstnán jejich vytvářením. To způsobuje citelné zpomalení práce systému.

5.3.2. Změna v řízení transakcí

Pro práci s Hibernatem jsou naimplementovány třídy označené jako DAO. Pomocí těchto tříd se provádí manipulace nad jednotlivými objekty Hibernatu. Pro zlepšení byla navržena vícevrstvá architektura pro manipulaci s daty, v níž dojde ke vzniku dalších vrstev nad úrovní DAO. Tyto vrstvy budou určitým způsobem fungovat jako logika pro řízení a manipulaci s daty Hibernatu.

Vznikly vrstvy Facade a Service, které budou zajišťovat rozvrstvení, jež nám zařídí lepší přehlednost a udržitelnost kódu a hlavně jednotnou manipulaci s transakcemi. Na obrázku č. 7 je vidět, jak došlo k přidání vrstev. Poslední dvě spodní vrstvy DAO a Hibernate byly již v původní verzi Portálu. Nově byly přidány vrstvy Facade a Service.



Obrázek 7: Jednotlivé vrstvy pro práci s Hibernatem

Service vrstva zajišťuje manipulaci s transakcemi. Všechny metody v této vrstvě používají anotaci @Transactional pro řízení transakcí a manipulují s daty, které obdrží od vrstvy DAO. Během implementace Wicketu byla logika na úrovni dat z původních controlerů přesunuta do této vrstvy. Například při vytvoření výzkumné skupiny dochází i k vytvoření číselných seznamů se základními hodnotami pro novou skupinu. Tato logika byla přesunuta do příslušné Service třídy pro manipulaci s výzkumnými skupinami. Vrstva Facade byla přidána s myšlenkou, že by se pro ni v budoucnu mohlo najít využití, což se dosud nestalo. Například by bylo možné ji využít na dodatečnou kontrolu práv při přístupu k datům. Přesto zatím nedojde k jejímu odstranění.

5.4. Wicket komponenty

Pro implementaci Wicketu do Portálu vznikly některé komponenty pro usnadnění vývoje, kterými se předcházelo nadbytečnosti zdrojového kódu.

5.4.1. Komponenty pro třídu Timestamp

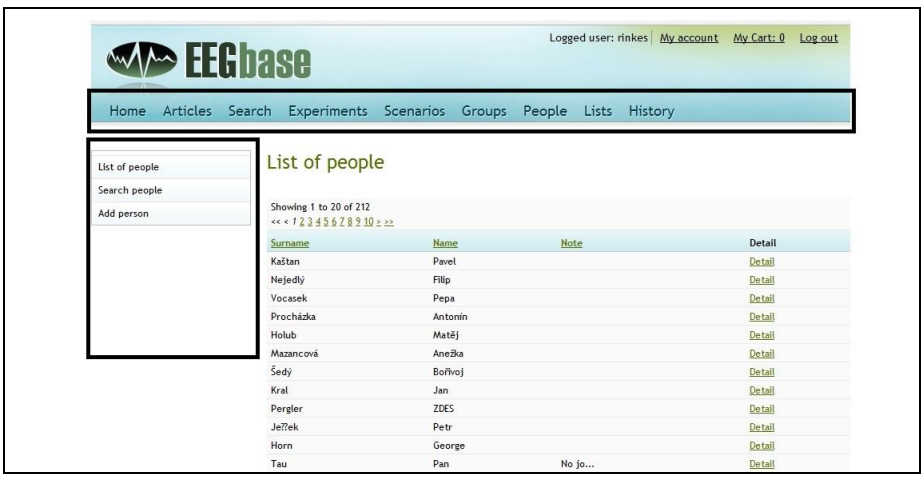
V Portálu se pro reprezentaci data a času používá třída Timestamp z balíčku **java.sql**. Pro jednodušší zobrazovaní a načítání hodnot vznikla skupina komponent pro třídu Timestamp, které usnadnily vývoj stránek ve Wicketu.

TimestampConverter.class třída pomáhá při převodu vstupních hodnot z textového pole do Timestamp instance, která je uložena v modelu. Jeho použití lze nastavit globálně pro celou aplikaci nebo lokálně jen pro dané textové pole při překrytí metody *public <C> IConverter<C> getConverter(Class<C> type)*.

TimestampLabel.class třída je rozšířená komponenta pro výpis hodnoty. Bylo přidáno formátování výstupní hodnoty Timestamp objektu. Do výstupního HTML je vložen formátovaný datum a čas podle formátu, který určí programátor, například rok/měsíc/den. Třídy **TimestampPropertyColumn.class** a **TimestampModel.class** se chovají podobně jako TimestampLabel. Výstupní hodnota objektu Timestamp se naformátuje dle použitého formátovacího řetězce. **TimestampPropertyColumn.class** je použit pro formátování hodnot v komponentě tabulky.

5.4.2. Komponenty pro generování Menu

Portál pro navigaci po stránkách používá dvě navigační menu. Hlavní horizontální menu slouží pro navigaci mezi jednotlivými sekcemi stránek. Druhé vertikální navigační menu slouží pro navigaci jednotlivými stránkami v dané sekci. Na obrázku č. 8 je vidět hlavní menu celého portálu a po té vertikální menu pro sekci zabývající se uživatelem. Obsahuje stránky pro výpis uživatelů a pro přidání nového uživatele.



Obrázek 8: Horizontalně hlavní menu, vertikálně sekční menu.

Pro generování hlavního menu slouží komponenta **MainMenu.class**. Její implementace je upravena tak, aby ji stačilo přidat na stránku a poskytnout ji jednoduchý popis menu pomocí enumeration, jako je například **EEGDatabaseMainMenu.java**. Aby byl popis dostačující, je nutné implementovat pro popis menu rozhraní **MenuDefinition**, které si vynutí informace nutné ke složení funkčního menu.

```
public interface MenuDefinition

   boolean isLinkable(); // jestli definice prvku menu je pro konečný odkaz na stránku
   boolean hasSubmenu(); // jestli definice prvku obsahuje podmenu
   String getTitle(); // nadpis prvku v menu
   MenuDefinition[] getSubmenu(); // podmenu které prvek obsahuje
   Class<?> getPage(); // třída reprezentující stránku na kterou odkaz povede odkaz
}
```

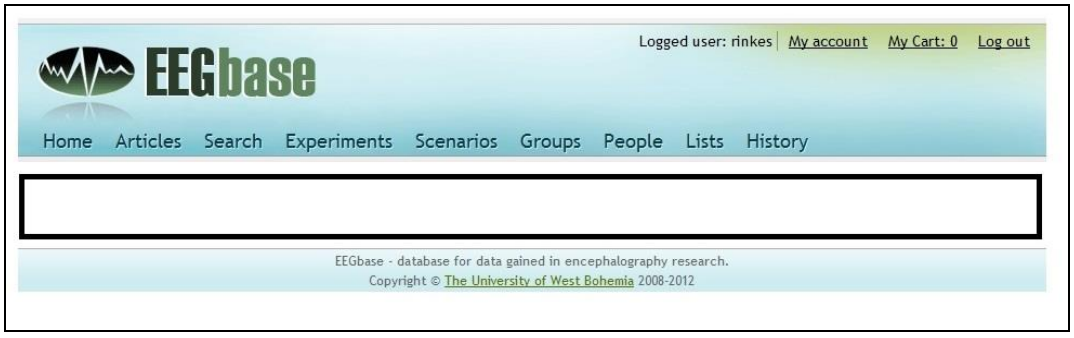
Komponenta dle popisu vygeneruje přímo HTML kód, který předá Wicketu pří generování výstupní podoby stránky.

Pro generování vertikálního menu pro navigaci po dané sekci je připravena komponenta **ButtonPageMenu.class**. Třída také vyžaduje popis toho, jak má menu vypadat. Opět byl pro popis použit enumeration s rozhraním **IButtonPageMenu**. Rozdíl v implementaci oproti **MainMenu.class** je ve vnitřním procesu pro generování menu. **ButtonPageMenu.class** je implementována použitím standardních Wicket komponent místo přímým generováním HTML kódu. Rozdíl mezi oběma implementacemi menu dovoluje, aby hlavní menu Portálu bylo možné navrhnout jako víceúrovňové. **ButtonPageMenu.class** je jednoduché menu s jednou úrovní.

```
public interface IButtonPageMenu {

   Class<? extends MenuPage> getPageClass(); // třída reprezentující stránku na kterou odkaz povede

   String getPageTitleKey(); // klič reprezentující nadpis, uložený  v .properties souboru

   PageParameters getPageParameters(); // parametry, které budou stránce předány při přechodu
}
```

5.4.3. Základní implementace webové stránky

Na webové stránce obvykle najdeme prvky, které se opakují na všech stránkách. V případě Portálu je to hlavička a patička stránky, hlavní menu a obsah HTML záhlaví. Protože Wicket umožnuje pracovat s dědičností u stránky, bylo dobré připravit si třídu, jež bude obsahovat tyto neměnné informace. Při využití dědičnosti se budou doplňovat jen proměnné části, což ušetří čas vývojářům i množství kódu, který bude potřeba napsat. Budoucí úpravy budou jednodušší, pokud bude nutné upravovat záhlaví nebo zápatí stránky. Následující obrázek č. 9 ukazuje vzhled základní stránky, kterou použijeme jako předka pro nově implementované stránky.



Obrázek 9: podoba základní stránky - MenuPage.class

MenuPage.class je implementace základní stránky. Všechny ostatní stránky ji používají jako předka a obsah stránek se doplňuje do rámečku vyznačeného na obrázku č.9. **MenuPage.class** dědí od **BasePage.class** ještě dodatečné vlastnosti pro bezpečnost a nastavení HTML záhlaví. Celá **MenuPage.class** obsahuje:

- do HTML záhlaví doplněné CSS soubory pro styly a design stránek.
- do HTML záhlaví doplněný nadpis pro stránky. Nadpis lze nastavit pomocí metody *protected void setPageTitle(IModel<String> model)*. Pokud není nadpis vyplněn, bude HTML záhlaví vždy obsahovat řetězec z .properties souboru pod klíčem *title.app*.
- připravené záhlaví stránky s logem, odkazy pro přihlášení/odhlášení, registraci a odkaz na detail přihlášeného uživatele.
- připravené zápatí stránky.

- komponentu FeedbackPanelu pro zprávy z validace formuláře. Komponenta pro tento typ zprávy je přidána, aby ji obsahovala každá stránka, a tedy bylo možné tyto zprávy všude zobrazovat.
- parametr DEFAULT_PARAM_ID, který se používá pro jednoduchý přechod mezi stránkami, kde se předávají parametry. Pomocí tohoto parametru je pak možné rychle vytvářet parametr pro přechod a po přechodu na stránku z něj získat hodnotu.

5.4.4. Pomocné třídy – Utils

Pro opakující se stejné úseky kódu vznikly pomocné třídy označované jako Utils.

- ResourceUtils je nejdůležitější pomocná třída. Wicket si při startu aplikace načítá soubory s lokalizací a udržuje si mapu hodnot a klíčů z těchto souborů v paměti. Přístup k těmto hodnotám se provádí pomocí Resource modelů, které lze rovnou vkládat do komponent. Aby se zjednodušil postup práce s lokalizací, obsahuje tato pomocná třída dvě skupiny metod pro získání hodnot pro lokalizaci.
    - Metody *getString(...)* vracejí hodnotu pro daný klíč jako řetězec znaků reprezentovaný datovým typem String.
    - Metody *getModel(...)* vracejí Wicket model obsahující hodnotu pro daný klíč.
- PageParametersUtils umožňuje jednodušší vytváření instancí **PageParameters.class**, které slouží k předávání parametrů mezi stránkami. V PageParametersUtils jsou implementovány metody pro:
    - rychlé vytvoření instance obsahující jeden parametr pod jedním klíčem.
    - rychlé vytvoření parametru s jednou hodnotou pod klíčem DEFAULT_PARAM_ID, který obsahuje každá stránka, která je potomkem třídy **BasePage.class**.
    - metoda *getUrlForPage(Class page, PageParameters parameters)*, která pro danou stránku předanou atributem *page* a dané parametry jako atribut *parameters* vytvoří úplnou podobu URL adresy.

- FileUtils je pomocná třída pro práci se soubory. Aktuálně obsahuje jednu metodu pro přípravu stažení souboru. Tato akce potřebuje, aby byla pro Wicket jádro připravena instance **ResourceStreamRequestHandler.class**, která bude do jádra předána a jádro umožní uživateli soubor stáhnout. Příprava tohoto objektu je delší úsek kódu, který by se mohl opakovat.

- StringUtils obsahuje řetězce pro formátování hodnot data a času a metodu pro generování náhodného řetězce znaků.

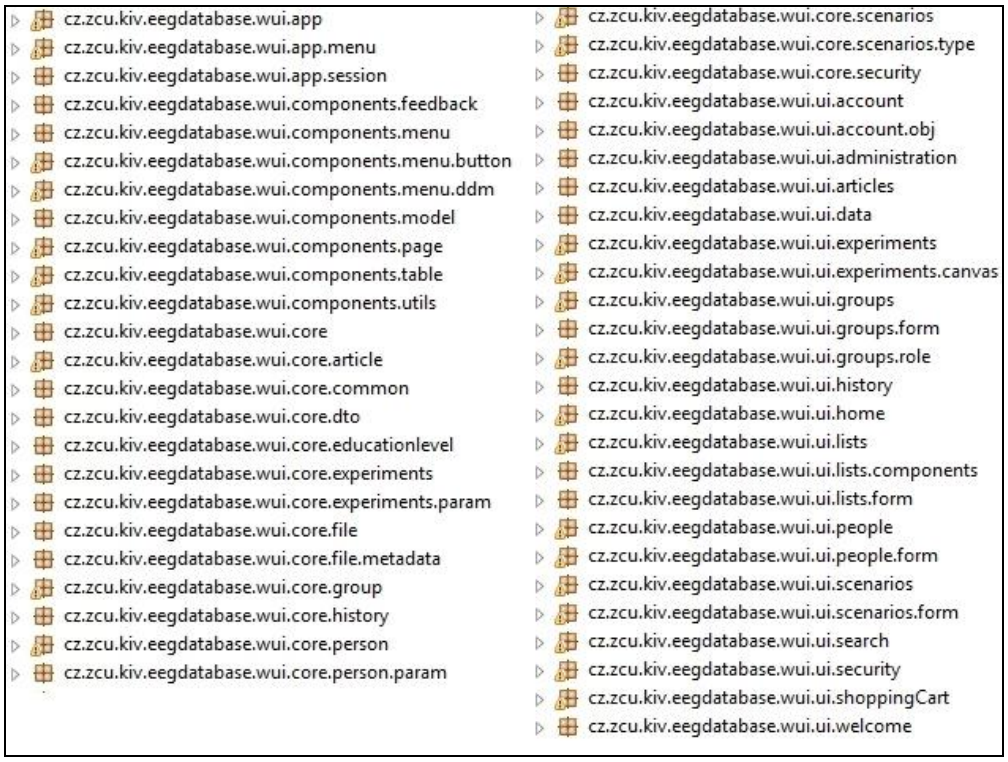
5.4.5. Převod JSP šablon do Wicketu

Během implementace byly postupně převáděny jednotlivé stránky z technologie Spring MVC a z JSP šablon do nových implementací za použití komponent Wicketu. Šablona JSP poskytovala vzhled a strukturu, a díky těmto znalostem bylo možné rychle navrhovat strukturu komponent pro Wicket. Logika, která obsluhovala stránku, se přenesla do Wicketu z controllerů a logika pro manipulaci s daty se přenesla buď do Wicketu, nebo do vrstvy Service. Validace se ze Spring MVC přenesly do jednotlivých validátorů ve Wicketu. Vzhled a design stránek zůstal zachován. Struktura HTML se musela místy přizpůsobit pro Wicket, ale žádné větší zásahy nebyly potřeba.

Některé stránky se spojily do jedné. Obvykle stránka s formulářem má druhou stránku se zprávou o výsledku zpracování formuláře. O tuto informaci se nově starají zprávy pro Feedback panely. Díky tomu se nemusely všechny stránky implementovat znova. Rozdělení jedné stránky na dvě došlo u domovské stránky, která obsahovala dva různé vzhledy podle toho, zda byl uživatel přihlášen nebo ne. Zde došlo k rozdělení na domovskou a uvítací stránku, aby kód stránky nebyl zbytečně dlouhý a složitý.

5.4.6. Struktura implementace

Pro implementaci Wicketu byla navrhnuta struktura balíčků se snahou o dobrou přehlednost (úplná struktura je znázorněna na obrázku č. 10):



Obrázek 10: kompletní struktura implementace Portálu ve Wicketu.

- **cz.zcu.kiv.eegdatabase.wui.app** - hlavní třídy nutné pro běh Wicketu.
- **cz.zcu.kiv.eegdatabase.wui.core** - třídy pro práci s daty a Hibernatem (Facade, Service, …).
- **cz.zcu.kiv.eegdatabase.wui.components** - vlastní komponenty a pomocné třídy pro Wicket.
- **cz.zcu.kiv.eegdatabase.wui.ui** – jednotlivé implementace webových stránek.
- **cz.zcu.kiv.eegdatabase.wui.ui.accounts –** implementace stránek pro práci s uživatelským účtem:
    - AccountOverviewPage.class – stránka s náhledem uživatelského účtu.
    - ChangePasswordPage.class – stránka pro změnu hesla uživatelského účtu.
- **cz.zcu.kiv.eegdatabase.wui.ui.administration –** implementace stránek pro administraci Portálu:
    - ChangeUserRolePage.class – stránka pro změnu uživatelské role.

- **cz.zcu.kiv.eegdatabase.wui.ui.articles** – implementace stránek pro sekci s články na Portálu. Aktuálně není implementováno.
- **cz.zcu.kiv.eegdatabase.wui.ui.data –** implementace stránek pro manipulaci s datovými soubory na Portálu:
    - DataFileDetailPage.class – implementace stránky pro zobrazení informací o datovém souboru.
- **cz.zcu.kiv.eegdatabase.wui.ui.experiments –** implementace stránek pro sekci s experimentama:
    - ListExperimentsPage.class – stránka se seznamem experimentů a základními informacemi.
    - ExperimentDetailPage.class – stránka pro zobrazení informací o experimentu.
    - ExperimentDownloadPage.class – stránka pro stažení dat k experimentu.
- **cz.zcu.kiv.eegdatabase.wui.ui.groups –** implementace stránek pro sekci s výzkumnými skupinami:
    - ListResearchGroupsPage.class – stránka se seznamem výzkumných skupin a základními informacemi.
    - ResearchGroupDetailPage.class – stránka pro zobrazení informací o výzkumné skupině.
    - ListOfMembersGroupPage.class – stránka se seznamem členů dané výzkumné skupiny.
    - MyGroupPage.class – stránka se seznamem skupin pro aktuálně přihlášeného uživatele.
- **cz.zcu.kiv.eegdatabase.wui.ui.groups.form –** implementace stránek pro manipulaci s výzkumnými skupinami:
    - AddMemberToGroupPage.class – stránka pro přidání uživatele do výzkumné skupiny.
    - ResearchGroupFormPage.class – stránka pro vytvoření a úpravu výzkumné skupiny.

- **cz.zcu.kiv.eegdatabase.wui.ui.groups.role –** implementace stránek pro manipulaci se skupinovými rolemi členů výzkumných skupin:
    - GroupRoleAcceptPage.class – stránka pro reakci na požadavek o skupinovou roli od člena skupiny.
    - GroupRoleRequestPage.class – stránka pro vytvoření požadavku o skupinovou roli pro člena skupiny.
- **cz.zcu.kiv.eegdatabase.wui.ui.history –** implementace stránek pro sekci historie. Aktuálně není implementováno.
- **cz.zcu.kiv.eegdatabase.wui.ui.home –** implementace domovské stránky
    - HomePage.class – domovská stránka pro nepřihlášeného uživatele. Po přihlášení je uživatel přesměrován vždy na WelcomePage.class.
- **cz.zcu.kiv.eegdatabase.wui.ui.lists –** implementace stránek s číselníky pro nastavení parametrů experimentů.
    - ListListsPage.class – stránka pro navigaci po jednotlivých čáselníkách.
    - ListArtifaktDefinitionsPage.class – stránka pro číselník definic artefaktů.
    - ListExperimentOptParamPage.class – stránka pro číselník parametrů experimentu.
    - ListFileMetadataPage.class – stránka pro číselník metadata parametrů souborů.
    - ListHardwareDefinitionsPage.class – stránka pro číselník definic hardwaru.
    - ListPersonOptParamPage.class – stránka pro číselník parametrů osoby.
    - ListWeatherDefinitionsPage.class – stránka pro číselník definic počasí.
- **cz.zcu.kiv.eegdatabase.wui.ui.lists.form –** implementace stránek pro manipulaci s číselníky:
    - ArtifactFormPage.class – stránka pro přidání a úpravu definic artefaktů.
    - ExperimentsOptParamFormPage.class – stránka pro přidání a úpravu parametrů experimentů.
    - FileMetadataFormPage.class – stránka pro přidání a úpravu metadata parametrů souborů.
    - HardwareFormPage.class – stránka pro přidání a úpravu definic hardwaru.
    - PersonOptParamFormPage.class – stránka pro přidání a úpravu parametrů osob.

    - WeatherFormPage.class – stránka pro přidání a úpravu definice počasí.
- **cz.zcu.kiv.eegdatabase.wui.ui.people –** implementace stránek pro sekci osob:
    - ListPersonPage.class – stránka se seznamem osob.
    - PersonDetailPage.class – stránka pro zobrazení informací o osobě.
- **cz.zcu.kiv.eegdatabase.wui.ui.people.form –** implementace stránek pro manipulaci s osobami:
    - PersonAddParamFormPage.class – stránka pro zadání parametrů osobě.
    - PersonFormPage.class – stránka pro přidání a úpravu osoby.
- **cz.zcu.kiv.eegdatabase.wui.ui.scenarios –** implementace stránek pro sekci scénářů experimentů:
    - ListScenariosPage.class – stránka se seznamem scénářů pro experimenty a základní informace.
    - ScenarioDetailPage.class – stránka pro zobrazení informací o scénáři pro experimenty.
- **cz.zcu.kiv.eegdatabase.wui.ui.scenarios.form –** implementace stránek pro manipulaci se scénáři experimentů:
    - ScenarioFormPage.class – stránka pro přidání a úpravu scénáře.
    - ScenarioSchemaFormPage.class – stránka pro přidání a úpravu schéma scénáře.
- **cz.zcu.kiv.eegdatabase.wui.ui.security –** implementace stránek pro správu a bezpečnost Portálu:
    - RegistrationPage.class – stránka pro registraci nových uživatelů.
    - ConfirmPage.class – stránka pro potvrzení registrace nového uživatele.
    - ForgottenPasswordPage.class – stránka pro získání nového hesla při ztrátě nebo zapomenutí.
- **cz.zcu.kiv.eegdatabase.wui.ui.welcome –** implementace uvítací stránky.
    - WelcomePage.class – domovská stránka pro přihlášeného uživatele s přehledem informací o článcích, experimentech a výzkumných skupinách.
- **cz.zcu.kiv.eegdatabase.wui.ui.search –** implementace vyhledávání informací na Portále. Aktuálně implementováno jiným autorem jako seminární práce.
- **cz.zcu.kiv.eegdatabase.wui.ui.shoppingCart –** výzkumná implementace. Aktuálně implementováno jiným autorem.

6. Zhodnocení a závěr

6.1. Zhodnocení dosažených výsledků

Během implementace došlo k převedení 48 webových stránek, přičemž některé původní stránky byly spojeny do jedné. Odhadem byly převedeny dvě třetiny všech stránek tvořící Portál, ale pořád to není dostatečné množství, aby mohla být nová implementace Portálu nasazena k užívání. Stránky jsou otestovány a fungují po vzoru původních stránek v Portálu.

Testování probíhalo uživatelsky při implementaci. Kontrolovala se správná funkčnost jednotlivých stránek, vzhled stránek a správné chovaní stránek proti původní implementaci. Rozsáhlejší testovaní i s pomocí testovacích nástrojů provádí Tomáš Pokryvka ve své bakalářské práci.

Dle nástroje CLOC [16], který analyzuje zdrojové kódy, bylo možné zjistit přibližný rozsah provedené implementace. Vzniklo nových 175 tříd v jazyce Java a 47 HTML souborů se strukturou komponent pro Wicket. Celkem vzniklo nových 12 tisíc řádků kódu jazyka Java a 2 tisíce řádků HTML kódu.

6.2. Návrhy na úpravy a další vývoj

- Změnit implementaci pro datum a čas. Aktuálně používaná třída Timestamp je pracná. Hibernate a Wicket umí velice dobře spolupracovat s knihovnou Joda-Time [17], která se dnes velice hojně používá a práce s ní je jednodušší.
- Přepsat všechny výčty hodnot, které jsou použity v Portále, do třídy typu Enumeration. Výčty hodnot jsou uloženy pouze v databázi nebo byly přímo v šablonách JSP. Tento přístup je přítěží při rozšiřování, úpravách a práci s těmito hodnotami.
- Přesunout všechny konstanty použité v projektu do CoreConstants.class nebo do pomocných tříd, aby byly na jednom místě.

- Upravit volání Facade, Service, DAO pro předávání informace o řazení prvků pokud volání vrací kolekci dat. Řazení dat bude obstarávat databáze, která to bude provádět rychleji než řazení, které bylo implementováno pomocí porovnávání prvků v kolekci objektem Comparator.

6.3. Závěr

Cílem této práce bylo seznámit se s aktuálním stavem EEG/ERP Portálu, prozkoumat dnešní nabídku frameworků pro tvorbu webového rozhraní s použitím jazyka Java a nalézt alternativu za aktuálně používanou technologii. Byl zvolen framework Wicket. Před samotnou implementací byly probrány klíčové vlastnosti a funkce Wicketu a byl nastíněn princip vývoje s tímto frameworkem. Došlo k implementování nového webového rozhraní pro Portál ve frameworku Wicket a kromě návrhů na úpravy je nutné ještě dokončit některé sekce Portálu. I tak lze považovat práci za úspěšnou, neboť se podařilo implementovat velkou část původního Portálu. Samotné dokončení, údržba a rozšiřování bude jednodušší díky dobrým výukovým materiálům pro Wicket a faktu, že Wicket je založen pouze na technologiích Java a HTML, které jsou vývojářům dobře známy.

Literatura

[1] Jan Štěbeták. Computational tools in eeg/erp portal. Master's thesis, University of WestBohemia, Faculty of Applied Sciences, 2011[cit. 2013-01-22].
URL: https://portal.zcu.cz/stag?urlid=prohlizeni-prace-detail&praceIdno=41416.

[2] BAUER, Christian a Gavin KING. Java persistence with Hibernate. rev. ed. Greenwich: Manning Publications, 2007, ISBN 19-323-9488-5.

[3] WALLS, Craig a Ryan BREIDENBACH. Spring in action. rev. ed. Greenwich: Manning Publications, 2005, ISBN 19-323-9435-4.

[4] BERGSTEN, Hans. JavaServer Pages: 3rd Edition. O'Reilly Media, June 2009.
ISBN 978-0-596-10421-4.

[5] GOOGLE. Google Web Toolkit [online]. [cit. 2013-04-26].
Dostupné z: https://developers.google.com/web-toolkit/

[6] Apache: Wicket. [online]. [cit. 2013-04-26].
Dostupné z: http://wicket.apache.org/

[7] SPRINGSOURCE. Grails [online]. [cit. 2013-04-26].
Dostupné z: http://grails.org/Documentation

[8] SPRINGSOURCE. Groovy [online]. [cit. 2013-04-26].
Dostupné z: http://groovy.codehaus.org/

[9] DASHORST, Martijn a Eelco HILLENIUS. Wicket in action. Greenwich: Manning Publications, 2009, ISBN 19-323-9498-2.

[10] Apache Wicket: Core. [online]. [cit. 2013-04-03].
Dostupné z: https://cwiki.apache.org/WICKET/wicket-inside.html

[11] Apache Wicket: Models. [online]. [cit. 2013-04-04].
Dostupné z: https://cwiki.apache.org/WICKET/working-with-wicket-models.html

[12] Apache Wicket: Wicket Tags. [online]. [cit. 2013-04-07].
Dostupné z: https://cwiki.apache.org/WICKET/wickets-xhtml-tags.html

[13] Apache Wicket: Quickstart. [online]. [cit. 2013-04-15].
Dostupné z: http://wicket.apache.org/start/quickstart.html

[14] Apache Wicket: @SpringBean. [online]. [cit. 2013-04-16]. Dostupné z: https://cwiki.apache.org/WICKET/spring.html#Spring-AnnotationbasedApproach

[15] Apache Wicket: Auth. [online]. [cit. 2013-04-16].
Dostupné z: http://wicket.apache.org/learn/projects/authroles.html

[16] CLOC: Count Lines of Code. [online]. [cit. 2013-04-25].
Dostupné z: http://cloc.sourceforge.net/

[17] Joda-Time: Java date and time API. [online]. [cit. 2013-04-25].
Dostupné z: http://joda-time.sourceforge.net/