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

Bakalářská práce

Responzivní design portálu pro správu organizací Jasenverkko.fi

Plzeň 2015 Marek Rasocha

Prohlášení

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

V Plzni dne 4.5.2015

Marek Rasocha

Poděkování

Touto cestou bych rád poděkoval vedoucímu této bakalářské práce Ing. Jakubu Daňkovi za trpělivost, cenné rady při vedení práce a za čas, který mi při konzultacích věnoval.

Abstract

The goal of this bachelor thesis is to create a responsive presentation layer for portal Jasenverkko.fi. The thesis introduces the portal Jasenverkko.fi and presents basic principles of responsive design and web frameworks Apache Wicket and Twitter Bootstrap. The practical part of the thesis deals with implementation of the portal's presentation layer and its testing on both PC and mobile devices.

Abstrakt

Tématem této bakalářské práce je vytvoření responzivního designu portálu pro správu organizací Jasenverkko.fi. Tato práce nejprve čtenáře seznamuje se samotným portálem Jasenverkko.fi a principy responzivního designu. Práce pokračuje v zavedení čtenáře do problematiky frameworků Apache Wicket a Twitter Bootstrap. V praktické části se práce zabývá implementací prezentační vrstvy zmíněného portálu. Tato implementace bude na závěr otestována na desktopu a mobilním zařízení s operačním systémem Android.

Obsah

1 Úvod

Tématem této práce je vytvoření responzivního designu portálu pro správu organizací Jasenverkko.fi. Jedná se o webovou aplikaci, která je určena pro ulehčení správy a řízení organizacích všech druhů. Současná implementace aplikace neodpovídá požadavkům na moderní webové portály, protože je určena pouze pro zobrazení na počítači. Tyto aplikace neberou v úvahu, na jak velkém displeji se zobrazují. Pro uživatele jsou nepohodlné kvůli manuálnímu přizpůsobování stránek na velikost svého displeje. Výsledkem této práce má být webová aplikace, která se přizpůsobí jakémukoliv displeji. Na všech displejích bude čitelná bez nutnosti manuálních úprav. Má splňovat principy responzivního designu. Funkčnost této aplikace musí zůstat nezměněná.

Teoretická část této práce informuje čtenáře o aktuální implementaci prezentační vrstvy aplikace. Dále seznámí čtenáře se základními vlastnostmi frameworků Twitter Bootstrap a Apache Wicket 6.x. Čtenář bude taky krátce informován o knihovně wicket-bootstrap. Na závěr jsou uvedeny základní principy responzivního designu.

V praktické části se práce zabývá samotnou implementací prezentační vrstvy portálu, který bude splňovat požadavky responzivního designu. Důležitým požadavkem je, aby funkčnost této aplikace zůstala nezměněná. Dále je nutné dodržet základní rozložení stránek a základní barevné schéma aplikace. Na závěr se práce zabývá testováním nové implementace této aplikace. Testování musí probíhat alespoň na mobilním zařízení s operačním systémem Android a počítači. Testování musí být provedeno alespoň na aktuálních verzích uvedených prohlížečů (Chrome, Firefox, Internet Explorer a Opera).

2 Portál Jasenverkko.fi

Portál Jasenverkko.fi je webová aplikace, jejíž úkolem je zjednodušit a ulehčit spravování organizací. Organizace mohou obsahovat různé množství oddělení, a také velké množství členů. Tento portál umožňuje mít všechny události, platby a členy pod kontrolou. Pověřený uživatel může ostatním uživatelům posílat zprávy, informovat je o událostech, povinnostech a i o placení libovolných příspěvků. Aplikace umožňuje správci organizace mít přehled o všech platbách a událostech. Umožňuje zjistit, kdo danou akci splnil či nesplnil nebo kdo se dané události zúčastnil apod. Z pohledu člena organizace přináší tento portál ucelené informace o jeho povinnostech. Například jakých událostí se musí účastnit, o povinnosti placení příspěvků, o datu jejich splatnosti atd. Všechny zmíněné funkce lze spravovat i je rozesílat ostatním členům.

Aplikace také umožňuje uživatelům zjistit informace o různých organizacích. Například adresu organizace, počet členů v organizaci nebo dokonce informace o veřejných událostech. Každý uživatel vyplňuje určité množství informací. Soukromé informace může vidět jen on popř. jeho nadřízený. Veřejné informace jsou přístupné komukoliv. Pokud uživatel patří do dané organizace, může zjistit i velké množství privátních informací o dané organizaci. Organizace totiž také obsahují soukromé a veřejné informace.

Tato práce se zabývá prezentační vrstvou portálu Jasenverkko.fi. Není proto důležité vědět, jak je daná aplikace implementována. Zajímá nás pouze samotný vzhled webových stránek. Prezentační vrstva aplikace je implementována pomocí frameworku Apache Wicket a skládá se asi z 37 stránek. Každá z těchto stránek dědí od *AbstractPage*, která definuje základní vzhled stránky. Každá stránka pouze zpracuje parametry a vytvoří potřebnou komponentu, kterých je v aplikaci kolem dvou stovek.

Problém této implementace spočívá v tom, že je určena pro prohlížení na počítači. Pokud chce uživatel tyto stránky zobrazit například na mobilu, musí si je přizpůsobit na svoji velikost displeje. To provede tak, že se stránkou manipuluje jako s obrázkem. Lze ji přibližovat, oddalovat nebo posouvat. Cílem této práce je přeimplementovat prezentační vrstvu tak, aby se stránky přizpůsobily uživateli. To znamená, že aplikace bude splňovat principy responzivního webu.

3 Principy responzivního webu

Responzivní web nám říká, že stylovaní HTML dokumentu zaručí optimalizaci stránek pro všechny druhy zařízení (mobily, počítače, netbooky, tablety atd.)[3, 2]. V dnešní době existuje velké množství rozlišení. Je tedy jasné, že ne všechna rozlišení jsme schopni pokrýt ne-responzivním webem. Mobilní vyhledávače jsou ale schopny tento problém obejít a to díky tzv. virtuálnímu viewportu. Ten způsobí, že se se stránkou bude manipulovat jako s obrázkem. Je tedy možné s ním pohybovat, přibližovat či oddalovat. Prohlížeč se díky virtuálnímu viewportu tváří, že má jinou šířku než ve skutečnosti má[1]. Prohlížeč si stáhne stránku celou, kterou následně zmenší podle velikosti displeje. Je tedy zmenšena a uživatel si jí musí přizpůsobit. Pokud viewport není deklarován, používá se výchozí nastavení šířky 980 pixelů[9]. V responzivním webu není žádoucí, aby se uživatel musel přizpůsobovat stránce, když se stránka přizpůsobí uživateli. Proto se musí použit tzv. reálný viewport, který způsobí, že prohlížeč bude používat skutečnou (reálnou) šířku[2]. Skutečnou šířku displeje lze zjistit pomocí *device-width*[9]. Pomocí viewportu lze také zakázat či povolit přibližování stránek.

Mezi základní principy responzivního webu můžeme zařadit:

- flexibilní layout
- Media queries

3.1 Flexibilní layout

Flexibilní layout znamená, že všechny prvky na stránce jsou udány v relativních jednotkách např. v procentech. Nemají tedy pevně určenou šířku, ale jsou uvedeny v poměru vzhledem k rodičovskému elementu nebo šířce stránky. Převod fixních jednotek na procenta se provádí za pomocí následujícího vzorce[2]:

$$\frac{cíl}{kontext} = výsledek[\%]$$

Příklad přepočtu mezi fixním a flexibilním layoutem je uveden na obr.1

Flexibilita obrázků se řeší také pomocí procent ve stylu, ale je potřeba si uvědomit, že dojde ke stažení celého obrázku a poté se zobrazí v požadované velikosti. Pokud tedy potřebujeme obrázek s šířkou 150px, a původní obrázek má 1200px, stejně se stáhne původní obrázek s úplnou velikostí. Tomu lze zabránit mnoha způsoby, ale všechny metody potřebují mít daný obrázek ve všech použitelných formátech.

[1] proto virtuální viewport

```css
/* fixed-style.css */

#page {
  width: 1140px;
}

#left {
  width: 200px;
}

#right {
  width: 940px;
}
```

```css
/* responsive-style.css */
#page {
  /* kontext */
  width: 100%;
}
#left {
  /* cil */
  width: 17.54385965%;   /* 200 / 1140 */
}
#right {
  /* cil */
  width: 82.45614035%;   /* 940 / 1140 */
}
```

Obrázek 1: Ukázka přepočtu[2]

Vhodným prostředkem pro flexibilní layout je mřížka, protože její responzibilita je již vyřešena. Mřížka totiž řeší přetečení řádku, zalomení textu a uspořádání. Toto lze jednoduše vysvětlit na frameworku Twitter Bootstrap[2], který používá dvanáctisloupcovou mřížku. V této mřížce je celá zobrazovací plocha rozdělena na 12 sloupců a záleží pouze na nás, jak si danou stránku uspořádáme.

3.2 Media queries

Media queries je CSS3 modul, který umožňuje vykreslovat webové stránky podle různých parametrů např. rozlišení a velikost obrazovky[2, 7]. Máme například stránku obsahující formulář, který je stylizován pomocí flexibilního layoutu. Pokud použijeme zařízení s dostatečně malým displejem, tak bude tento formulář tak malý, že se stane nepoužitelným. Pomocí media queries lze definovat, jaké styly se mají použít pro dané rozlišení obrazovky, velikost obrazovky nebo dokonce orientaci displeje.

Podpora prohlížečů media queries není automatická, ale všechny nové prohlížeče ji podporují. IE je podporuje až od verze 9, nicméně pro verzi 8 lze použít java-scriptové řešení respond.js nebo media-queries.js. Podpora verze prohlížečů je znázorněna v tab. 1.

Prohlížeč	IE	Chrome	Opera	Firefox	Safari
Verze	9+	4+	9.5+	3.5+	3+

Tabulka 1: Podpora media queries

3.2.1 Syntax

Syntaxe media queries vychází z pravidel gramatiky CSS2. V CSS3 došlo k vylepšení o dotazy na vlastnosti média, na kterém jsou dané stránky zobrazovány[2, 7].

[2]viz http://getbootstrap.com/

Media queries můžeme použít dvěma způsoby. První způsob spočívá v tom, že vložíme media queries do HTML dokumentu na místě reference CSS souboru. Pokud bychom chtěli definovat styly pouze pro zařízení s maximální šířkou 480 pixelů, můžeme to provést takto:

```
<link rel="stylesheet" href="m.css" media="max-width:480px">
```

Druhým způsobem je vložení media do CSS souboru. Pro stejný příklad to lze udělat následujícím způsobem:

```
@media (max-width: 480px) {
  /*css kod, ktery se aplikuje jen po splneni podminek*/
}
```

Podmínky lze spojovat pomocí operátorů and, not a only. Jako podmínku lze použít např. min-width, max-width, max-height, device-width, color atd.

4 Architektura MVC

Architektura MVC dělí aplikaci do tří částí tak, aby je bylo možné upravovat samostatně. Dopad změn na ostatní části musí být co nejmenší[5]. Základní myšlenkou této architektury je oddělení aplikační logiky od zobrazovací vrstvy. Změna v jedné části poté nevyžaduje úpravu jiné.

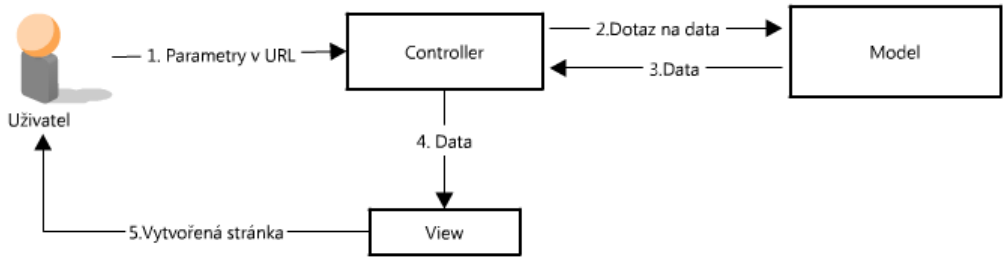
MVC architektura se skládá z:

- Model - model obsahuje logiku aplikace. V modelu se provádí veškerá důležitá činnost, včetně validace dat. Ta se může provádět i ve view, protože je to uživatelsky příjemné, ale v aplikační logice nesmí validace stejně chybět.
- View - se zabývá zobrazením dat uživateli. Například se jedná o HTML stránku s tagy značkovacího jazyka (např. Apache Wicket).
- Controler - se zabývá tokem dat v aplikaci. Propojuje model, view a uživatele.

4.1 Životní cyklus stránky

1. Životní cyklus stránky začíná uživatel, který zadá adresu webu a parametry, kterými nám sdělí, jakou stránku chce zobrazit
2. Controler přijme parametry, podle kterých ví, co se má dělat. Zavolá tedy příslušnou metodu modelu.
3. Metoda modelu data přijme, zpracuje je a vrátí požadované údaje.
4. Controler předá všechna vygenerovaná data view, který je přijme a vloží do šablony.
5. Hotová stránka je předána uživateli

Tento životní cyklus stránky můžeme vidět na obr.2



Obrázek 2: Životní cyklus stránky

5 Apache Wicket

Apache wicket je framework pro tvorbu webových aplikací v programovacím jazyce Java. Tento framework vznikl v roce 2004. Jeho první verze byla uvolněna o rok později. Autoři tohoto frameworku jsou Jonathan Locke a Miko Mastsumura[1].

5.1 Vlastnosti

Wicket se řadí, mezi komponentově řízené frameworky, které se vyznačují vysokou abstrakcí nad HTTP protokolem. U těchto typů framewokrů programátor nezpracovává přímo HTTP požadavky, ale reaguje na nějakou událost např. stisk tlačítka. Další výhodou komponentově řízených frameworků je to, že lze vytvářet znovupoužívatelné komponenty. Používání komponent má tu výhodu, že je lze využívat na více místech bez jakýchkoliv úprav. V případě změny komponenty se změny projeví na všech místech, kde je použita. Není tedy nutné měnit všechny stránky, kde se vyskytuje.

Představme si například stránky, které obsahují navigaci. Vytvoříme tedy komponentu, která implementuje danou navigaci. Následně ji vložíme do všech stránek, ve kterých je obsažena. Pokud se v budoucnu rozhodneme navigaci změnit, tak stačí změnu provést na jediném místě. Výhodou komponentového přístupu je také to, že existuje několik předpřipravených komponent, které je možno v aplikaci použít (např. validace vstupů, tabulky odkazy, tlačítka atd).

Webová aplikace psaná pod tímto frameworkem se skládá z prostého HTML pro prezenční vrstvu a Java kódu pro logiku. Toto oddělení usnadňuje vývoj, protože je od sebe oddělena funkční a grafická část kódu. Java kód lze přiřadit controler vrstvě a HTML view vrstvě z MVC architektury, viz bod 4. K provázání těchto dvou kódů se používají speciální wicket značky, jejichž princip lze demonstrovat na příkladu, který vytvoří stránku s výpisem „Ahoj“. Nejprve se vytvoří HTML soubor, který bude obsahovat HTML tag např. *span*. Do *span* vložíme *wicket:id=“label“*. Dalším krokem je vytvoření Java souboru, který musí mít stejný název jako zmíněný HTML soubor a i shodnou cestu. Tato třída bude dědit od *WebPage*. V konstruktoru této třídy přidáme štítek, který obsahuje dva parametry. První je hodnota id, druhým parametrem je to, co chceme vypsat. Zmíněný příklad tedy může vypadat takto:

MyPage.java

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

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

Wicket umožňuje integraci s jinými frameworky, kterými jsou například Spring, Ajax atd. Od verze 6.0 obsahuje přímou podporu JQuery. Což je javascriptová knihovna, která klade důraz na interakci mezi JavaScrtiptem a HTML. Podporuje tlačítko *zpět* v prohlížeči. Tato podpora je umožněná díky ukládání verzí stránek do mapy stránek pro každé okno prohlížeče.

5.1.1 Internacionalizace

Internacionalizace znamená přizpůsobení textu podle příslušné lokality (tzv. locale). Wicket k ní využívá zdroje. Jako zdroj je chápán soubor vlastností s příponou *.properties*, který je umístěn vedle třídy hlavní aplikační třídy. Existuje výchozí soubor, který se používá v případě, kdy žádný jiný soubor neodpovídá příslušné lokalitě. Tento výchozí soubor se jmenuje *WicketApplication.properties* a obsahuje použité zprávy. Kromě toho může aplikace obsahovat například soubor *WicketApplication_de.properties*, který se použije pouze při německých locale.

Wicket také umožňuje vytvářet internacionalizaci pro jednotlivé komponenty. Název toho souboru musí být opět shodný s Java i HTML souborem. Rozdíl bude jen v příponě *.properties*. Princip vytváření těchto souborů lze vidět na tomto příkladě. Properties soubor bude vypadat následovně:

```
value=label
label2.value=hodnota
form.label2.value=hodnota2
```

Pokud v příslušném Java soboru použijeme následující:

```
add(new Label("label",new ResourceModel("value")));
add(new Label("label2",new ResourceModel("value")));
```

Štítek s id *label2* zobrazí text *hodnota*, zatímco první štítek zobrazí výchozí hodnotu *label*. Za povšimnutí stojí id *ResourceModelu*. Jeho hodnota je důležitá vzhledem k properties souboru. Pokud bychom umístili druhý štítek do formuláře, zobrazí se hodnota *hodnota2*. Java kód by vypadal takto:

```
Form form = new Form("form");
form.add(new Label("label2",new ResourceModel("value")));
add(form);
```

5.2 Wicket značky

Základem celého wicketu je několik tzv. wicket značek, do kterých se snažíme dosadit komponenty. Tyto značky se vkládají do HTML kódu, který se nazývá markup.

5.2.1 Wicket Tags

- **wicket:link** - Jedná se o statický odkaz, který je definován nejen pro <a>, ale také pro <link>,<script>,<img> atd. Používá se např. takto:

- **wicket:panel** - Uvnitř tohoto tagu se definuje markup komponenty složené z jiných komponent (v Javě její implementace dědí od třídy Panel). Musí být v HTML souboru odpovídajícímu třídě definující danou komponentu

- **wicket:fragment** - je podobný wicket:panel, ale je deklarován v rodičovském markupu, a ne vždy musí být zobrazen. Hodí se pro případy, kdy komponenta není znovupoužitelná jinde a nechceme pro ni tedy vytvářet zvláštní panel. Ukázku použití wicket:fragmentu může vidět na následujícím příkladu.

  ```
  <body>
    <span wicket:id="panel1">panel</span>
    <span wicket:id="panel2">panel</span>
    <wicket:fragment wicket:id="frag1">1</wicket:fragment>
    <wicket:fragment wicket:id="frag2">2</wicket:fragment>
  </body>
  ```

  V markupu máme dva fragmenty, které budeme chtít vložit do *wicket:id = panel1* a *wicket:id = panel2*. Java kód bude tedy vypadat takto:

  ```
  public MyPage() {
    add(new Fragment("panel1", "frag1", MyPage.this));
    add(new Fragment("panel2", "frag2", MyPage.this));
  }
  ```

- **wicket:extends** -Je to tag, který rozšíří markup nadřazené třídy svým obsahem. Použití tohoto tagu je demonstrováno na obr.3

- **wicket:child** - používá se u oddděděných komponent. Jak se tento tag používá je vidět na obr.3

Rodičovský markup	Markup potomka	Výsledný markup
<body> super markup <wicket:child /> </body>	<body> child markup <wicket:extend> in child markup </wicket:extend> </body>	<body> super markup <wicket:child><wicket:extend> in child markup </wicket:child></wicket:extend> </body>

Obrázek 3: Dědičnost

5.2.2 Wicket Atributy

- **wicket:id** - je to atribut, který identifikuje daný tag. Používá se na každém elementu, do kterého chceme přidat některou z komponent. Odpovídá id u komponenty v Java kódu.
- **wicket:message** - jedná se o atribut, který použijeme v případě, že na daný element chceme dát hodnotu, která je získána ze zdrojů. Používá se především pro lokalizaci, viz bod 5.1.1.
- **wicket:enclosure** - Jedná se o atribut, který umožní nastavit viditelnost širšího kusu stránky v závislosti na viditelnosti konkrétní komponenty. Mějme následující markup:

  ```
  <wicket:enclosure>
    <label>Celkem:</label><span wicket:id="celk"></span>
  </wicket:enclosure>
  ```

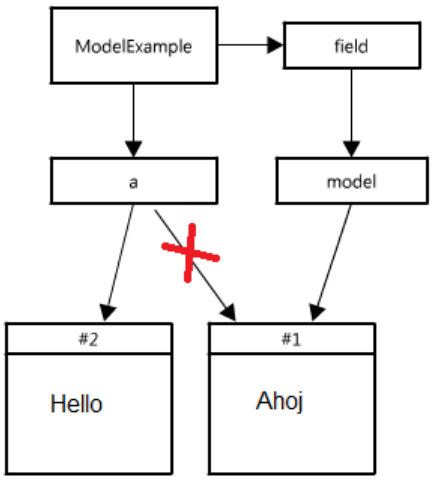
  Tento markup způsobí to, že jestliže není komponenta s id *celk* viditelná, zůstane skrytý i štítek s příslušným textem.

5.3 Modely

Model je jakákoliv implementace rozhraní *IModel*. Toto rozhraní obsahuje metody *getObject* a *setObject*. Model udržuje referenci na potřebná data a umožňuje komponentám s nimi pracovat. Pokud komponenta je vykreslována, potom je volána metoda *getObject()*, která má návratovou hodnotu na daný objekt.

Základní princip modelů bude demonstrován na následujícím příkladu

```
public class ModelExample extends Panel{
  String a;
  TextField field;
  public ModelExample(String id){
    super(id);
```



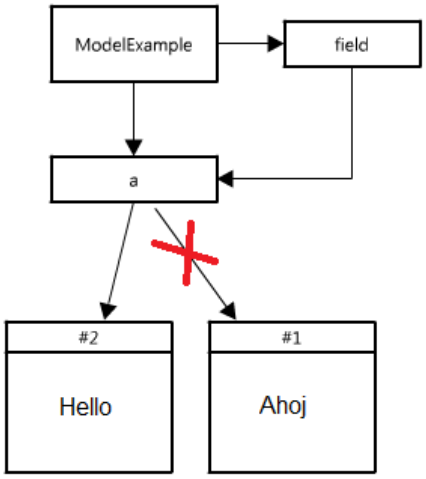
Obrázek 4: Příklad na modely

```
        ...
        field=new TextField("id",new Model(a));
  }
}
```

V prvotním okamžiku není v tomto použití zřejmý žádný problém. Ten však nastává v okamžiku, kdy se do proměnné *a* uloží jiný objekt. Model totiž neustále udržuje referenci na předchozí místo v paměti. To bude mít za následek to, že *field* bude zobrazovat zastaralou hodnotu *a* obr. 4. Tento problém lze vyřešit velmi jednoduše a to následovně:

```
public class ModelExample extends Panel{
  IModel<String> a;
  TextField field;
  public ModelExample(String id){
    super(id);
        ...
        field=new TextField("id",a);
  }
}
```

Změna hodnoty řetězce, který chceme zobrazit se bude provádět pomocí metody *setObject*. Jelikož si komponenta drží referenci na model, nikoliv jeho hodnotu, získá vždy aktuální data (obr. 5).



Obrázek 5: Správné použití modelu

- **Model** - nejjednodušší model, který slouží k uchování statických dat. Nemá žádné speciální vlastnosti.
- **PropertyModel** - Model, který svou hodnotu získává ze zadaného objektu pomocí řetězce s cestou k libovolnému atributu. Jeho použití lze pochopit na příkladu[6][3]:

```
public class Person{
  private String name;
  public String getName(){
        return name;
  }
  public void setName(String name){
        this.name = name;
  }
}
```

Úkolem bude vytvořit štítek, který vypíše jméno aktuální osoby. Druhým úkolem, bude vytvořit textové pole, jehož hodnota bude aktualizovat jméno osoby. Řešení bude tedy následující:

```
Person person = getPerson();
add(new Label("id", new PropertyModel(person,"name")));
add(new TextField("id2", new PropertyModel
```

[3]Všimneme si, že property model standardně poskytuje přístup k privátním atributům a metodám [1]

- **CompoundPropertyModel** - jedná se o zvláštní druh modelu, který se obvykle používá při dědění. V případě, že komponentě nebyl přidělen model, ale ona ho potřebuje použít, bude hledat model ve svých rodičích. Model, který hledá, musí implementovat rozhraní IComponentInheritedModel. Jakmile je CompoundPropertyModel získán, chová se stejně jako PropertyModel

- **LoadableDetachableModel** - jedná se o abstraktní model, ke kterému je třeba dopsat metodu load(). Tato metoda slouží k zisku dat, které se následně předají dané komponentě.

6 Twitter Bootstrap

Bootstrap je oblíbený HTML, CSS a javasriptový framework, který slouží pro tvorbu webových aplikacích. Díky Bootstrapu je vývoj webových aplikací rychlejší a jednodušší [11]. Pro použití tohoto frameworku jsou potřeba pouze základní znalosti o HTML a CSS. Je vyroben pro vývojáře všech úrovní, pro zařízení všech tvarů a pro projekty všech velikostí[11]. Vyvinuli ho Mark Otti a Jacob Thorton ve firmě Twitter jako framework pro podporu konzistence interních souborů[11]. V jejich firmě byly pro vývoj rozhraní používány různé knihovny, což vedlo k nekonzistenci kódů a vysokým nákladům[11]. V roce 2011 vydali jako open-source[4].

6.1 Vlastnosti

Bootstrap obsahuje kompilované CSS a JS soubory, které lze použít v kterémkoliv projektu. Součástí bootstrapu jsou i tyto soubory v minimalistické verzi. V případě potřeby využití tohoto frameworku je nutné vložit tyto soubory do hlavičky HTML dokumentu. Použití frameworku spočívá ve správném vnořování CSS tříd do HTML dokumentu.

Mezi výhody patří jednoduchost použití jednotlivých komponent, možnost tvorby responzivního vzhledu a široká podpora prohlížečů. Bootstrap podporuje tyto prohlížeče:

- Chrome
- Firefox
- Opera
- Internet Explorer od verze 8.0. Pro verzi 8.0 je ale nutné použít Respond.js.

Bootstrap poskytuje moderní a jednotný vzhled pro formátování základních HTML elementů. Toho dosahuje poskytováním sady stylopisů, které definují základní funkce stylů. Kromě HTML elementů bootstrap poskytuje další běžně používané prvky uživatelské rozhraní jako jsou například seskupování tlačítek, tlačítka s rozbalovací nabídkou, navigace, štítky atp. Tyto komponenty jsou implementovány pomocí CSS tříd, které musí být použity na určitém HTML elementu.

Bootstrap také poskytuje některé javascriptové komponenty, které jsou poskytovány ve formě jQuery pluginů. Tyto pluginy také rozšiřují funkčnost stávajících elementů například o autodoplňování a validaci vstupů atp.

Mezi nevýhody lze zařadit to, že v důsledku přesného dodržování vnořování HTML kódu může daný kód narůst.

6.2 Grid layout

Grid layout se používají pro rozvrhování naších stránek prostřednictvím sloupců a řádků. Bootstrap mřížka se skládá ze dvanácti sloupců. Tyto sloupečky mají na každém zařízení definovánu jinou velikost. V případě, že nějaký řádek obsahuje více sloupců, je tento řádek zalomen.

Základní principy grid layoutu v Bootstrapu.

1. Každý řádek stránky je umístěn ve třídě *row*

2. Každý řádek lze libovlně rozdělit až do 12 sloupců. Název tříd pro jednotlivá zařízení je uveden v tab.2.

Zařízení	Extra malá	Malá	Střední	Velká zařízení
Šířka displeje	<768px	≥768px	≥992px	≥1200px
Třída	.col-xs-*[4]	.col-sm-*	.col-md-*	.col-lg-*
Počet sloupců	12			

Tabulka 2: Třídy pro jednotlivá zařízení

3. Pokud zadáme více jak 12 sloupců, budou přebývající sloupce umístěny na nový řádek.

4. Odsazení sloupců lze provést např. třídou *.col-md-offset-3*

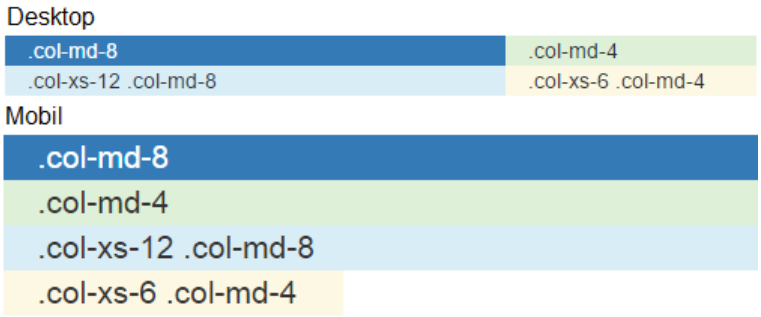
Tyto principy si můžeme ukázat na následujícím příkladu, který je na obr. 6. Na obr.7 lze vidět výstup tohoto příkladu na desktopu.

```
<div class="row">
 <div class="col-md-8">.col-md-8</div>
 <div class="col-md-4">.col-md-4</div>
</div>
<div class="row">
 <div class="col-xs-12 col-md-8">.col-xs-12 .col-md-8</div>
 <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
</div>
```

Obrázek 6: Používání sloupců v Bootstrapu

1. Řádek je rozdělen do dvou sloupců, přičemž první z nich zabírá 8 sloupců a druhý 4.

2. Řádek je rozdělen stejně jako v prvním případě. Ovšem pro velmi malá zařízení se první sloupec zobrazí přes všech 12 sloupců. Druhý sloupec zabere sloupců 6, ale bude zobrazen na novém řádku. To z důvodu přetečení.



Obrázek 7: Grid layout v Bootstrapu

7 Knihovna Wicket:Bootstrap

Jedná se o knihovnu, která se snaží rozšířit Wicket komponentami využívající Twitter Bootstrap. Výhodou této knihovny je to, že programátor nemusí psát příslušné komponenty sám. U jednoduchých komponent je vhodné použití knihovny zvážit z důvodu porušení principu MVC architektury. Knihovna totiž přidává CSS třídy z Java kódu. Míchá tedy view a controler vrstvu dohromady. Pro složitější komponenty jako je např. menu se přesto použití knihovny vyplatí, jelikož není třeba je programovat od píky.

Pro použití této knihovny je potřeba přidat požadováné závislosti do maven projektu.

8 Implementace

Celá aplikace je rozdělena do dvou modulů. Modul *membernet* obsahuje vlastní aplikaci - tzn. jednotlivá view. Aplikace obsahuje 37 stránek, které obsahují přes 137 komponent. Modul *yosocommon* obsahuje obecné komponenty, které lze využít i jinde než v projektu Jasenverkko. Těchto komponent je celkem 75.

8.1 Příprava aplikace

Nejprve bylo třeba odstranit načítání původních CSS souborů ze všech komponent. Načítání souborů lze ve Wicketu provádět několika způsoby. Prvním způsobem je vložení souborů pomocí hlavní třídy aplikace. Tato třída se jmenuje *WicketApplication.java*. V této třídě bylo potřeba odstranit metodu na vkládání CSS souborů. Druhý způsob umožňuje CSS soubory vkládat i z konkrétních komponent pomocí přetížení metody *renderHead*. Bylo proto potřeba projít všechny komponenty a hledat přetížení těchto metod.

Dále bylo potřeba přidat knihovnu wicket:bootstrap. Do hlavní třídy aplikace byl vložen následující úsek kódu.

```
BootstrapSettings settings = new BootstrapSettings();
/* here is possible to set some theme */
Bootstrap.install(this, settings);
```

Od tohoto okamžiku muže být použita nejen tato knihovna, ale i samotný bootstrap. To z toho důvodu, že předchozí kód automaticky vloží potřebné soubory do hlavičky markupu.

Posledním krokem příprav bylo vytvoření skriptu. Tento skript automaticky prošel všechny HTML soubory a odstranil původní CSS třídy a identifikátory. Jelikož je ale možné CSS třídy přidávat i v Java části, bylo nutné projít i jednotlivé třídy.

8.2 Úprava abstraktních komponent

Abstraktní komponenty udávají hlavní (globální) vzhled stránek a jejich jednotlivých sekcí (panelů). Abstraktní komponenty se vyplatí používat tehdy, jestliže se v aplikaci objevují prvky se stejnými vlastnostmi. Například v případě, kdy aplikace obsahuje několik druhů panelů se stejnou hlavičkou. Výhoda jejich použití je v tom, že tyto společné vlastnosti lze měnit na jednom místě.

8.2.1 AbstractPage

AbstractPage je stránka, která definuje základní rozdělení všech ostatních stránek. Stránka je rozdělena na hlavičku, hlavní panel a patičku. Hlavička obsahuje menu a skupinu tlačítek. Patička se skládá ze tří tlačítek. Hlavní panel je složen z varovné komponenty, feedbacku a obsahu stránky. Obsah stránky je již závislý na určité stránce.

Postup úpravy této stránky byl následující:

1. Minimalizace markupu - z markupu byly odstraněny všechny HTML elementy, které neobsahovaly wicket značky.

2. Úprava hlavičky - Jako menu byla použita komponenta navbar z knihovny wicket:bootstrap. Pokud je navbar zobrazen na malém zařízení, tak se odkazy automaticky schovají pod rozklikávací tlačítko. Druhou upravovanou komponentou byla skupina tlačítek. Bohužel tlačítko a menu se zobrazují nezávisle na sobě (navbar není zobrazen na každé stránce). Proto nelze obě komponenty sloučit do jedné. Implementace skupiny tlačítek probíhala správným vnořením CSS tříd do HTML kódu.

3. Úprava patičky - Úprava tlačítek spočívala v úpravě markupu. Tlačítka byla umístěna do jednoho řádku. Na malém zařízení byla jednotlivá tlačítka umístěna vždy na nový řádek.

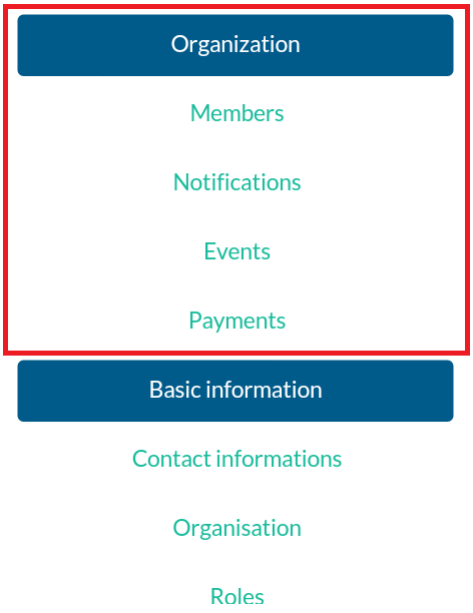
4. Úprava varování - Toto varování slouží k výpisu varovných zpráv. Zobrazuje se například, pokud účet uživatele není aktivován. Celé toto varování bylo roztaženo přes celý řádek.

5. Úprava globálního feedbacku - Feedback dává zpětnou vazbu uživateli o jeho akcích. Zobrazuje potvrzující, zamítavé nebo informativní hlášky. Jeho detailní úprava je popsána v kapitole 8.3.4.

Nejdůležitější wicket značkou v tomto markupu je <*wicket:child*>. Na toto místo se vloží markup, který implementuje potomek. Tento markup musí být umístěn v tagu <*wicket:extend*>, viz bod 5.2.1. Každá stránka v aplikaci se liší pouze v tomto bodě.

8.2.2 AbstractManagerPage

Tato stránka slouží jako šablona pro stránky, které jsou v administrativní části aplikace. Její rodičem je *AbstractPage*, proto obsahuje markup, který byl zmíněn v bodu 8.2.1. Navíc implementuje panel, který se zobrazí v případě, že daná organizace není aktivní. Tento panel byl vložen na řádek o velikosti 12 sloupců.



Obrázek 8: Dvě menu na mobilu

Dále vytváří administrační menu, které slouží pro pohyb v administrační části aplikace. V původní verzi aplikace se objevují pod tímto menu ještě další podmenu. Na desktopech problém nenastane, nicméně na malých zařízeních ano. Obě dvě menu se totiž roztáhnou pod sebe. Tato situace je znázorněna na obr. 8[5]. Z obrázku je vidět, že se celá navigace stala nepřehlednou. Proto bylo vhodnější schovat první menu pod rozklikávací tlačítko. Menu tedy funguje tak, že na desktopech je zobrazeno klasicky, ale na mobilu se skryje pod tlačítko. Když se na toto tlačítko klikne, tak se zobrazí nabídka odkazů. Z této nabídky si potom uživatel vybere, kam chce přesměrovat. Jedná se vlastně o navbar. Tento navbar by šlo opět vytvořit pomocí knihovny wicket:bootstrap. Nicméně v tomto případě bude vhodnější upravit pouze markup. To z důvodu pozdějšího odlišného stylování oproti hlavnímu navbaru. Původní markup této navigace byl:

```
<ul>
 <li class="nav-link nav-link-bottom" wicket:id="links">
  <a wicket:id="link">
        <wicket:container wicket:id="label"/>
  </a>
 </li>
</ul>
```

Bootstrap požaduje k *ul* přidat třídy *nav navbar-nav*. Dále potřebuje následné umístění celého seznamu do divu obsahující třídy *collapse navbar-collapse*. Následně se do tohoto divu vloží identifikátor, který musí být jedinečný např. *adminMenu*. Dále

[5]V červeném rámečku je zvýrazněno hlavní administrační menu.

je nutno vytvořit navbar hlavičku, která bude obsahovat třídu *navbar-header.* Tato třída bude obsahovat tlačítko, které po jeho stisku zobrazí dané linky. Nakonec se hlavička i seznam odkazů vloží do tagu *nav*, ke kterému se přidají třídy *navbar* a *navbar-inverse*. Výsledný markup bude vypadat následovně:

```
<nav class="navbar navbar-inverse">
 <div class="navbar-header">
  <button type="button" class="navbar-toggle collapsed"
          data-toggle="collapse" data-target="#adminMenu">
    ...
  </button>
 <div class="collapse navbar-collapse" id="adminMenu">
  <ul class="nav navbar-nav">
   <li role="presentation" wicket:id="links">
    <a wicket:id="link">
       <wicket:container wicket:id="label"/>
    </a>
   </li>
  </ul>
 </div>
</nav>
```

8.2.3 AbstractPanel

Tento panel poskytuje základní vzhled všem panelům, které jsou v aplikaci použity. Obsahuje hlavičku, ve které je zobrazen nadpis. Tělo panelu je implementováno v potomkovi této třídy a obsahuje například formulář pro registraci, přihlášení nebo vyhledávání atd. Úprava markupu tohoto panelu bude prostá. Použijeme opět třídy pro panel, které nám poskytuje Bootstrap.

Panel má několik možností použití. V některém případě je roztaženo přes celý zobrazovací prostor (např. registrační formulář). V druhém případě je roztažen jen přes určitou velikost. To z toho důvodu, že vedle něj se nachází panel, který obsahuje seznam odkazů. Tato vlastnost je vyřešena v controler vrstvě. Třída *AbstractPanel.java* obsahuje atribut *isTabPanel*, který je typu boolean. Hodnoty *false* nabývá v případě, že je panel roztažen přes celý prostor. Dle tohoto atributu nastavíme této komponentě požadované třídy. K tomu se využívá následující metoda, která se volá při inicializaci komponenty.

```
private AttributeModifier getMainContAttr(){
 if(isTabPanel()){
  return AttributeModifier.append
```

```
                ("class", "col-xs-12 col-sm-9 col-md-9");
  } else {
   return AttributeModifier.append("class", "");
  }
}
```

Pokud panel má být roztažen přes celý prostor, tak není potřeba přidávat další třídy. V opačném případě je třeba vložit panel do určitého počtu sloupců. Na mobilu je vhodné panel umístit přes všech dvanáct sloupců z důvodu malé šířky displeje.

8.3 Úprava dynamických komponent

Pod pojmem dynamická komponenta si lze představit komponentu, která se vykresluje v závislosti na příchozích datech. Důležitost těchto komponent spočívá ve znovupoužitelnosti. Všechny dynamické komponenty jsou umístěny v balíku *yosocommon*.

8.3.1 DynamicFormContent

Jedná se o panel, který generuje markup formuláře. Komponenty se do něj přidávají pomocí metod *addField*, *addGroup*, *addDisplayField* nebo *addDisplayLink*. Tyto komponenty jsou následně vykreslovány pomocí speciální komponenty (repeater), která umožňuje do jedné wicket značky vložit více komponent. U této dynamické komponenty není třeba měnit controler vrstvu. Markup se změní tak, aby využíval Bootstrap a zároveň byl i responzivní. Z původního markupu můžeme vidět, že jeden řádek formuláře obsahuje maximálně tři položky. Je tedy nutné určit správné rozdělení do sloupců. V rámci testování bylo zjištěno, že nejpřehlednější se jeví následující rozdělení:

```
<div class="form-group">
  <label class="col-xs-12 col-sm-6 col-md-3 control-label"
                wicket:id="label"></label>
  <div class="col-xs-12 col-sm-6 col-md-5 ">
   <wicket:container wicket:id="input"></wicket:container>
  </div>
 <div class="col-xs-12 col-sm-12 col-md-4" wicket:id="feed">
 </div>
</div>
```

Tato ukázka kódu představuje jeden fragment se třemi položkami. Všechny ostatní fragmenty musí mít stejné rozložení do sloupců. To z toho důvodu, že tato kom-

ponenta může obsahovat více fragmentů různých druhů. Tímto dosáhneme stejného zarovnání bez ohledu na typ fragmentu. Důležité je umístit celý fragment do třídy *form-group*, která říká, že se jedná o nový řádek formuláře. Je ale nutné dát si pozor na správné vnořování. Aby totiž tato třída správně řádkovala, musí být umístěna ve třídě *form-horizontal*. Všem formulářům, ve kterých je tato dynamická komponenta použita, se musela doplnit zmíněnou třídou.

Třída *control-label* formátuje štítek tak, aby byl zarovnán doprava se stejným zarovnáním jako vstupní pole. Při testování bylo zjištěno, že u fragmentu obsahující 2 štítky vedle sebe, je příliš velká mezera mezi řádky. Tento problém způsobovalo použití této třídy. Pro tento typ fragmentu byla vytvořena nová CSS třída, viz bod 9.1.1.

8.3.2 TabbedPanel

TabbedPanel reprezentuje panel se záložkami. Tyto záložky slouží k přepínání mezi různými obsahy uvnitř tohoto panelu. Je implementován Apache Wicket. Nicméně bylo vhodnější použít wicket:bootstrap verzi. To z důvodu ulehčení následného stylování. Proto všude, kde se tato třída v controler vrstvě vyskytovala, byla nahrazena třídou *BootstrapTabbedPanel*. Z testování vyplynulo (kapitola 9.2.4), že je lepší vložit ještě třídu *nav-justified*. Ta způsobí to, že všechny záložky se roztáhnou přes celou šířku. To tak, že velikost každé záložky bude stejná. Na malých zařízeních navíc způsobí, že se záložky zobrazí pod sebe. Tato třídu se vloží pomocí přepsání metody *getTabContainerCssClass*. Všechny tyto panely budou tedy vytvářeny tímto způsobem:

```
new BootstrapTabbedPanel<ITab>("tabs", tabs){
 @Override
 protected String getTabContainerCssClass(){
  return "nav nav-tabs nav-justified";
 }
}
```

8.3.3 Drobečková navigace

Drobečková navigace slouží pro zobrazení zanoření v hierarchii stránek. Dále poskytuje rozbalovací nabídku pro položky na stejné úrovni. V souboru *BreadcrumbsPanel.java* se dříve vkládali CSS soubory a javascripté programy, které dodávaly navigaci vzhled a funkčnost. Tyto řádky byly odstraněny, protože byl použit Bootstrap. V dalším kroku bylo potřeba upravit makup tak, aby odpovídal Bootstrapu. K daným tagům se přidaly požadované třídy.

Součástí navigace je i rozbalovací tlačítko, které je implementováno třídou *BreadcrumbsDropDownItem.java*. Tento soubor nebylo potřeba nějak upravovat. Měnil se bude pouze markup a to tak, aby odpovídal rozbalovacímu tlačítku z Bootstrapu.

8.3.4 Feedback panel

Feedback panel slouží k zobrazení zpráv. Dává zpětnou vazbu uživateli v tom, jestli se určitá akce zdařila nebo ne. Používá se například u přihlášení. Pokud uživatel zadá špatné jméno nebo heslo, je vypsána chybová zpráva.

Pro tento panel je možné opět použít wicket:bootstrap knihovnu, které ho implementuje ve třídě *NotificationPanel*. Úprava těchto komponent probíhala následovně:

1. Nalezení třídy *GlobalFeedbackPanel*
2. Úprava této třídy tak, aby dědila od *NotificationPanel*
3. Nalezení třídy *FilteredFeedback*. Upravit třídu dle bodu 2.
4. Ve třídě *FilteredFeedback* odstranit metodu *newMessageDisplayComponent*. Tato metoda je již obsažena v *NotificationPanel*, kde pracuje dle našich představ.

8.4 Úprava statických komponent

Pod statickou komponentou si můžeme představit komponentu, která je za všech okolností stejná. Jedná se např. o registrační, vyhledávací a přihlašovací formuláře, informační panely členů atd. Úprava těchto komponent spočívá v úpravě markupu tak, aby vše odpovídalo responzivnímu webu, využívalo bootstrapu a rozložení stránek se co nejvíce přibližovalo původním stránkám. U každé komponenty je potřeba zkontrolovat i controler vrstvu. To především kvůli tomu, že nechceme aby se do aplikace přidávaly jiné CSS soubory než z Bootstrapu.

8.5 Vyskakovací okna

Jedná se o okna, která se zobrazí po nějaké akci např. kliknutí na tlačítko. Zmizí opět po vyvolání nějaké akce např. kliknutí na křížek, tlačítko atd.

8.5.1 Původní řešení

Původní řešení bylo provedeno pomocí Apache Wicket, který implementuje vlastní vyskakovací okno. To se implementuje ve třídě *ModalWindow*. Funguje tak, že se oknu přiřadí obsah, který se má vykreslit. Oknu lze přiřadit určité vlastnosti. Podle nich rozdělujeme okna v aplikaci na:

1. potvrzovací okno - okno, které potvrzuje nějakou akci. Obsahuje tlačítka ano a ne. Lze zavřít pouze těmito tlačítky.

2. editovatelné okno - okno, ve kterém lze měnit údaje. Obsahuje tlačítka uložit a storno.

3. fixní okno - okno, které má pevné rozměry. V responzivním webu není přijatelné. Musí být nahrazeno.

4. informativní okno - okno, které má pouze informativní účely.

8.5.2 Řešení pomocí wicket:bootstrap

K vyřešení problematiky responzivních vyskakovacích oken je vhodné použít knihovnu wicket:bootstrap. A to konkrétně panel *Modal*, který řeší vyskakovací okna pomocí javascriptu. Důležité je si uvědomit, že z pohledu Wicketu se jedná o panel, nikoliv o *ModalWindow*. Proto je nutné vytvořit novou komponentu s názvem *ModalPanel*. Markup tohoto panelu bude definovat tělo modálního panelu. Hlavička a patička tohoto panelu je implementována v rodičovské třídě. Markup vypadá následovně:

```
<wicket:extend>
<div wicket:id="content"></div>
</wicket:extend>
```

Třída *ModalPanel* dědí od *Modal*. Tato třída obsahuje 2 důležité metody. První z nich je metoda *setContent*, jejíž úkolem je nastavit obsah modálního okna. Tato metoda nejprve zkontroluje zda předávaný kontejner má id *content*. Pokud ano, tak je přidán do modálního okna. Metoda vypadá následovně:

```
public Modal<T> setContent(final WebMarkupContainer con){
 if(!con.getId().equals(getContentId())){
        throw new WicketRuntimeException
            ("Modal content id is wrong.");
 }
 content.setOutputMarkupId(true);
 addOrReplace(con);
 return this;
}
```

Druhou metodou je *show*, která zobrazuje modální okno. Tato metoda sice ve třídě Modal implementována je, ale nevyhovuje požadované funkčnosti. To z toho důvodu, že pouze zavolá metodu javascriptu. Naším požadavkem je, aby metoda znova překreslila původní okno. Pokud by se tohle neudělalo, v okně by zůstala

stará data. Respektive data, která v něm byla po uzavření okna. K tomuto důvodu je ve Wicketu implementována třída *AjaxRequestTarget*. Tato třída slouží k označení komponent, které mají být aktualizovány při ajaxovém volání. Pomocí metody *add* se musí modální okno označit, aby bylo při ajaxovém volání aktualizováno. Pomocí této třídy lze k ajaxovému volání připojit javascript. Metoda vypadá takto:

```
public Modal<T> show(final AjaxRequestTarget target){
  target.add(this);
  super.show(target);
  return this;
}
```

Tento panel tvoří základ všech ostatních vyskakovacích oken. Tato implementace vytváří informační okno. Editovatelné okno je implementováno v abstraktní třídě *ModalEditPanel*. Potomek této třídy musí dopsat metodu *setEditObject*, která slouží k načtení existujících dat dovnitř okna pro editaci.

Potvrzovací okno je vytvořeno ve třídě *ConfirmationModalPanel*. Jedná se o abstraktní třídu, ke které je třeba dopsat metody *onYes* a *onNo*. Tyto metody jsou volány v případě, že se klikne na příslušné tlačítko. Určují tedy, co se má po kliknutí na tlačítko stát. V této třídě je také vytvořen formulář, který obsahuje štítek pro vypsání zprávy a dvě tlačítka.

Fixní okno už není používáno z důvodu pevných jednotek. Dle jeho původních vlastností bylo nahrazeno informačním nebo editovatelným oknem.

8.5.3 Problémy

Během testování byl zjištěn problém se zavíráním okna. Ten nastává v případě, kdy máme editovací nebo potvrzovací okno umístěno ve formuláři. Pokud se po zavření modálního okna aktualizuje formulář pomocí technologie AJAX, nastane problém. Okno se sice zavře, ale ve stránce zůstane průhledné pozadí. Toho se lze zbavit pouze znovunačtením stránky.

K tomuto problému dochází, protože obslužné rutiny AJAXu odstraní okno a následně ho nahradí. Důsledkem toho Bootstrap nebude moci najít referenci na příslušné okno.[2]. Proto nebudou dokončeny všechny javascripty.

Možné řešení by mohlo být v tom, že se modální okno umístí mimo formulář. Bohužel tato možnost není vždy možná. Problém by totiž nastával v editovacích oknech, které je potřeba také aktualizovat. Při první změně problém nenastane a okno se zavře dle očekávání. Problém nastane, pokud se rozhodneme otevřít okno znovu bez jakékoliv aktualizace stránky. Okno totiž nebylo aktualizováno, a tak načítá stará data. Aplikace je proti tomuto ošetřena a upozorní nás varovnou

hláškou a k zobrazení okna nedojde.

Další řešením bylo přepsání metody *close* ve třídě *ModalPanel*. Zápis bude vypadat následovně:

```
public Modal close(AjaxRequestTarget target){
 super.close(target);
 target.appendJavaScript("$('body').removeClass
                          ('modal-open');");
 target.appendJavaScript("$('.modal-backdrop').remove();");
 return this;
}
```

Nejprve se zavolá metoda z nadřazené třídy, která zavře okno. Další řádek odstraní z *tagu* body třídu *model-open*. V následujícím kroku se odstraní pozadí okna. Výsledkem je správné zavření okna, a to i v případě aktualizace okna Ajaxem. Vzhledem k bezproblémovému zavření okna bylo zvoleno toto řešení.

8.6 Stylování

Stylování webových stránek znamená vytváření CSS stylů, které se následně vkládají do HTML pomocí identifikátorů nebo tříd. Těmito styly je možné stránkám nastavit vzhled. Vzhledem k požadavkům zadavatele není třeba stylovat stránky do původního vzhledu. Je třeba zachovat původní barevné schéma a rozložení stránek (rozumí se ergonomie).

8.6.1 Bootstrap

Bootstrap na svých stránkách umožňuje vytvořit si vlastní variantu Bootstrap souborů. Je zde možné si vybrat, které komponenty chceme využít, a jak je chceme nastylovat. Vzhledem k malým požadavkům na vzhled bude lepší použít šablonu, ve které se změní pouze potřebné vlastnosti určitých prvků. Pro získání šablony lze použít Bootswatch[6]. Po stáhnutí šablony budeme pokračovat tak, že si najdeme prvek ze stránky v souboru *bootstrap.css*, ve kterém následně změníme barvu tohoto prvku.

8.6.2 Vlastní styly

Na základě testování, viz sekce 9, bylo potřeba vytvořit několik vlastních stylů:

1. *text-right* - tento styl zarovná text doprava. Na malých zařízení je potřeba tento text zarovnat doleva, viz kapitola 9.2.1

[6]Jedná se o stránky poskytující Bootstrap šablony viz https://bootswatch.com/

2. *panel-noBorder* - tento styl ruší na malých zařízení ohraničení a odsazení panelu,
3. *fieldset* - bootstrap ruší výchozí stylování. Musí být proto znovu nastaveno.

9 Testování

V rámci testování se bude hodnotit responzibilita webu, přehlednost, rozložení stránek a v neposlední řadě zachování funkčnosti aplikace. Testování bude probíhat na desktopu s rozlišením 1366 x 768 a na mobilním zařízení HTC Desire 500. Dále otestujeme funkčnost webu na prohlížečích Chrome verze 42, Firefox 37, IE 11 a Opera 29.0.

Bohužel nebylo možné otestovat úplně všechny komponenty. Některé části aplikace se totiž nepodařilo zobrazit. Jedná se například pro stránky aktivace uživatele. Když chce být uživatel aktivován, je mu odeslán email s odkazem, který ho přesměruje na požadované stránky. V prototypu této aplikace toto odesílání emailu nefunguje. Dále nelze otestovat vše, co souvisí s platebním stykem. Prototyp totiž při stisku tlačítka, které má přesměrovat na stránky s výběrem banky, hlásí chybu. Tato chyba byla už v původní aplikaci. Tyto komponenty byly tedy implementovány naslepo, bez jakéhokoliv testování. Nicméně díky komponentovému frameworku byly konkrétní použité komponenty otestovány jinde. Je pravděpodobné, že budou i na těchto konkrétních stránkách fungovat v pořádku.

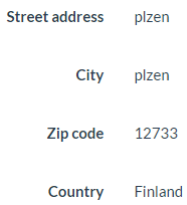
9.1 Desktop

Přehlednost webu na tomto zařízení je dostačující. Rozložení stránek odpovídá původní aplikaci. Všechny prvky se zobrazují dle očekávání. Funkčnost stránek je změněna pouze v případě navigace a rozklikávajícího tlačítka. Změna nastala v tom, že se dříve nabídka zobrazila po najetí myší na příslušné tlačítko. Nyní toto bylo nahrazeno kliknutím. Tato změna je nutná, protože uživatel není schopen na dotykových zařízeních simulovat najetí kurzorem. Proto bylo zvoleno kliknutí.

Testování funkčnosti webu na aktuálních verzí uvedených prohlížečů proběhlo bezproblémově. Všechny komponenty plnily svoji funkci dle očekávání.

9.1.1 Nalezené nesrovnalosti

Během testování byl nalezen problém v přehlednosti, viz obr. 9. Problémem je příliš velká mezera mezi jednotlivými řádky. Ta způsobí, že formulář je zbytečně velký a uživatel se bude hůře orientovat. Tento problém se vyskytuje jen v případech, kdy řádek formuláře neobsahuje vstupní pole např. obsahuje 2 štítky pro výpis textu.



Obrázek 9: Nesrovnalost v řádkování

Tento problém způsobovalo použití Bootstrap tříd *control-label* a *form-control-static*. Tyto třídy udělají mezeru mezi řádky tak velkou, aby se do řádku pohodlně vešlo vstupní pole. Tyto třídy musely být u vhodných fragmentů odstraněny. Odstraněním těchto tříd došlo k dalšímu problému. Problém byl v tom, že první sloupeček nebyl zarovnán doprava. Ke správnému zarovnání použijeme třídu *text-right*.

Dále byly nalezeny problémy, které se týkaly správného zarovnání. Tento problém si lze představit například tak, že navigace měla větší odsazení než navbar. Problém byl vyřešen správným použitím tříd pro bootstrap layout, viz sekce 6.2.

9.2 Na mobilu

Testování proběhlo na mobilu s operačním systémem Android verze 4.1.2 s obrazovkou velkou 4,3 palce. Bylo testováno pouze na prohlížeči Chrome.

9.2.1 Problém s řádky formuláře

Tento problém souvisí s problémem nalezeném na desktopu v kapitole 9.1.1. Vyřešení problému na desktopu způsobilo rozhození vzhledu na mobilu. Na mobilu by měly být oba štítky vloženy pod sebe a oba zarovnány doleva. První z nich je ale zarovnán doprava, viz obr. 10. Proto se musí styl *text-right* na malých zařízení zarovnat doleva. To se provede pomocí media queries a to následovně:

Obrázek 10: Chyba zarovnání na mobilu

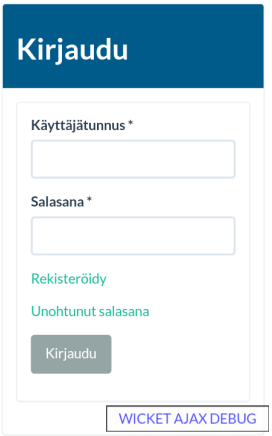
```
@media (max-width: 768px){
 /* In small devices need text align left*/
 .text-right{text-align: left;}
}
```

Tento kód říká, že jestliže je maximální šířka displeje 768px, použijí se následující styly.

9.2.2 Problém vnořených panelů

Na desktopu je pro zvýšení přehlednosti použito vnořování panelů. Na mobilu to může být v některých případech nevhodné. To z toho důvodu, že každý takový panel obsahuje vnitřní i vnější okraj. Vnoření dvou panelů tedy způsobí zmenšení zobrazovací plochy mobilu obr. 11. Proto byla vytvořena CSS třída *panel-noBorder*, která v případě mobilu zruší okraj panelu, u kterého byla uvedena.

Je nutné podotknout, že rušení vnořených panelů není vhodné použít vždy. Především v případech, kde je vnitřních panelů více než jeden. Například oddělení vyhledávací komponenty od výsledku hledání a nebo oddělení odlišných aktivit.

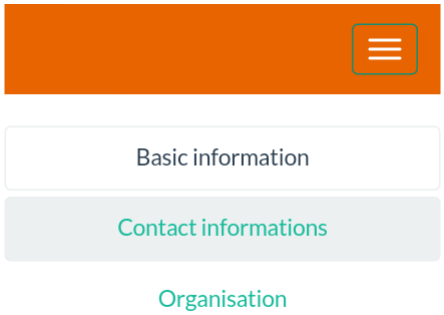


Obrázek 11: Vnořování panelů

9.2.3 Více menu na jedné stránce

Tento problém se týká především administrační části aplikace, ve které jsou celkově 3 menu. První menu je hlavní a zobrazuje se na každé stránce. Je tvořeno navbarem, takže se na mobilu schová pod rozklikávací tlačítko. Nebudeme ho tedy dále rozebírat.

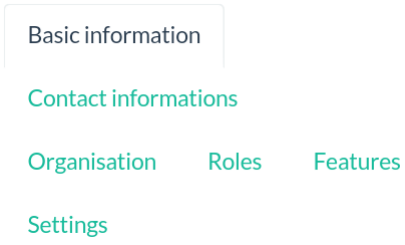
Druhé menu je hlavní administrační menu, které slouží pro pohyb v administrativní části aplikace. Dle zvolené záložky se zobrazí třetí menu. V první verzi této aplikace byly tyto menu vytvořeny tak, že se všechny odkazy zobrazily pod sebe. Problém byl však v tom, že na mobilním zařízení se stala celá navigace nepřehledná obr. 8. Menu se zpřehlednilo díky použití navbaru i v druhém menu této aplikace. Implementace tohoto menu je nastíněna v kapitole 8.2.2. Třetí menu bylo ponecháno v původním stavu. Výsledek této úpravy lze vidět na obr. 12



Obrázek 12: Nové administrační menu

9.2.4 Zobrazení panelu s taby

Jak výchozí wicket:bootstrap zobrazuje jednotlivé záložky je vidět na obr. 13. Tento panel tedy nerozlišuje, jestli se zobrazuje na mobilním zařízení nebo na desktopu. Vždy se zobrazuje stejně. Pokud se na šířku displeje nevejde, je zalomen. Kvůli tomuto zalomení se na velmi malých displejích stává nepřehledný a neestetický. Bootstrap implementuje CSS třídu *nav-justified*, která na displejích širších jak 768px způsobí, že se všechny záložky roztáhnou na celý řádek. Přičemž všechny tyto záložky budou mít stejnou šířku. Na displejích menších jak 768px se všechny záložky zobrazí pod sebe, viz obr. 8. Každá jednotlivá záložka bude roztažena na celý řádek. Jak vložit třídu *nav-justified* do tabbed panelu je popsáno v kapitole 8.3.2



Obrázek 13: TabbedPanel

9.2.5 Zarovnání

Stejně jako při testování na desktopu byly nalezeny nesrovnalosti v zarovnání. Tyto chyby byly opraveny pomocí správného a jednotného vnořování tříd *row*.

10 Závěr

Hlavním úkolem této práce bylo přeimplementovat prezentační vrstvu portálu Jasenverkko.fi a to tak, aby odpovídala responzivnímu webu. Práce spočívala v seznámení se s frameworky Twitter Bootstrap a Apache Wicket. Práce s Twitter Bootstrap byla velice pohodlná a jednoduchá. Bootstrap řeší mnoho problémů za nás, o čem vypovídá i velikost CSS souboru a počtu javasriptových souborů. Mnou napsaný CSS soubor obsahuje pouze 2 styly a javascript nebyl vůbec použit.

Framework Apache Wicket se mi zpočátku používal velice obtížně. Je třeba si zvyknout na používání modelů, značek a vhodného využívání dědičností. S přibývajícím časem jsem se s frameworkem více sbližoval, a nakonec práce s ním byla jednodušší než na začátku. Na frameworku mě nejvíce zaujala velice jednoduchá práce s technologií AJAX. Z mého pohledu je značnou výhodou tohoto frameworku to, že používá jazyk Java. Díky tomu bylo pro mne seznamování se s Wicketem pohodlnější.

Samotná přeimplementace prezentační vrstvy probíhala především pomocí úpravy markupu. Java kód byl občas také měněn. Hlavním důvodem změn Java kódu bylo použití knihovny wicket:bootstrap. Největší výzvou této části práce byla pro mě modifikace modálních oken. Při přeimplementaci těchto oken se vyskytl nejeden problém, nicméně nakonec vše funguje dle představ.

Největším přínosem pro mne bylo použití frameworku Apache Wicket. Také pro mne bylo přínosné to, že aplikace byla celkem rozsáhlá a práce bylo hodně. Proto bylo důležité si vše správně naplánovat a rozložit.

Bohužel nebylo možné otestovat všechny komponenty aplikace z důvodu neúplně funkčního prototypu. Nicméně díky komponentovému frameworku je pravděpodobné, že na neotestovaných stránkách vše bude fungovat tak jak má. Aplikace by podle mě zasloužila propracovanější a kvalitnější design. Nicméně i přes uvedený problém s testováním lze konstatovat, že cíl této práce byl splněn v celém svém rozsahu.

Reference

[1] Apache Wicket [online]. [cit. 2015-04-30]. Dostupné z: http://wicket.apache.org/

[2] Responsivní webdesign. [online]. [cit. 2015-02-23]. Dostupné z: http://blog.martinkelnar.cz/responsivni-webdesign-komplexne/

[3] Jak vytvořit responzivní design [online]. [cit. 2015-04-30]. Dostupné z: http://www.whitehat.cz/jak-vytvorit-responzivni-design/

[4] W3shools: bootstrap. [online]. [cit. 2015-02-23]. Dostupné z: http://www.w3schools.com/bootstrap/bootstrap_get_started.asp

[5] Úvod do architektury MVC. [online]. [cit. 2015-02-24]. Dostupné z: http://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/

[6] Wicket: Property model. [online]. [cit. 2015-03-06]. Dostupné z: http://ci.apache.org/projects/wicket/apidocs/6.0.x/org/apache/wicket/model/PropertyModel.html

[7] Media queries [online]. [cit. 2015-04-30]. Dostupné z: http://www.w3.org/TR/css3-mediaqueries/

[8] Bootstrap: Modal's bug. In: [online]. [cit. 2015-03-30]. Dostupné z: http://stackoverflow.com/questions/11519660/twitter-bootstrap-modal-backdrop-doesnt-disappear

[9] Viewport. [online]. [cit. 2015-04-17]. Dostupné z: https://www.interval.cz/clanky/trapeni-webdesigneru-s-viewporty/

[10] Wicket: Internacionalizace. [online]. [cit. 2015-04-17]. Dostupné z: https://wicket.apache.org/guide/guide/i18n.html#i18n_1

[11] Bootstrap [online]. [cit. 2015-04-24]. Dostupné z: http://getbootstrap.com/