

Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Vývoj portálových aplikací
Diplomová práce

Autor: Zdeněk Věcek
Studijní obor: Informační management

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 23.2.2015

Zdeněk Věcek

Poděkování:

Chtěl bych upřímně poděkovat panu doc. Ing. Filipu Malému, Ph.D. za odborný dohled, cenné rady, trpělivost a ochotu, kterou mi v průběhu zpracování diplomové práce věnoval.

Anotace

Diplomová práce se zabývá analýzou portálových řešení, od jejich vnitřní architektury po hotové produkty. Důraz je kladen na zdokumentování klíčových úseků v portálu Liferay. Vývoj nových zásuvných modulů a možnosti správy jsou velice rozsáhlé, proto byly některé z nich demonstrovány na funkční aplikaci pro upomínkování a organizování úkolů. Čtenář bude seznámen se strukturou portálu a budou mu osvětleny možnosti jaké Liferay nabízí se zaměřením na často používané části.

Annotation

Title: Portal application development

The diploma thesis is concerned about analysis of portal solution, starting from inner architecture to whole product. One of the main concerns is documenting of key segments in Liferay portal. Development of new plugin modules and options for administration are very comprehensive, that is why some of them are demonstrated on running application for scheduling, notifications and task organization. Reader will be acquainted with portal structure and enlightened into options which Liferay provides, aiming for frequently used parts.

Obsah

1	Úvod.....	1
2	Systémy pro správu obsahu – CMS.....	2
3	Portál a portlety	5
3.1	Portál.....	5
3.2	Portlet kontejner	5
3.3	Portlet	6
3.3.1	Životní cyklus portletu	7
3.4	Portletové specifikace	8
3.4.1	JSR 168.....	9
3.4.2	JSR 286.....	9
4	Konkurence na trhu.....	11
4.1	Backbase	12
4.2	IBM WebSphere	12
4.3	eXo platform.....	13
5	Liferay portál	14
5.1	Architektura	14
5.2	Vývoj plugin modulů	17
5.3	Marketplace	18
6	Administrační rozhraní Liferay	19
6.1	Uživatelé	19
6.2	Weby.....	20
6.3	Aplikace.....	21
6.4	Konfigurace.....	21
7	Portlet plugin Liferay	22
7.1	Konfigurační soubory	22

7.2	Spring MVC Portlet	23
7.2.1	Nastavení Spring MVC portletu.....	23
7.3	Meziportletová komunikace	24
7.3.1	Události.....	24
7.4	Více instancí portletu.....	26
7.5	Friendly URL	26
7.6	Alloy UI	28
7.7	Service Builder	28
8	Theme plugin Liferay.....	32
8.1	CSS.....	33
8.2	Look and feel	33
9	Layout plugin Liferay.....	35
10	Hook plugin Liferay.....	37
10.1	Aplikační adaptér	37
10.2	Přepsání zabudovaných servisních tříd	38
10.3	Pre a post akce	38
10.4	Filtry	39
11	Ext plugin Liferay.....	40
11.1	Nasazení pluginu.....	41
12	Funkční popis aplikace	44
13	Výběr JS frameworku	45
13.1	ReactJS.....	46
13.2	Flux	48
14	Architektura aplikace Evrem	51
14.1	Klientská část aplikace.....	51
14.2	Serverová část aplikace	55

14.2.1	Apache Maven.....	59
14.3	Administrační nastavení	61
14.4	Použité technologie.....	61
15	Výhody a nevýhody použití platformy Liferay	62
16	Shrnutí výsledků	63
17	Závěry a doporučení.....	64
18	Seznam použité literatury	66
19	Přílohy	71
20	Zadání závěrečné práce.....	76

Seznam obrázků

Obrázek 1 Moderní Web Content Management System	4
Obrázek 2 Diagram spolupráce portálu a portletů	6
Obrázek 3 Životní cyklus portletů	8
Obrázek 4 Srovnání horizontálních portálů	11
Obrázek 5 Architektura komponent v Liferay portálu	16
Obrázek 6 Technologická základna Liferay portálu	17
Obrázek 7 Management uživatelů v portálu Liferay	20
Obrázek 8 Unidirectional flow	50
Obrázek 9 Schéma klientské části	55
Obrázek 10 Schéma serverové části	60
Obrázek 11 Ukázka aplikace - úvodní obrazovka	71
Obrázek 12 Ukázka aplikace - potvrzení registrace	72
Obrázek 13 Ukázka aplikace - emailové notifikace	72
Obrázek 13 Ukázka aplikace - interaktivní zed' poznámek	73
Obrázek 14 Ukázka aplikace – úvodní přehled událostí	73
Obrázek 15 Ukázka aplikace - kalendář událostí	74
Obrázek 16 Ukázka aplikace - filtrace a zobrazení všech poznámek	74
Obrázek 17 Ukázka aplikace - administrační panel a přidání portletu	75
Obrázek 18 Administrační rozhraní Liferay 6.2.	75

Seznam tabulek

Tabulka 1 Základní parametry URL	26
Tabulka 2 aplikované technologie	61

Seznam kódu

Kód 1 Definice Spring servletu	23
Kód 2 Definice Spring třídy DispatcherPortlet	23
Kód 3 Spring aplikační kontext	24
Kód 4 Definice události	25
Kód 5 Odeslání události a její příjem	25
Kód 6 Události na straně klienta	26

Kód 7 Získání ID instance v třídě controller nebo JSP	26
Kód 8 Tvorba URL adres přes JavaScript.....	27
Kód 9 Registrace mapovacího souboru.....	27
Kód 10 Mapování URL	28
Kód 11 Vytvoření entity v servisní třídě.....	31
Kód 12 Dva odlišné vzhledy definované v souboru look-and-feel.xml.....	34
Kód 13 Dvousloupcový layout s fixně vloženým portletem.....	36
Kód 14 Programové přidání portletu.....	36
Kód 15 Modifikace portálu pomocí Hook pluginu	37
Kód 16 Zakomponování původního JSP portálu.....	38
Kód 17 Nahrazení interní servisní třídy	38
Kód 18 Modifikace Liferay portálu skrze Ext plugin.....	42
Kód 19 Definice akce pro dispatcher	48
Kód 20 Flux controller-view.....	49
Kód 21 Flux store.....	49
Kód 22 Tvorba URL v JSP	51
Kód 23 Inicializace view vrstvy	52
Kód 24 Compass a SASS použití	54
Kód 25 Spring MVC controller – render a resource požadavek.....	56
Kód 26 Nadefinovaný Liferay scheduler.....	57
Kód 27 Programové přidání administrátorských práv danému vláknu	57
Kód 28 Entity definované v service builder definici	58
Kód 29 Zaslání emailů přes zabudované komponenty.....	59
Kód 30 Properties nastavené v settings.xml v Maven	59

1 Úvod

Portálové aplikace zastávají již delší dobu klíčovou roli ve sféře enterprise systémů. Rozsáhlé korporace obsahují stovky malých, středních, ale i velice komplexních webových aplikací, ke kterým musí zaměstnanec denně přistupovat. V takovémto prostředí je uživatel nucen u každé aplikace poskytnout své autentizační údaje na různých přístupových adresách či pracovat na různých platformách. Kooperace těchto aplikací je řešena pomocí rozmanitých řešení. Vývoj probíhá v různých jazycích, kde znalosti nebývají přenositelné. To mnohdy vede po odchodu vývojáře ke ztrátě cenného know-how, jež se jen velmi těžko jeho nástupcům získává. Rozmach v oblasti podnikového softwaru musel být postupně korigován do větších integrovaných celků, jež poskytovaly odpovědi na společné často řešené problémy a poskytovaly jednotné rozhraní.

Portletová architektura odpovídá na tyto otázky. Na trhu můžeme nalézt jak kvalitní řešení typu open source, tak především systémy s většinou nemalými licenčními poplatky. Téměř vždy se jedná o velice komplexní systémy s velkou učící křivkou, z důvodů potřeby mnoha technologií a konceptů, které se v tomto segmentu využívají. Dokumentace pokrývá pouze základní část a to z důvodů udržení know-how, nabídnutí svých expertních služeb za poplatek nebo z důvodů rychlého vývoje těchto produktů a zanedbání souběžného zpracování znalostí. V případě platformy Liferay tomu není jinak a vývojář je tak často odkázán na analýzu kódu portálu, jakožto jediný spolehlivý, dostupný zdroj informací.

Tato práce si klade za cíl analyzovat portletovou architekturu a nastínit obecné možnosti použití. Po vymezení obecných standardů budou srovnány některé konkurenční produkty na trhu. Hlavní důraz je kladen na představení a zevrubném prozkoumání portálu Liferay, jakožto jednoho z hlavních lídrů na trhu JAVA horizontálních portálů. Čtenář bude postupně seznámen s vnitřní architekturou a administračním rozhraním umožňující významné zásahy do celé aplikace. Klíčová část se zabývá vývojem zásuvných modulů, umožňující rozšíření a kompletní transformaci portálu. Popsány budou důležité úseky, možnosti řešení různých problémů, ale i nedostatečně zdokumentovaných kód. V poslední části bude představena aplikace demonstrující jeden ze způsobů vývoje pod platformou Liferay s použitím nejmodernějších přístupů a technologií.

2 Systémy pro správu obsahu – CMS

Content management system či specificky web content management system je nástroj umožňující flexibilní a uživatelsky přívětivou správu obsahu na webu. Obsahem může být myšleno téměř cokoliv – text, obrázky, videa, hudba nebo dokumenty. V dřívějších dobách probíhal proces publikování nových informací následovně. Vlastník přišel s požadavkem, co by potřeboval umístit na své stránky. Sepsal text včetně obrázků a dalších médií do emailu a zaslal ho svému správci webu. Ten převedl obsah do HTML podoby a s trochou štěstí zaslal zformátovaný fragment ke schválení. Poté aktualizoval stránky nahráním nových souborů přes ftp připojení na web server. Celý proces vyžadoval úzké propojení vývojáře a zadavatele a ne vždy se představy o finální podobě střetávaly. Rychlost zpracování nebyla dostatečně pružná a vyžadovala dodatečné náklady.

Jednou z odpovědí na tuto problematiku byly WYSIWYG editory neboli software pracující na principu „What You See Is What You Get“. Nástroje jako MS FrontPage nebo Adobe Dreamweaver umožnily mírně pokročilým uživatelům tvořit vlastní stránky za pomoci jejich vizuálního aranžování. Uživatel snadno uchopil element, který naskládal na svoji stránku a editor se postaral o překlad do HTML kódu. Jejich nevýhodou bylo vygenerování mnoha nepotřebných fragmentů, které často způsobovaly problémy s kompatibilitou, a byly umísťovány hůře při indexování roboty. Ruku v ruce s vývojem obdobných nástrojů, šel i vývoj nových standardů pro HTML, CSS a JavaScript. Tím se vývoj začal stávat složitější ve všech ohledech, jelikož nové trendy vyžadovaly weby dynamičtější, responzivní a především uživatelsky přívětivé a zajímavé.

Tento pokrok přispěl ke vzniku WCMS nástrojů, jež většinu základních problémů řeší a nechávají prostor pro uživatelskou správu, tak aby nebyla narušena konzistence a celková funkčnost. WCMS umožňují vysokou modularitu s možností spravovat nejen obsah, ale i rozšiřovat aplikaci o zásuvné moduly, poskytující nové funkcionality. Běžný uživatel mohl bez asistence vývojáře vložit jednoduchý obsah s určitými formátovacími možnostmi do předem vydefinovaných bloků na stránkách, spojených navigací. Tento pokrok přispěl k decentralizované tvorbě nových dokumentů a obsah začal vznikat flexibilněji a byl sdílen mezi zainteresovanými stranami. To byly počátky WCMS systémů [Williams, n.d.]. S rozvojem sociální sítí a dynamický webů obecně, si uživatelé postupně zvykli na vyšší standard v oblasti interaktivity, UX a možností poskytovaných webovou

aplikací. Adaptování na tyto trendy dospělo do stavu, kdy lze WCMS definovat několika body, jež většina z nich implementuje.

- Tvorba, editování, mazání a **sdílení rozmanitého obsahu** pro běžného uživatele.
- **Autentizační a autorizační** přístupové bariéry s managementem uživatelů, vrstvení do rolí a skupin.
- **Spolupráce** a komunikace mezi uživateli.
- **Škálovatelnost** za běhu nebo s relativní jednoduchostí. Lze přidávat jak nové stránky, tak nové aplikace.
- **Stylování, šablonování** a celková změna uživatelského designu skrze nastavení.
- **Workflow** tvorby obsahu. Nový obsah mohou zakládat určití uživatelé, podléhající schválení zodpovědnou osobou, která obsah zreviduje a vypublicuje nebo vrátí k dopracování.
- **Staging** umožňující náhled před samotným vypuštěním obsahu do světa.
- Obsah lze snadno **internacionalizovat** do více jazyků.
- **Verzování** jednotlivých změn.
- **Vyhledávání** v obsahu, indexování.
- **Emailování** uživatelům na základě vydefinovaných šablon.

Následující diagram výstižně popisuje možnosti moderních webových systémů pro správu obsahu. Propojení se sociálními sítěmi zaštiťuje autentizaci přes sociální účty nebo provázanost s firemními stránky, založenými například na Facebooku, Twitteru či Pinterestu. Aplikace jsou optimalizovány pro vysoký výkon a poskytují možnosti rozšíření. Obsah lze nejen tvořit, upravovat, ale i sdílet a seskupovat do vlastních struktur. Mnoho CMS poskytuje vlastní wiki, blogy a řešení pro nahrávání a sdílení dokumentů. Zásuvné moduly poskytují doplňkové funkce a často zde nechybí ani propojení s ostatními uživateli pro spolupráci na společném úkolu. A/B testování umožňuje neinvazivní experiment pro získávání informací od uživatelů o úspěšnosti nového obsahu, designu či funkcionality zobrazené pouze části uživatelů z demografického spektra. Trh mobilních aplikací se neustále rozvíjí a odhaduje se, že každý třetí návštěvník je mobilní. Z tohoto důvodu je brán velký ohled na optimalizaci pro tato zařízení, aby přinesla uživateli co nejlepší zážitek z prohlížení. Celé rozvržení si

může uživatel, případně administrátor upravit dle vlastního uvážení do vydefinovaného layoutu bez zásahu do kódu [Shipley, 2015].



Obrázek 1 Moderní Web Content Management System

Převzato z [Episerver, 2015].

Základní funkce a moduly dodávané s produktem většinou nedostačují. Uživatel může hledat možnosti rozšíření skrze obchody se zásuvnými moduly, kam široká veřejnost může přispívat a rozšiřovat tak působnost platformy nebo může požadavek zadat vývojáři, který takový modul vytvoří. Pro snadnější modifikace jsou tyto systémy dodávány se sadou vývojových nástrojů připravených pro tvorbu nových rozšíření nebo úpravu stávajících řešení. Všechny podpůrné komponenty, API, IDE mají souhrnný název Content management frameworks. Při výběru je důležitý především programovací jazyk v jakém je produkt napsaný a jaký je dále nutné použít pro samotný vývoj. Hned poté jsou to technologie, které daná platforma umožňuje nebo je přímo v sobě integruje.

Produktem stojícím vedle WCMS jsou portály. Rozdíl bychom mohli spatřit v zaměření, které je v prvním případě orientováno na bohatý obsah zobrazovaný koncovému uživateli tvořený správcem. Druhá varianta je orientována na sdružení rozmanitých zdrojů do uceleného systému zobrazovaných dle oprávnění a rolí. Většina portálů však zahrnuje integrovaná WCMS řešení, proto nelze zcela vymezit působnost těchto dvou pojmů.

3 Portál a portlety

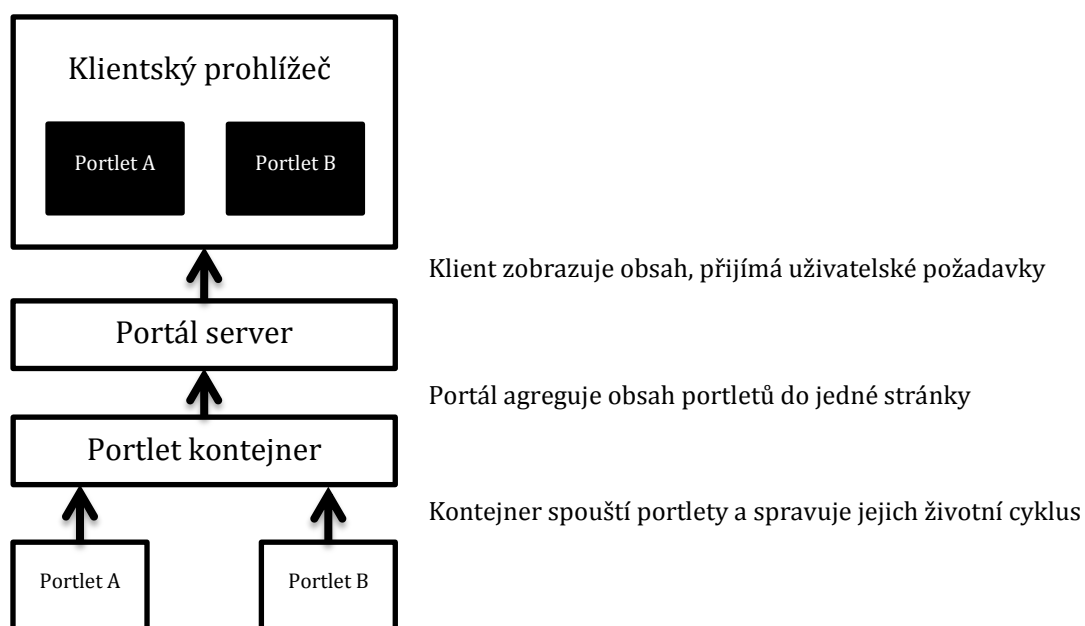
3.1 Portál

Tento název je do IT světa přejat z definice vstupní brány do jiného světa či ozdobný vstup do budovy. Výběr tohoto jména začne dávat větší smysl po přesnějším vydefinování, kdy lze spatřit analogii mezi různými interpretacemi. Portál je webový systém poskytující centrální přístupový bod uživateli k rozmanitému obsahu a službám. Agreguje obsah podle uživatelských preferencí a nabízí řadu možností pro personalizování prvků na stránce, rozložení, jejich vzhled nebo bližší nastavení vztahující se k uživateli. Lze je dělit podle působnosti na horizontální a vertikální. Vertikální ucelují zdroje a služby na úzce specifikovanou oblast například pojišťovna nebo mobilní operátor. Horizontální nemají definované hranice a mohou tak sdružovat zdroje a služby z celého spektra témat od počasí, politiku, financí přes automobily, vědu či módu [Williams, n.d.]. Příkladem je seznam.cz či yahoo.com. Nemusí se vždy jednat o giganty, překlenující všechna odvětví. Horizontálním portálem může být i aplikace zaměřená na jeden trh, využívaná více společnostmi nebo pouze čerpá z rozmanitých zdrojů či platforem.

Zajišťuje autorizační či autentizační mechanismy jakými jsou například Single Sign On, integrace se sociálními sítěmi, adresářovými službami LDAP, openID nebo OAuth. Poskytují kompletní správu uživatelů, rolí a hierarchicky je uspořádává do organizací nebo logických skupin. Nechybí ani základní komponenty pro sdílení a úpravu dynamického obsahu či přímo integrovaného CMS. Výsledkem portálové stránky je sloučení několika různých HTML fragmentů generovaných portletovými aplikacemi. Portál přímá požadavky od klienta a deleguje je do portlet kontejneru.

3.2 Portlet kontejner

Portlet kontejner poskytuje běhové prostředí pro portlety, podobně jako servletový kontejner zaštiťuje servlety. Spravuje jejich životní cyklus a stará se o přijímání požadavků od hostujícího portálového serveru, které dále deleguje konkrétním portletům. Jak servlety tak portlety generují obsah, který může být statický i dynamický.



Obrázek 2 Diagram spolupráce portálu a portletů

Převzato a upraveno [Meera, 2014]

3.3 Portlet

Portlety jsou webové komponenty zodpovědné za specifické funkcionality, služby či určitý obsah. Jejich klíčovým prvkem je možnost shlukování na jednu stránku jako nezávislé a soběstačné celky. Oproti servletové architektuře kdy je na danou adresu namapován příslušný servlet, vyřizující příchozí požadavky a vykreslující celou stránku, portletová architektura pracuje s konceptem mapování jedné URL více portletům, obsluhovaných jejich kontejnerem a vykreslující pouze část – fragment HTML [Basham, 2008].

Odeslání formuláře uvnitř portletu zpravidla nekončí obnovením celé stránky, ale pouze rámcem ve kterém je umístěn, data mohou pocházet z několika zdrojů, aniž by uživatel poznal rozdíl a osoby s vyšším oprávněním mohou mít přístup k více portletům.

Vyznačují se také rozšiřujícími vlastnostmi typu stav okna (Window state) nabývající hodnot maximalizovaný, normální či minimalizovaný určující jakou část stránky zabírají. Mód portletu (Portlet mode), kde mimo výchozího typu View podporuje také Edit, jež je vhodný pro administrátorské úpravy a Help, který slouží pro sdílení informací o portletu a o tom jak s ním pracovat. Mnohdy je zapotřebí perzistovat dodatečné informace a nastavení, dostupná i po restartu serveru, a proto je zde mechanismus umožňující nastavovat a získávat jednotlivé položky Preferences

(nastavení portletu). Ty je možné nastavit staticky do souboru *portlet.xml* nebo dynamicky do databáze spravované portálem.

3.3.1 Životní cyklus portletu

Životní cyklus portletu vychází z interface `javax.portlet.Portlet`, jež nám definuje čtyři základní metody zajišťující interakci s portlet kontejnerem. Ve stejném API nalezneme výchozí implementaci tohoto rozhraní v podobě třídy `GenericPortlet` dostačující většině užitých případů. S rozšiřováním standardu přibýly další funkce, které tato třídy řeší. Jedná se o interface `PortletConfig`, `EventPortlet` a `ResourceServingPortlet`.

Základní metody životního cyklu:

- **`init(PortletConfig config)`** – je volána jednou ihned po vytvoření instance portletu. Může sloužit pro spouštění inicializačních úkonů. `PortletConfig` obsahuje statické informace specifikované v popisném souboru (*portlet.xml*).
- **`processAction(ActionRequest request, ActionResponse response)`** – metoda volána po odeslání formuláře nebo kliknutí na odkaz. Úkony v této lokaci slouží pro vykonávání business logiky či modifikaci dat nebo změnu stavu portletu.
- **`render(RenderRequest request, RenderResponse response)`** – metoda volaná při prvním přístupu na stránku nebo po action fázi v životním cyklu. Slouží pro vykreslení fragmentu obsahu, který je také závislý na aktuálním stavu portletu.
- **`destroy()`** – posledním úkonem před automatickým spuštěním nástroje Garbage Collector je vykonání této metody, jejímž úkolem je uvolnění všech rozpracovaných zdrojů.

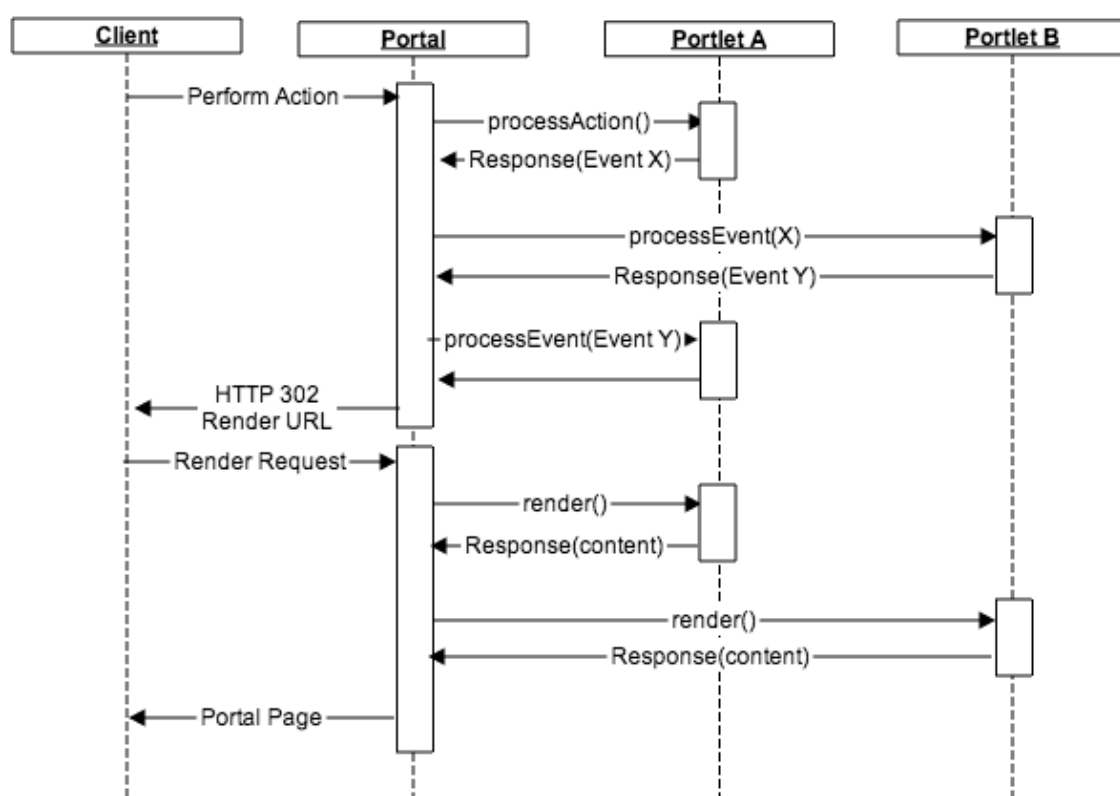
Rozšiřující operace životního cyklu:

- **`EventPortlet`** – po vykonání action fáze je před render fází, v případě nadefinování, zavoláno vykonávání události. To umožňuje meziportletovou komunikaci, která je popsána jako „loosely coupled“ (volně svázaná). Portlet zavolá `setEvent()` během vykonávání action fáze a konzumující strany poslouchající na danou událost, vykonají `processEvent()`, v němž je dostupný

odesílaný objekt, metodou `getEvent()` pod daným objektem `EventRequest`. Takto spolu mohou jednosměrně interagovat vzájemně propojené strany.

- **ResourceServingPortlet** – obsahuje jedinou metodu `serveResource()` sloužící pro AJAX komunikaci volanou JavaScriptem na klientovi.

Provázanost základního životního cyklu s událostmi je znázorněna na diagramu níže.



Obrázek 3 Životní cyklus portletů

Převzato z [Dalquist, 2013].

3.4 Portletové specifikace

V době kdy přicházely na trh portálová řešení, nebyl specifikován žádný standard. Díky tomu vznikala dodavatelsky závislá řešení a uživatelé byli odkázáni na použití jediné platformy. Netrvalo dlouho a velcí lídři na trhu s portály vznesly požadavek do Java Community Process (JCP) na standardizaci těchto aspektů, pro zajištění kompatibility, přenositelnosti a jednotného přístupu k této problematice. Reakcí bylo v roce 2003 vytvoření specifikace JSR 168 a později JSR 286.

3.4.1 JSR 168

Tato první specifikace definuje portlety jako webové komponenty založené na jazyce Java spravující nadřazeným kontejnerem. V tomto kontextu se jedná o zásuvné moduly rozšiřující presentační vrstvu portálu [Sarin, 2012].

Specifikace standardizuje následující aspekty:

- Definuje **běhové prostředí portletů** – kontejner a API pro jejich komunikaci.
- Zajišťuje mechanismus **ukládání dočasných a trvalých dat**.
- Poskytuje způsob **integrace se servlety**, potažmo JavaServer Pages (JSP).
- Definuje strukturu pro nasazení do prostředí webového kontejneru.
- Umožňuje **portabilitu** napříč portály dodržující JSR 168.
- **Agreguje portlety** do portálové aplikace ve formě WAR.
- Definuje **popisný soubor *portlet.xml***, rozšiřující standardní popisný soubor *web.xml* u servletových aplikací.
- Zavádí **módy** pro správu, nápovědu a zobrazování
- Definuje **stavy** pro určení rozsahu zobrazení fragmentu na stránce

3.4.2 JSR 286

Po několika letech uvedení do provozu prvního standardu, bylo otestováno a zjištěno mnoho nedostatků. Jelikož neexistoval žádný standard, kterého by se dodavatelé portálových řešení měli držet, tvořili svá vlastní řešení, kterým kompenzovali mezery ve specifikaci. Toto konání způsobilo opět ztrátu portability, ze které portlety těží. V reakci na to, vznikl v roce 2008 standard JSR 286, který tyto nedostatky řeší [McKenzie, 2013a].

Nově zpracované a rozšířené oblasti:

- Meziportletová komunikace prostřednictvím **událostí**.
- **Veřejné vykreslovací parametry** umožňující sdílení dat mezi portlety.
- **Poskytování zdrojů** (souborů) pomocí metody `serveResource()`.
- Podpora **AJAX** požadavků.
- Rozšíření oblasti působnosti o přístup ke **cookies a hlavičky dokumentu**.

- Zapracování **dalších typů životního cyklu** ze specifikace servletů, umožňující více přístupových bodů pro vlastní reakce na dané události.
- Přidání možností rozšíření pro vlastní reakce, modifikující či obalující určité chování našimi funkcemi.
 - **Filtry** – akce vykonávané před nebo po dané události.
 - **URL Listener** pro globální modifikaci adres a parametrů.
- **Přidání parametrů** běhovému kontejneru.
- **Zpětná kompatibilita** s předchozím standardem.
- Nové možnosti **kešování** pro zvýšení výkonu.
- Přiblížení **PortletSession** servletové variantě `HttpSession` umožňující pamatování uživatelského stavu po dobu prohlížení.

4 Konkurence na trhu

Agentura Gartner každoročně srovnává a vyhodnocuje enterprise nástroje, zařazené ve stejných kategoriích. Mezi lídry v oblasti horizontálních portálů patří IBM, Liferay, Microsoft, Oracle a SAP. Tato pětice patří v roce 2014 k jedničkám na trhu, které jsou svoji stabilitou a perspektivou vhodné pro nejnáročnější portálová řešení. Ne náhodou je tématem této práce platforma Liferay, která již po páté v roce obsadila pozici lídra v magickém kvadrantu. Pro srovnání jsem si vybral kandidáty IBM WebSphere, eXo platform a Backbase. Všechny portály jsou postavené na Java EE standardech pro objektivnější srovnání.



Obrázek 4 Srovnání horizontálních portálů

Převzato z [Murphy et al., 2014]

4.1 Backbase

Backbase portal se pyšní titulem vizionář roku. Důvodem je využití moderních technologií a inovativního přístupu ve vztahu k zákazníkům. Díky tomu byl označen jako Customer Experience Platform, vhodný pro použití především ve scénářích kdy je v roli uživatele zákazník. Nabízí řadu nástrojů pro podporu marketingu. Obsah může být zobrazován na základě demografického zařazení, zdali je uživatel student, podnikatel, senior, atp. Integrovaný WYSIWYG editor a nástroje pro analýzu návštěvníků. Snaží se o jednotný uživatelský zážitek, který není narušován skoky mezi různými aplikacemi. Jeho prezentační vrstva je založena na dialogích a widgetech, odstraňujících potřebu obnovovat stránku. Svoji sílu demonstruje např. na bankovní aplikaci, jež připomíná sociální síť s nástroji pro komunikaci, inteligentní kalkulačkou, správou svých financí v předdefinovaných kategoriích, ale i snadnou platbu a grafické vyhodnocení cashflow.

Jeho čerstvost na trhu má ovšem své nevýhody, v podobě nedostatečné historie v určitých odvětvích, což může odradit potencionální zákazníky. Nejedná se o open source licenci a svoje know-how si velice dobře chrání. Není proto snadné s touto platformou začít a řádně odzkoušet její možnosti [Murphy et al., 2014].

Funkční fragmenty v případě Backbase nazýváme widgety. Jedná se o standardní portlety implementující oba dva standard JSR 168 a 286, od čehož se odvíjí i základní dodávané parametry popsané v předchozí kapitole.

4.2 IBM WebSphere

IBM WebSphere představuje dlouhodobou jedničku na trhu s portálovými řešeními. Díky své dlouholeté zkušenosti si vybudoval pověst tradičního portálu. Mnoho zákazníků na tyto zkušenosti slyší, což podporuje i faktická implementace snad každého případu B2C, B2B a B2E. Je dodáván s nejpestřejší škálou zabudovaných nástrojů, které jsou dále specializované na konkrétní použití ve vztahu k zákazníkovi nebo zaměstnanci.

IBM usiluje o dodání výjimečného zážitku uživateli používajícího portál, tyto komponenty a principy označuje pojmem Digital Experience. Sem spadají nastavení pro responzivitu, propojení se sociálními sítěmi, tvorbu formulářů během okamžiku přímo na webu, cloudová řešení, ale i bohatý obsah a analýzu uživatelů [IBM, 2014].

Velkou bariérou je cena, za kterou je možné portál provozovat. Roční licence jsou velice nákladné a mohou být odrazující. Je nutné hledět na celkové náklady na provoz IT, kam spadá integrace se stávajícími službami, hardware, nový vývoj, penále za výpadky či

chyby, které ve většině případů několikanásobně převyšují výdaje za software. Analýza, školení, testování a mnoho dalších položek v rozpočtu, související s provozem a vývojem portálu, jež mohou být součástí již hotového řešení [Kharkovski, 2013].

4.3 eXo platform

Exo platform spadá do kategorie intranetových sociálně založených enterprise portálů. Z tohoto důvodů není zařazen mezi horizontální portály. Jeho licence je však nabízena i v modelu open source, a proto se jeví jako ideální kandidát vedle Liferay portálu. Ačkoliv je tato technologie na trhu od roku 2003, prošla si několika zásadními změnami a stejně jako ostatní portály představené výše se zaměřuje na úzkou vazbu na potřeby uživatele a personalizovaný zážitek. Dokumentace však nabízí pouze základní případy užití a příspěvky na komunitním fóru zůstávají často nezodpovězené. Exo platform bych volil jako vhodnou alternativu vůči Liferay. V mnoha ohledech, především ve využití frontendových technologií se eXo jeví lépe. Bohužel ještě nějakou dobu potrvá, než se vytvoří dostatečná znalostní základna pro plynulý vývoj.

Byly představeny 3 vedoucí portálová řešení, excelující vždy ve specifickém ohledu. Na trhu je bohužel velice málo open source produktů s kvalitní výbavou a informační základnou. Společným atributem je vize jakou všechny produkty sdílejí. **Orientace na uživatele a dokonalý zážitek z prohlížení.** Tento princip nalezneme v každém z nich. Portály se postupně začínají podobat sociálním sítím a je jen těžké odhadnout, zdali bude tento faktor dostatečně využit a neodradí potenciální zákazníky. V následujících kapitolách si představíme platformu Liferay, která dominuje v open source řešeních a díky tomu má nejrozšířenější znalostní základu, která začíná u oficiální dokumentace, vydaných knih, příspěvcích na fórech, pokračující přes vypracovaná řešení na GitHub nebo popsané postupy na blogách či wiki. Tento aspekt shledávám klíčovým pro vývoj, jelikož výrazně zkracuje dobu řešení požadavku.

5 Liferay portál

Technologie se každým okamžikem ve světě programování mění. Neustále probíhá vývoj nových funkcionalit a frameworky jsou transformovány dle aktuálních trendů. U portálu toto platí dvojnásob. Jejich široká působnost vyžaduje zapojení mnoha knihoven, systémů nebo principů. Není snadné integrovat stále nejnovější verze technologií současně s moderními trendy. Liferay poskytuje mimo open source také enterprise edici, díky které musí být ve svém release cyklu mnohem opatrnější, jelikož si nemůže dovolit ztrátu kompatibility pro své platící zákazníky. Tento fakt často způsobuje nutnost využití starších frameworků a uzamknutí se v historických verzích. V této práci se dočtete o současně stabilní verzi 6.2, ačkoliv je verze 7 již na obzoru (milestone), přinášející například některé kýžené změny v oblasti frontendu.

5.1 Architektura

Platforma Liferay implementuje a podporuje řadu standardů a technologií, kterými se tato kapitola bude zabývat. Integrace mnoha klíčových enterprise nástrojů bývá díky tomu snadná a vývojář se může soustředit na tvorbu business logiky.

Základním předpokladem funkční instalace je operační systém, jenž je možné využít. Seznam v aktuální verzi čítá 19 položek, kde nechybí populární Windows, Unixový systém Mac, Solaris nebo Linuxový Ubuntu či Red Hat. Dalším předpokladem je nainstalované běhové prostředí pro Java aplikace Java Virtual Machine, kde lze využít JDK od Oracle nebo J9 od IBM.

Instanci Liferay portálu lze provozovat na několika aplikačních serverech či lehčích servletových kontejnerech, jež musí obsahovat několik knihoven a nastavení. Na oficiálních stránkách lze stáhnout hotové balíčky s předpřipravenými prostředími. Jedná se o servery Tomcat, Glassfish, Geronimo, JBoss, Jetty, JOnAs + Tomcat nebo Resin. Podpora dalších však není vyloučena. Liferay se snaží o maximální odstínění serverově založených technologií, jako je EJB, jež je podporováno jenom částí zmíněných kontejnerů a nahrazuje tuto sadu komponent, serverově nezávislou technologií Spring. Pomocí ní, je vybudovaná komplexní servisní vrstva, zajišťující transakce, low-level logiku portálu a spolu s ORM frameworkem Hibernate datovou vrstvu. Liferay dále využívá některé Java EE komponenty v konjunkci se serverem, mezi něž patří:

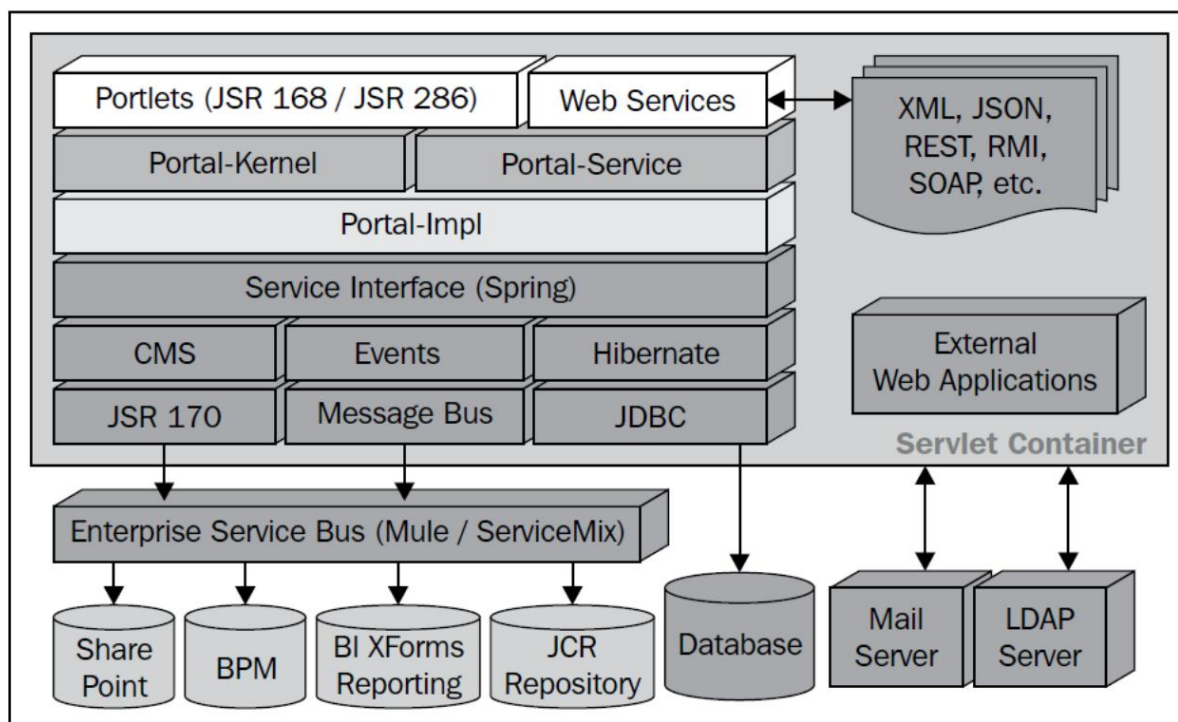
- **JNDI** (Java Naming and Directory Interface) – získávání objektů a dat prostřednictvím jména přes jednotné rozhraní (SPI).
- **JDBC** (Java Database Connectivity) – standardní způsob pro práci s databází.
- **JTS** (Java Transaction Service) – transakční management splňující ACID (Atomicity, Consistency, Izolation a Durability).
- **JAAS** (Java Authentication and Authorization Service) – API pro zabezpečení aplikace, jež popisuje autentizaci, autorizaci, šifrování nebo infrastrukturu veřejných klíčů.
- **JWS** (Java Web Service) – komunikace s externími službami pomocí webových služeb.
- **JSP/Servlet** (Java Server Pages) – dynamická značkovací view vrstva překládaná za běhu do Java servletů.
- **JavaMail** – sada nástrojů pro emailovou komunikaci.
- **JMS** (Java Messaging Services) – nepřímá výměna zpráv či naplněných objektů mezi různými klienty.

Pro vyhledávání a indexaci dat lze využít Apache Solr postavený na knihovně Lucene od stejného dodavatele. Mezi jejich přednosti patří především rychlost, škálovatelnost a pokročilé funkce při fulltextovém vyhledávání s možností tvorby speciálních dotazů. Solr je samostatný vyhledávací server komunikující prostřednictvím REST API. Přijímá data různých formátů přes protokol http, jež si zaindexuje a vystaví webové služby pro vyhledávání vracející výsledky na dotaz. Obsahuje nástroje pro správu, monitorování, dolování dat, optimalizaci serveru nebo i rozšíření v podobě vyhledávání v pokročilých formátech Word či PDF.

Nasazení do korporátního světa vyžaduje leckdy komunikaci mezi různými službami, kde formát zpráv a jejich vyřizování může nabývat různých podob. Jednotný prostředkem pro komunikaci je zabudovaná ESB neboli Enterprise Service Bus vrstva, starající se o interakci s vnějšími aplikacemi. Díky ESB je možné zapojovat rozmanité systémy, aniž by bylo nutné většího úsilí při integraci ze strany portálu. ESB se chová jako jakási vstupně výstupní výhybka, umožňující využití jednotného API pro výměnu zpráv s různými službami. Na výběr máme mezi Mule nebo Apache ServiceMix. Reálným příkladem je komunikace s různými Workflow systémy jako je Intalio nebo jBPM. Dále je

možná integrace s portálem Microsoft SharePoint nebo propojení s JCR (Java Content Repository) Apache JackRabbit.

Dalšími prostředky pro komunikaci jsou webové služby. Liferay poskytuje servisně orientovanou architekturu (SOA), jež umožňuje zasílat zprávy v rozličných formátech. Lze tak zasílat zprávy ve formátu XML, XML-RPC, JSON, REST, SOAP, RMI, Hessian, Burlap nebo tvořit vlastní tunely [Yuan et al., 2009].



Obrázek 5 Architektura komponent v Liferay portálu

Převzato z [Yuan et al., 2009]

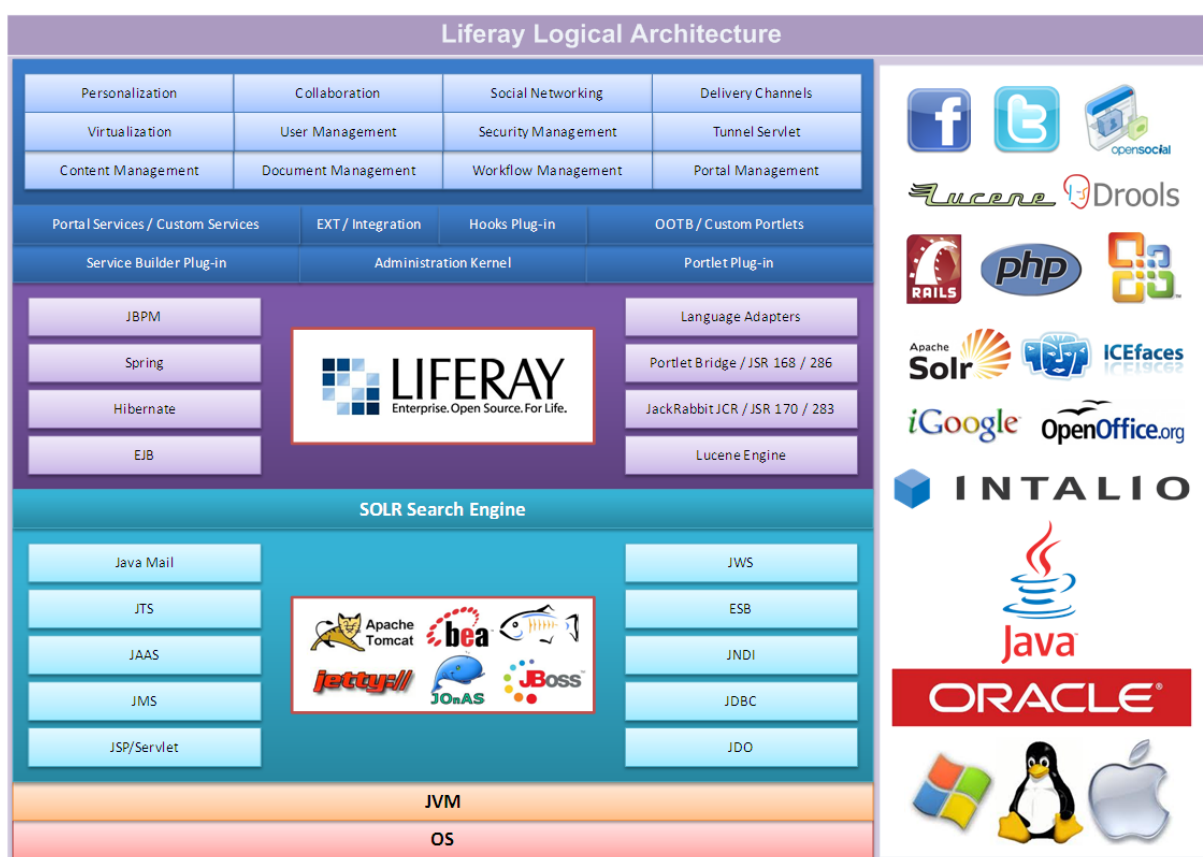
Psaní portletů je možné nejen v jazyce Java, ale také v dalších skriptovacích jazycích jako je PHP, Ruby on Rails, Python, Groovy atd. Díky tomu je klientská základna rapidně rozšířena a vývojáři ovládající rozličné programovací jazyky mohou těžit z výhod, jež portál přináší [Montero et al., 2010].

Zvyšování výkonu v kriticky důležitých oblastech je zajišťováno prostřednictvím klastrování, správného rozvrstvení požadavků přes load balancer, kešování či replikaci. V tomto ohledu lze využít širokou škálu zabudovaných nástrojů a nastavení.

Ve své nejvyšší vrstvě Liferay nabízí nástroje pro personalizaci, spolupráci, propojení se sociálními sítěmi, správu uživatelů, kompletní zabezpečení, tunelování (WebDAV – správa souborů přes http, vlastní servlety) a virtualizaci.

Mnoho položek nastavení je uživatelsky přístupné přes administrační rozhraní, což umožňuje rychlou reakci na změny bez nutnosti zásahu do kódu a přenasazování. Liferay integruje Content Management System propojený se standardem JSR-170 (JCR), Document Management System, ale také již zmíněné nástroje pro modelaci business proces modelů a jejich workflow [Augé et al., 2012].

V následujících kapitolách bude představena pro vývojáře nejdůležitější vrstva, představující tvorbu rozšíření v podobě portletů, modifikování v podobě modulu Hook, úpravu jádra přes Ext, sestavení vlastní šablony Theme nebo rozvržení obsahu do vlastního layoutu.



Obrázek 6 Technologická základna Liferay portálu
Převzato z [Augé et al., 2012]

5.2 Vývoj plugin modulů

Rozšíření portálu lze realizovat několika způsoby. Nejčastějším způsobem jsou portlety. Stejně esenciální bývá nutnost vlastního designu v podobě Theme pluginu. Následující kapitoly popíší některé klíčové části tvorby těchto zásuvných modulů pro platformu Liferay. Neexistuje jediný správný postup, a proto si každý vývojář musí zvolit

vyhovující variantu a vzít si pouze to co ve své aplikaci potřebuje. Kapitoly se nebudou zabývat elementárními tvorbou modulů od začátku až do konce, ale budou zaměřeny na důležité, zajímavé, problematické či špatně popsane části v oficiální dokumentaci. V následujících příkladech budeme používat nástroj pro správu projektu Maven a vývojové prostředí Eclipse s integrovaným Liferay IDE.

5.3 Marketplace

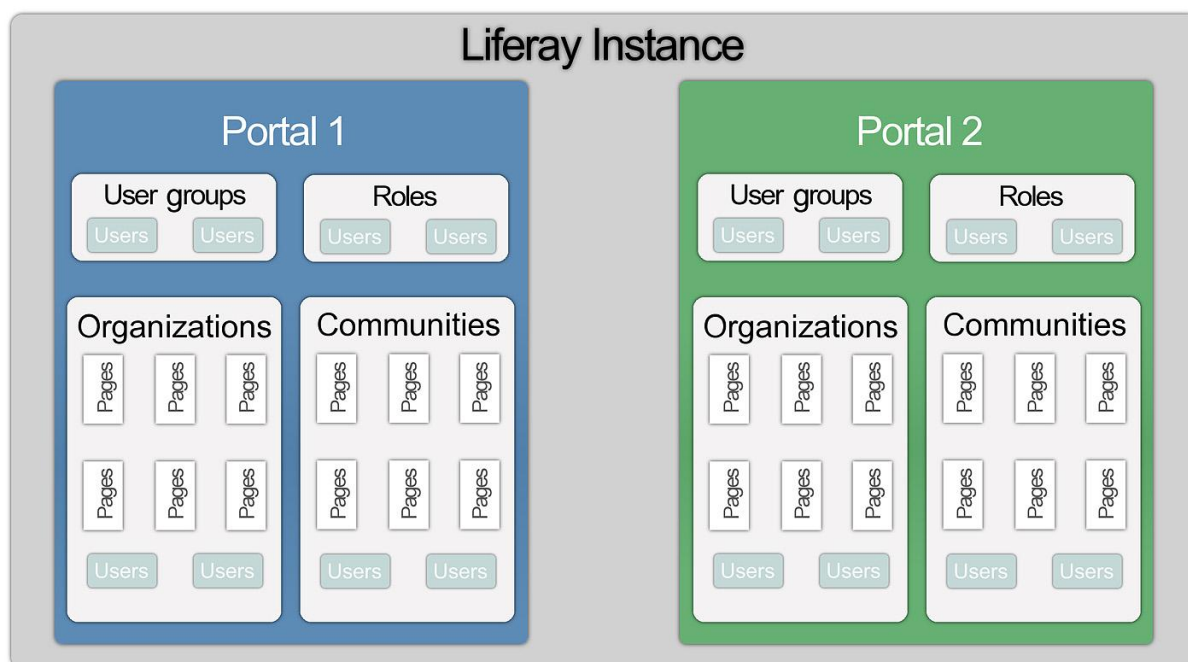
Velkou příležitostí pro nadšené vývojáře je tvorba zásuvných modulů pro Marketplace neboli internetový obchod, kde lze tyto doplňky prodávat a na opačné straně samozřejmě zakoupit a rozšířit si tak svůj portál bez znalosti jakýchkoli programovacích jazyků. Nalezneme zde nejen komerční produkty, ale i mnoho nezaplatněných řešení. Jsou zde portlety, Theme pluginy a layouty. Zakoupení již hotového a ověřeného doplňku se i přes jeho licenční poplatky může mnohokrát vyplatit, jelikož se jedná o masově distribuovaný produkt, jenž bývá zpravidla levnější než vlastní analýza, vývoj a testování.

6 Administrační rozhraní Liferay

Administrační rozhraní poskytuje nástroje pro komplexní správu celého portálu. Uživatel s příslušnou rolí vidí nejen lištu správce nazývanou Dockbar, ale prostřednictvím zabudovaných odkazů, může přejít do sekce Control panel, dělíci veškerá nastavení do čtyř hlavních kategorií viz Obrázek 19.

6.1 Uživatelé

Sekce uživatelé nabízí kompletní management uživatelů, organizací, rolí či skupin. Síla portálu je v jeho distribučních možnostech a personalizaci. Pomocí těchto členění není problém na jedné běžící instanci portálu provozovat služby pro několik organizací s vlastním odděleným místem pro správu uživatelů a vlastním administrátorem. Organizace lze hierarchicky členit, čímž lze kopírovat skutečnou strukturu společnosti. Každá jednotka mává reálně jiné potřeby a díky různým rolím a oprávněním lze portál jednoduše nastavit pro zobrazování specifického obsahu. Skupiny sdružují uživatele napříč organizacemi a umožňují tak konkretizovat oprávnění na úrovni jiné než organizační. Uživatelé ve skupině případně organizaci přejímají stejná práva, nastavená těmito rodičovským jednotkám. Na úrovni role lze nastavit detailní oprávnění pro práci na konkrétních stránkách, portletech, ale i jaké úkony lze provádět v administraci. Monitorování lze spustit nastavením příznaku v našem *portal-ext.properties* souboru, po čemž dostaneme možnost sledovat aktivní session přihlášených uživatelů s několika detaily o jejich aktivitě. Management uživatelů je doplněn o uživatelsky přívětivou politiku hesel zajišťující mnohé principy pro zvýšení bezpečnosti, jež mohou být nastavovány dle uživatelů nebo organizací.



Obrázek 7 Management uživatelů v portálu Liferay

Převzato z [Sezov, 2011]

6.2 Weby

Sites neboli weby společností jsou logická uspořádání stojící na vrcholu, obsahující konkrétní stránky s konkrétním obsahem. Vytvoříme-li novou Site, máme na výběr přidání veřejných a soukromých stránek. Veřejné stránky jsou nezabezpečené weby pro anonymního návštěvníka. Privátní naopak slouží pro přihlášené uživatele patřící do organizace nebo role s oprávněními pro danou Site s adresou dělenou názvem organizace. Zde lze kompletně personalizovat obsah pomocí vlastních pluginů typu Theme, portletů a layoutů, jež mohou být naprosto nezávislé na ostatních webech jiných společností běžících na stejné instanci portálu. Můžeme například vytvořit Site pro oddělení marketingu využívající nástroje pro analýzu uživatelů, emailování, správu novinek a jiný web pro oddělení účetnictví, kde budou portlety týkající se financí, výplat a dalších nákladů. Obě oddělení mohou využívat různého vzhledu pro lepší identifikaci či stejného pro sladění korporátního designu. To vše závisí na administrátorovi, jenž většinu úkonů provede pouhým „naklikáním“.

Pro snazší správu a větší znovu použitelnost zde existují šablonovací nástroje pro weby a stránky. Lze tak jednoduše vytvořit vzorovou stránku s rozložením a konkrétními portlety a umožnit tak rychle tvořit stránky s již vydefinovanými vlastnostmi s možností automatické aktualizace.

Zajímavou možností je také Staging. Jedná se o nástroj umožňující postupné publikování nových změn a obsahu. Zodpovědná osoba tak může za chodu vytvořit nový obsah, jenž si nejprve otestuje, předá ke schválení a teprve poté zveřejní pro všechny uživatele. V neposlední řadě zde lze pracovat se zabudovanými aplikacemi, umožňující správu dokumentů, blogů, zpráv nebo celých článků. Do tohoto menu je možné přidat i vlastní portlety, nakonfigurované pro zobrazení v administraci.

6.3 Aplikace

V této sekci je pravděpodobně nejdůležitějším nástrojem obchod s doplňky popsáný v kapitole 5.3. Nalezneme tu i vlastní aplikace, které můžeme fyzicky odstraňovat ze serveru nebo pouze deaktivovat. Užitečnou pomůckou mohou být i zobrazené ID u konkrétních pluginů, které je využíváné na mnoha místech při vývoji. Všechny aplikace jsou sdružovány do katalogů a u zakoupených modulů můžeme sledovat stav licencí. Administrační rozhraní umožňuje také aplikace přidávat přímo do běhového kontejneru pouhým nahráním webového archivu přes HTML formulář.

6.4 Konfigurace

Pokročilejší nastavení celého portálu je sdružováno na tomto místě. Úpravy zde provedené budou vztaženy napříč portálem, organizacemi a uživateli. Patří sem autentizační nastavení, konfigurace emailů, URL adres, lokalizace nebo propojení se sociálními sítěmi. Běžící instance lze monitorovat a lze vyčíst, s jakými konfiguračními položkami aktuálně pracuje. Logování aktivit lze nastavit pro konkrétní třídy nebo balíčky intuitivně v seznamu dle logovací úrovně. Pro zamezení spamu a jiného zneužití skrze automatické boty je zde integrovaný nástroj reCaptcha, vyžadující opsání zkomoleného textu, případně rozpoznání audionahrávky. Administrátor může zmigrovat data ze staré databáze a v případě potřeby načasovat vypnutí serveru s globální hláškou pro uživatele s automatickým zablokováním a odhlášením.

7 Portlet plugin Liferay

V předchozí kapitole 3.4, byly vydefinovány dvě specifikace popisující vlastnosti a funkčnosti portletů. Liferay tyto standardy implementuje a mimo běžných funkcí nabízí i řadu rozšíření, kompatibilních pouze s touto platformou. Jak již bylo zmíněno, jedná se o klasickou webovou aplikaci nasazovanou do webového kontejneru. Co však taková aplikace musí splňovat, aby se jednalo o Liferay portlet?

7.1 Konfigurační soubory

Odpovědi jsou popisné a konfigurační soubory ve složce */WEB-INF*. Skrze ně lze celkově ovlivnit chování, vzhled, typ, název portletu a mnoho dalších. V základním projektu vygenerovaném přes archetypy máme následující soubory.

web.xml – standardní servletový deskriptor. Ve výchozím nastavení je tento soubor prázdný, avšak pro využití Spring portletu, je zde nutné, zaregistrovat `ViewRendererServlet`.

portlet.xml – analogie k *web.xml* v portletovém světě dle standardů JSR. Patří sem název, obslužná třída pro příchozí požadavky, módy, stavy, podporované MIME typy, definice veřejných parametrů a událostí pro meziportletovou komunikaci, oprávnění definované rolemi a další. V jednom souboru lze nadefinovat více než jeden portlet a celý projekt tak může sloužit pro několik aplikací zároveň.

liferay-portlet.xml – rozšíření předchozího souboru o Liferay specifika. Mezi některé z nich patří definice CSS či JS souborů, které lze pomocí tohoto nastavení deklarovat v hlavičce nebo patičce portálu, respektive do umístění mimo kontejner portletu. Některé parametry slouží jako mapovací položky ze souboru *portlet.xml* na Liferay portál. Příkladem může být mapování rolí z *portlet.xml* na role uložené v portálové DB. Dále zde můžeme najít časovače spouštějící rutiny, definice URL, konfigurace sociálních, dokumentově či obsahově založených aplikací, položky pro přidání portletu do administračního rozhraní, zdali je možné vložit více instancí na stránku, viditelnost, globální nastavení portletu nebo parametry vzhledu.

liferay-plugin-package.properties – soubor obsahuje popisná metadata pro Liferay katalog softwaru, závislosti na knihovny obsažené v portálu nebo kontexty, jež je nutné nastartovat před tímto samotným.

liferay-display.xml – zařazení a název zobrazený při přidávání portletu na stránku.

7.2 Spring MVC Portlet

Vyřizování požadavků je ve výchozím nastavení realizováno třídou `GenericPortlet`, jež pokrývá životní cyklus portletu. Pro snadnější práci existují knihovny třetích stran zajišťující snadné mapování metod, parametrů, validování a další funkce, které vývojáři usnadňují život a zpřehledňují kód. K nejvýznamnějším z nich patří JSF, Struts a aktuálně probíraný Spring MVC.

7.2.1 Nastavení Spring MVC portletu

Ukázkový controller neboli vstupní bod mezi klientem a business vrstvou, je možné nalézt v kapitole 14.2. Správné fungování je podmíněno několika nastaveními [Nilang, 2012].

- 1) Zaregistrování `ViewRendererServlet` do *web.xml*. Tento servlet slouží jako most z portletového světa do servletového. Převádí požadavky do podoby `HttpServletRequest` a umožní tak přepoužít celou vrstvu kódu použitou v servletech.

```
<servlet>
  <servlet-name>ViewRendererServlet</servlet-name>
  <servlet-class>
    org.springframework.web.servlet.ViewRendererServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>ViewRendererServlet</servlet-name>
  <url-pattern>/WEB-INF/servlet/view</url-pattern>
</servlet-mapping>
```

Kód 1 Definice Spring servletu

- 2) Zaregistrování třídy `DispatcherPortlet` do *portlet.xml* spolu s propojením a inicializací aplikačního kontextu. Tato třída slouží jako vstupní bod, delegující požadavky na příslušné controller třídy.

```
<portlet-class>org.springframework.web.portlet.DispatcherPortlet</portlet-class>
<init-param>
  <name>contextConfigLocation</name>
  <value>/WEB-INF/{portlet-name}-portlet.xml</value>
</init-param>
```

Kód 2 Definice Spring třídy `DispatcherPortlet`

- 3) Vytvoření aplikačního kontextu zmíněného v bodu 2. Ve složce */WEB-INF* založíme soubor ve tvaru *{portlet-name}-portlet.xml*. Název není náhodný a je

nutné dodržet uvedenou konvenci malými písmeny. Každá aplikace využívající Spring, musí obsahovat, alespoň jeden kořenový kontext. Zde nadefinujeme použití anotací, rozlišování a mapování JSP dle zjednodušeného názvu a skenování balíčku s našimi třídami typu controller.

```
<context:annotation-config />
<bean
class="org.springframework.web.portlet.mvc.annotation.DefaultAnnotationHandlerMapping" />

<bean id="jspViewResolver"
class="org.springframework.web.servlet.view.UrlBasedViewResolver">
    <property name="viewClass" value="org.springframework.web.servlet.view.JstlView" />
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
    <property name="order" value="1" />
</bean>

<context:component-scan base-package="my.package.with.controllers"/>
```

Kód 3 Spring aplikační kontext

- 4) Vypnutí přepojmenovávání (namespacing) u parametrů v liferay-portlet.xml jednoduše přes položku `requires-namespaced-parameters` na hodnotu `false`.
- 5) Přidání závislostí na Spring knihovny. V případě Maven projektu se zavádějí do `pom.xml`. Potřebné knihovny **spring**-{aop, aspects, beans, context, core, expression, test, webmvc-portlet, webmvc}.

7.3 Meziportletová komunikace

Jedním z hlavních problémů řešených v počátcích portletů byl mechanismus, umožňující vzájemnou komunikaci. Modularita, jako jedna z velkých výhod portálu nemohla být bez tohoto aspektu splněna. S příchodem JSR 286 bylo na tuto otázku poskytnuto řešení v podobě veřejných vykreslovacích parametrů (`public render params`) a komplexního mechanismu zasílání událostí. Méně vhodným způsobem je výměna proměnných skrze objekt `PortletSession`, jež lze nakonfigurovat pro sdílení dat mezi portlety či portálem. Další možností komunikace je výměna dat přes prohlížečové Cookies, kde je nutné brát v potaz limitace na velikost, která bývá stanovena 4kB a počet položek musí být menší jak 20.

7.3.1 Události

Jak již zaznělo v kapitole 3.4.2, jedná se o standardní funkcionalitu, portletových aplikací. Komunikovat lze jak na straně klienta tak serveru. Události na straně serveru je

nutné zaregistrovat do souboru *portlet.xml*, spolu s datovým typem, jmenným prostorem a lokálním názvem na obou stranách u odesílatele i příjemce. Odesílatel nastaví do aktuální instance *ActionResponse* událost se zmíněnými parametry, čímž je odeslána událost a dané poslouchající strany, na danou událost zavolají obslužné metody. V příkladu níže je zobrazena definice odchozí a příchozí události a zaregistrování datového typu události. V případě výměny vlastních objektů je nutné, aby k dané třídě měly přístup obě strany a objekt byl serializovatelný [Zubair et al., 2014].

```
<portlet>
  <!-- Odesílatel -->
  <supported-publishing-event>
    <qname xmlns:x="http://myapp.com/events">x:note</qname>
  </supported-publishing-event>
  <!-- Příjemce -->
  <supported-processing-event>
    <qname xmlns:x="http://myapp.com/events">x:note</qname>
  </supported-processing-event>
</portlet>
<!-- Odesílatel i příjemce -->
<event-definition>
  <qname xmlns:x="http://myapp.com/events">x:note</qname>
  <value-type>java.lang.String</value-type>
</event-definition>
```

Kód 4 Definice události

Vytvoření takového mostu je pouze nezbytný předpoklad, po kterém můžeme začít psát samotnou logiku odeslání události a přijetí.

```
//Odesílatel
@ProcessAction
public void processAction(ActionRequest request, ActionResponse response){
  QName qName = new QName("http://myapp.com/events", "note", "x");
  response.setEvent(qName, "Nová poznámka");
}
//Příjemce
@ProcessEvent(qname = "{http://myapp.com/events}note")
public void processNoteRefreshEvent(EventRequest request, EventResponse response){
  Event event = request.getEvent();
  String value = (String) event.getValue();
}
```

Kód 5 Odeslání události a její příjem

Komunikace prostřednictvím událostí je silnou zbraní, která by se neměla užívat bez řádného promyšlení. Událostní propojování může při častém a nevhodném užití výrazně zneprůhlednit kód aplikací, a proto se běžněji setkáváme s komunikací pouze na klientské vrstvě pomocí JS příkazů níže. Zde stačí jednoduše nadefinovat název a kdekoliv jinde na aktuální stránce v jiném portletu je odchycena událost pod stejným jménem a zavolán obslužný callback. Jelikož je JavaScript netypový jazyk, nemusíme se

zatěžovat s definicí třídy a rovnou lze posílat data v polích, složitých objektech nebo strohých hodnotách.

```
//Odesílatel
Liferay.fire(eventName, data);
//Příjemce
Liferay.on(eventName, callback, [scope]);
```

Kód 6 Události na straně klienta

7.4 Více instancí portletu

V základním režimu můžeme portlet umístit na specifickou stránku pouze jednou. V takovém případě dostane název ve tvaru **portletId_WAR_webapplicationcontext**, který je odvozen z názvu portletu dle souboru *portlet.xml* a názvu kontextu aplikace, pod nímž běží ve webovém kontejneru. Kdekoliv se na tento portlet odkazujeme, použijeme zmíněný identifikátor. V případě více instancí je za tento název přidán ještě unikátní identifikátor pro označení konkrétní instance v podobě **_INSTANCE_ABC123**. Výhodou je zobrazení různých dat a specifická nastavení pro dvě stejné aplikace. Nastavení provedeme v *liferay-portlet.xml* položkou *instanceable* na hodnotu *true*.

```
ThemeDisplay themeDisplay = (ThemeDisplay) request.getAttribute(WebKeys.THEME_DISPLAY);
themeDisplay.getPortletDisplay().getInstanceId();
```

Kód 7 Získání ID instance v třídě controller nebo JSP

7.5 Friendly URL

Každá adresa v portálové aplikaci obsahuje mnoho informací. Důvodem je nezbytnost identifikace a další funkční parametry pro určení vzhledu, pozice v layoutu či módu v jakém portlet pracuje. Ačkoliv se jedná o potřebné informace, ve většině případů nás jejich nastavení nezajímá a vystačíme si s výchozími hodnotami. Běžná adresa může vypadat takto:

```
http://localhost:8080/web/evrem/login?p_auth=s6JWEcZf&p_p_id=loginportlet_WAR_loginportlet&
p_p_lifecycle=1&p_p_state=normal&p_p_mode=view&p_p_col_id=column-1&p_p_col_count=1
```

Tabulka 1 Základní parametry URL

Parametr	Funkce
p_p_id	ID portletu viz kapitola 7.4.
p_p_lifecycle	Životní fáze požadavku. Na výběr je z 3 hodnot: 0 – Render fáze, 1 – Action fáze, 2 – Resource URL

p_p_state	Stav okna portletu dle JSR-168 - normal, maximized nebo minimized.
p_p_mode	Mód portletu dle JSR-168 - view, edit nebo help.
p_p_col_id	ID sloupce v použitém layoutu.
p_p_col_pos	Vertikální pozice ve sloupci.
p_p_col_count	Počet sloupců v layoutu.
[p_p_id].jspPage	Další parametry lze specifikovat pomocí prefixu id portletu. V tomto případě určujeme jakou JSP stránku chceme použít.

Převzato a upraveno z [Sezov, 2011]

Adresy v portálu lze generovat několika způsoby. Mezi nejspolehlivější a nejčastější patří generování přes JSP tag (viz kapitola 14.1) či vygenerování v třídě controller přes aktuální objekt response pomocí metod create{Resource|Action|Render}Url(). URL lze také generovat skrze JavaScript za využití objektu AUI.

```
AUI().ready("liferay-portlet-url", function(a) {
    var resourceUrl = Liferay.PortletURL.createResourceURL();
    resourceUrl.setPortletId("newnoteportlet_WAR_newnoteportlet");
    resourceUrl.setResourceId("saveNote");
    saveNoteUrl = resourceUrl.toString();
});
```

Kód 8 Tvorba URL adres přes JavaScript

Chceme-li zpřístupnit naši akci, data či obrazovku přes snáze zapamatovatelnou adresu, máme možnost využít tzv. Friendly URL. Díky nim lze adresu s mnoha často nedůležitými parametry transformovat do jednoduchého formátu, s možností parsování parametrů do příslušných proměnných pomocí speciálních výrazů a není nutné pro její tvorbu použít komponenty zmíněné výše. Pro tato mapování vytvoříme soubor ve zdrojové složce /resources s názvem {název}-friendly-url-routes.xml a zaregistrujeme ho do liferay-portlet.xml.

```
<friendly-url-mapper-class>
    com.liferay.portal.kernel.portlet.DefaultFriendlyURLMapper
</friendly-url-mapper-class>
<friendly-url-mapping>{zkrácený-název}</friendly-url-mapping>
<friendly-url-routes>{název}-friendly-url-routes.xml</friendly-url-routes>
```

Kód 9 Registrace mapovacího souboru

Pro získání například produktů na základě jejich názvu a za předpokladu, že máme vtagu `friendly-url-mapping` zkrácený název `products`, zavedeme do mapovacího souboru tag `route` níže. Po zaslání požadavku na URL `{aktuální-adresa}/products/getProduct/notebook`, bude zavolána metoda typu `resource` s názvem `getProduct` s příchozím atributem `productName` obsahující hodnotu `"notebook"`. Vrátil by se nám pravděpodobně seznam laptopů například ve formátu JSON. Můžeme zde také nastavit řadu implicitních parametrů, jež budou napevno posílány při každém volání adresy.

```
<route>
  <pattern>/getProduct/{productName}</pattern>
  <ignored-parameter name="p_p_cacheability" />
  <implicit-parameter name="p_p_resource_id">getProduct</implicit-parameter>
  <implicit-parameter name="p_p_lifecycle">2</implicit-parameter>
</route>
```

Kód 10 Mapování URL

7.6 Alloy UI

Alloy je frontendový framework založený na JS knihovně YUI a CSS knihovně Twitter Bootstrap. Poskytuje velké množství často používaných komponent a usnadňuje tím tak práci. Nalezneme zde nápovědy, modální okna, tlačítka, vyjížděcí nabídky, palety pro výběr barev, ořezy obrázků a mnoho dalších. Liferay je dodáván s touto technologií a ve svém jádru ji hojně využívá. Díky ní lze generovat komponenty nejen pomocí JavaScriptu, ale také za použití JSP tagů, integrující tyto styly a funkce. Liferay je v aktuální verzi dodáván s verzí Alloy UI 2, čímž je uzamčen s nyní již mrtvou knihovnou YUI a velice starou knihovnou Twitter Bootstrap 2. Ačkoliv v současné době existuje verze 3, povyšující alespoň CSS framework, v jejich prohlášení stojí kompletní přechod na jQuery a tvrdí, že aktuální verze není cílená na použití v Liferay 6.2 [Rocha, 2014].

7.7 Service Builder

Nástroj Service Builder (SB) slouží k usnadnění vytvoření low-level vrstvy. Hlavními technologiemi interně využitými jsou Spring a Hibernate. Budujeme-li novou aplikaci od základů, musíme se postarat o vytvoření doménového modelu s objektově relačním mapováním. Dalšími nezbytnými kroky je zajištění transakcí, správa session či navazování připojení k databázi. Postupem času optimalizujeme výkon a zpracováváme metody pro kešování výsledků. Ke každé entitě je nutné napsat CRUD operace a mnoho

elementárních, ale i složitých vyhledávacích metod v našich DAO (Data Access Object) vrstvách, jež nám vrací výsledky na dotazy. DAO vrstvu využívá zpravidla servisní vrstva, jež se stará o veškerou business logiku a sama je dostupná na mnoha místech aplikace pro pouhé nainjektování již z inicializovaného objektu. Těchto pár problémů pouze nastiňuje jaké možnosti Service Builder přináší v kontextu platformy Liferay. Ačkoliv je ve vývojovém prostředí označován jako další zásuvný modul, ve finále se jedná o běžný portlet s několika odlišnými konfiguračními soubory a předpřipravenou generovanou strukturou.

Celá komponenta je funkční pouze se správně nastaveným vývojovým prostředím, obsahující Liferay IDE, kterým je generování servisní vrstvy spouštěno. Po vytvoření Service Builder portletu dle průvodce dostaneme dva projekty. Při pojmenování projektu *service* dostanou název **service-portlet** a **service-portlet-service**.

Service-portlet-service je vygenerované JAR dle definice v *service.xml* a dalších pomocných xml v projektu service-portlet. Po spuštění nástroje Service Builder jsou vygenerovány předpisové třídy pro implementaci ve zdrojovém projektu a v cílovém projektu jsou vygenerovány všechny potřebné třídy pro volání, persistenci a další. Vygenerované metody obalují naši implementaci pomocným kódem, zajišťují low-level logiku, ale i samotnou inicializaci servis. Díky tomu máme v celé aplikaci přístup k naší servisní vrstvě. K metodám ze service-portlet modulu zvenčí projektu přistupujeme přes třídu `{EntityName}LocalServiceUtil` obsahující statické metody pro naše použití. Ačkoliv zdrojový projekt obsahuje mnoho tříd vztahujících se k vytvořené entitě, modifikovatelné jsou pouze tři [Yuan et al., 2009].

- **{EntityName}LocalServiceImpl** – implementace lokální servisní vrstvy dostupné ve stejné VM. Tato třída by měla sloužit pro práci se samotnou entitou – ukládání, modifikaci či mazání, ale měla by i obsahovat kompletní business logiku.
- **{EntityName}ServiceImpl** – implementace servisní vrstvy pro vzdálená volání, webové služby atd. Na tomto místě by měly být umístěny dodatečné kontroly oprávnění a provolány lokální servisy.
- **{EntityName}ModelImpl** – rozšíření doménové entity o vlastní implementaci

Ostatní třídy jsou generovány na základě definice v *service.xml* a dalších konfiguračních souborech ze zdrojové složky */resources*, kde ty důležité si nyní popíšeme.

- ***service.properties*** – obsahuje odkaz na externí *service-ext.properties* soubor, číslo, namespace, datum běhu nástroje service builder a důležitou položku pro automatické generování DB schématu. Dále obsahuje odkazy na konfigurační soubory pro Spring. Tento soubor by neměl být upravován.
- ***service-ext.properties*** – slouží pro vlastní modifikace původního nastavení.
- ***META-INF/portlet-model-hints.xml*** – nastavení dodatečného chování atributů v daných entitách. Lze zde nastavit jak chování z hlediska generování SQL, tak chování na straně klientské, při použití JSP tagu *liferay-ui*.
- ***META-INF/ext-spring.xml*** – tento soubor slouží pro vlastní definice Bean (nakonfigurovaná pojmenovaná instance třídy) v rámci Spring kontextu. Toto místo slouží například pro zaregistrování vlastního datového zdroje s objektem transaction manager a session factory instancí. Ostatní soubory ze *spring.configs* položky, by neměly být modifikovány, z důvodu jejich automatického generování. Po spuštění SB by byly naše úpravy smazány.

Nejdůležitější soubor se ukrývá ve složce */WEB-INF* a je to již několikrát zmiňovaný *service.xml*, jež řídí veškeré nastavení a generování. Konkrétní příklad vytvoření entity a servisy je k nalezení v kapitole 14.2. Tento soubor usnadňuje využívání principu servisně orientované architektury (SOA), jelikož nám kromě kompletního zaobalení doménových entit zprostředkuje i třídy pro webové služby SOAP. Je třeba dát si pozor na fakt, že schéma a modifikace nejsou zcela vždy generovány DDL skripty automaticky a je tak nutné, postarat se o své databázové schéma vlastními silami. V některých případech lze využít Hibernate položky v *portal-ext.properties* **hibernate.hbm2ddl.auto=update**, avšak vazby pomocí cizích klíčů či integritní omezení jsou na databázovém administrátorovi.

Nejjednodušší metoda v lokální servisní třídě pro založení nové poznámky může vypadat následovně.

```
public void createNote(NoteFormModel form) throws SystemException {
    Long noteId = super.counterLocalService.increment(Note.class.toString());
    Note note = super.createNote(noteId);
    NoteFormModelToNoteConverter.convertToNote(form, note);
    super.addNote(note);
}
```

```
}
```

Kód 11 Vytvoření entity v servisní třídě

Zde máme dostupné základní persistenční metody zděděné z předka. V případě, že bychom chtěli využívat specifické vyhledávací metody, mazání dle parametrů a další rozšíření spojené s persistencí lze využít objektu `notePersistence`, kdekoliv v aktuální třídě. V příkladu výše vstupuje do metody naplněný objekt `data` z view vrstvy v pomocné DTO třídě. Pomocí zabudované komponenty `counterLocalService` pro získávání ID vygenerujeme primární klíč a založíme novou entitu. Liferay poskytuje vlastní řešení pro inkrementaci nových ID, díky čemuž rozšiřuje kompatibilitu skrze různé databázové. Pro zvýšení výkonu je vhodné zvýšit počet ID udržovaných v paměti pomocí property `counter.increment` v *portal.properties*. V této chvíli ještě není poznámka persistována do DB. Převedeme data z DTO objektu do doménové entity a pomocí předpřipravené metody `addNote` uložíme entitu do DB. Kdekoliv v našich portletech, obsahující potřebnou závislost, můžeme jednoduše zavolat `NoteLocalServiceUtil.createNote(form)` bez nutnosti starání se o transakce, kešování nebo správu session.

8 Theme plugin Liferay

Celkový vzhled aplikace bývá jednou z nejdůležitějších věcí vnímající příchozí, ale i stávající uživatele. Liferay nabízí jednoduchou cestu jak změnit celkové vyznění portálu s možností personalizace. Theme je ideální místo pro umístění všech CSS stylů, globálních JS knihoven, obrázků či fontů používaných napříč portálem. Výchozí instalace je dodávaná s dvěmi takovými pluginy, ze kterých je možné dědit a není nutné tvořit celý vzhled a strukturu od základů, ale pouze přepisovat dílčí soubory. Při tvorbě kostry aplikace máme na výběr ze třech dynamických jazyků, jimiž jsou JSP, Freemarker a Velocity. Zde si přiblížíme jazyk Velocity, jež zastává největší procento ve všech užitých návodech a literaturách. Jeho syntaxe však není o mnoho odlišná od jazyka Freemarker.

Pomocí šablon v tomto projektu konstruuje stránku od kořenového HTML tagu po jeho konec. Lze zde kompletně modifikovat hlavičku s meta informacemi, veškeré importy skriptů a stylů, navigační panel jež dynamicky přidává položky podle nastavení v administraci, kontejner pro jednotlivé portlety, až po patičku dokumentu. Zmíněné kontejnery jsou poslední věcí, kam až se lze pomocí Theme pluginu dostat. Uspořádání se provádí pomocí dalšího zásuvného modulu popsáno v kapitole 9. Ačkoliv je možná téměř kompletní změna vzhledu v tomto projektu, na některé komponenty doplňující vzhled a funkcionality zde sáhnout nelze. Jedná se o prvky udávající fungování portálu jako takového. Administrační rozhraní pro přidávání stránek, portletů, obsahu, chybové hlášky, hlavní lišta s názvem dockbar a mnoho dalších. Díky tomu, že se jedná o JSP soubory v projektu ROOT, lze i tyto fragmenty v případě potřeby změnit pomocí pluginu Hook či Ext popsaných v dalších kapitolách.

Po založení projektu Theme, nenalezneme mnoho souborů ani složek. V případě, že jsme se rozhodli pro dědění z již existujícího pluginu Theme, máme v souboru pom.xml uvedenou variantu `_styled`. To nám zajistí zdědění kompletního vzhledu a umožní přepsat pouze potřebné soubory. Pokud by byla zvolena varianta `_unstyled`, bylo by nutné veškerá CSS dotvořit ručně. Ačkoliv se i zde nachází stejné množství CSS souborů, jedná se pouze o předpisové prázdné třídy pro vlastní vyplnění. Ostatní soubory jako obrázky či Velocity šablony jsou poskytnuty ve všech variantách. Vedle těchto dvou možností stojí ještě Theme typu Classic, což je jakýsi kompromis mezi již hotovým vzhledem s nízkou flexibilitou a vysokou rychlostí použití a nenastylovaným vzhledem umožňujícím kompletní možnost vlastního vzhledu s vysokou flexibilitou, avšak velkou náročností.

První kompilace vytvoří v cílové složce veškeré soubory zděděné z rodičovského Theme pluginu. V lokaci */webapp* pak založíme složky odpovídající struktuře v cílovém projektu. Soubory, které si přejeme modifikovat, lze převzít z rodičovského projektu a upravit dle libosti. Opětovná kompilace přepíše oproti děděnému projektu pouze soubory, jež jsme vložili do */webapp*. Typicky se zde nachází složky */templates*, */css*, */js*, */images*.

8.1 CSS

Kaskádové styly pokročily ve svém vývoji nejen v možnostech úpravy vzhledu elementů na stránce, ale i v samotném procesu tvorby programátory. CSS stylů bývá mnoho a dosud není v samotném standardu zavedena žádná možnost dědičnosti či jiného přepoužívání kódu bez nutnosti opakování se. Nelze zde tvořit funkční konstrukty nebo strukturovat soubory a bloky dle logické celistvosti. U velkých projektů je toto nemalý problém, kdy se kód stává špatně udržitelný a škálovatelný. Od aktuální verze 6.2 jsou všechny kaskádové styly psány ve vyšším jazyce SASS umožňující spolu s knihovnou Compass psát styly za využití výrazně méně kódu s mnohými funkčními možnostmi. Kompilace ze SASS do CSS probíhá při prvním požadavku na dané soubory. Ze strany klienta, proto bývá první načtení stránky po změně pomalejší. Vlastní styly by měly být situovány do souboru *custom.css*, avšak nic nebrání vytvoření vlastní struktury, na kterou bude odkázáno přes import notace. Výhodou tohoto souboru je jeho přednost před ostatními styly, tudíž pokud chceme přepsat cokoli z Liferay designu, je vhodné použít právě *custom.css*.

8.2 Look and feel

Theme projekt je dodáván se specifickým konfiguračním souborem *liferay-look-and-feel.xml*. Zde můžeme nastavit parametry vzhledu, jež budou například voleny dle společnosti, aktuálně používající portál. Můžeme mít například pro jednu organizaci modrý vzhled a pro druhou červený. Lze zde upravit cesty k našim zdrojům, jako jsou obrázky, JavaScript soubory či CSS. Dalším prvkem je bezpečnost, jež dovoluje specifikovat společnosti používající dané Theme. V neposlední řadě je možné vymezit řadu vlastních klíč-hodnot nastavení, ke kterým lze v našich dynamických šablonách přistupovat a na jejich základě podmiňovat vzhled a logiku. Ty lze navíc upravovat i v administračním rozhraní po přidání příznaku *configurable* na hodnotu *true*.

V následujícím příkladu je vidět nastavení pro dvě různé korporace využívající jiná nastavení, barevná schémata a obrázky. K nastavením lze v našich Velocity šablonách přistupovat pomocí proměnné `$theme`, která nabízí pro tyto účely metodu `getSetting(key)`.

```
<theme id="corporate1" name="Corporate 1">
  <root-path>/html/themes/corporate</root-path>
  <images-path>${root-path}/images1</images-path>
  <settings>
    <setting key="header-type" value="detailed" />
  </settings>
  <color-scheme id="01" name="Blue">
    <css-class>blue</css-class>
    <color-scheme-images-path>
      ${images-path}/color_schemes/${css-class}
    </color-scheme-images-path>
  </color-scheme>
</theme>

<theme id="corporate2" name="Corporate 2">
  <root-path>/html/themes/corporate</root-path>
  <images-path>${root-path}/images2</images-path>
  <settings>
    <setting key="header-type" value="brief" />
    <setting configurable="true" key="slogan"
      type="textarea" value="Lorem ipsum.." />
  </settings>
  <color-scheme id="02" name="Red">
    <css-class>red</css-class>
    <color-scheme-images-path>
      ${images-path}/color_schemes/${css-class}
    </color-scheme-images-path>
  </color-scheme>
</theme>
```

Kód 12 Dva odlišné vzhledy definované v souboru look-and-feel.xml

9 Layout plugin Liferay

Portál agreguje obsah, který je zprostředkován pomocí portletů. Jak ovšem určíme, kolik místa budou portlety zabírat na stránce, v jakém pořadí budou skládány nebo zda půjdou fragmenty přemísťovat? Ve výchozí instalaci máme na výběr několik layoutů od základního jednosloupcového, dvoj sloupcového s různými poměry šířek nebo různé variace rozdělení stránky na několik dílčích bloků. Často jsou tato rozložení pro naše specifické stránky nedostačující. Pro tyto účely slouží právě vytvoření vlastního layoutu. Jedná se téměř o nejsnazší komponentu, kterou lze v rámci zásuvných modulů vytvořit.

Šablona se označuje koncovkou *.tpl* a nalezneme ji na úrovni kořenové složky webové aplikace. S touto příponou se zde nachází soubory dva. Druhý z nich končící na *wap.tpl* je určen pro mobilní telefony a můžeme si proto dovolit seskupovat portlety například do jednoho sloupce v případě mobilní platformy a na desktopových stanicích, kde bývá širší obrazovka využít třísloupcové rozložení. Poslední soubor, jenž se zde nachází, je ikona našeho rozložení, která slouží pouze pro lepší identifikaci v administračním rozhraní. V zanořené složce */WEB-INF* nalezneme konfigurační soubor *liferay-layout-templates.xml*, jež definuje cesty k našim šablonám.

Kód šablony se skládá z několika blokových elementů se správně uvedenými třídami pro korektní vykreslení. Struktura i třídy se mohou s každou verzí Liferay měnit, a proto je nejsnazší způsob tvořit layouty podle již vytvořených v lokaci */ROOT/layouttpl/custom*, kde */ROOT* je umístění běžící instance portálu v našem webovém kontejneru.

Následující příklad zobrazuje dvojsloupcové rozvržení v poměru 2:1, kde druhý sloupec obsahuje na pevně umístěný portlet s ID *upcomingportlet*. Další aplikace lze přidávat do obou sloupců nad i pod fixní portlet. V šabloně lze využívat řadu dalších objektů vyjmenovaných ve wiki od [Swaim et al., 2013].

```
<div class="wall-layout" id="main-content" role="main">
  <div class="portlet-layout row-fluid wall-upper">

    <div class="portlet-column portlet-column-first span8" id="column-1">
      $processor.processColumn("column-1",
                              "portlet-column-content portlet-column-content-first")
    </div>

    <div class="portlet-column portlet-column-last span4" id="column-2">
      $theme.runtime("upcomingportlet_WAR_upcomingportlet")
      $processor.processColumn("column-2",
                              "portlet-column-content portlet-column-content-last")
    </div>
  </div>
</div>
```

```
</div>  
</div>
```

Kód 13 Dvousloupcový layout s fixně vloženým portletem

Ve většině případů využijeme pro správu a aranžování portletů administrační rozhraní kde pomocí Drag-and-Drop provedeme potřebné změny. Programově lze tyto úkony provést zrovna tak, avšak s dobrou znalostí layoutu, s kterým pracujeme. V případě, že využíváme zanořené portlety umožňující využít vlastní layout uvnitř layoutu, je situace ještě komplikovanější. Celé rozložení je ukládáno do textového políčka TYPESETTINGS uvnitř Liferay objektu třídy Layout. Zde je metodou klíč hodnota ukládán identifikátor layoutu, obsažené sloupce uvnitř a portlety v pořadí, jakém jsou ve sloupci vrstveny, případně dodatečné konfigurační parametry.

```
layout-template-id=1_column\u000acolumn-1=  
upcomingportlet_WAR_upcomingportlet,newnoteportlet_WAR_newnoteportlet,\u000a
```

Programové přidání portletu do aktuálního layoutu vypadá následovně. Z aktuálního požadavku je získán layout objekt, jenž je převeden na LayoutTypePortlet, který obsahuje potřebné metody pro práci s layoutem a jeho nastavením. Následně je portlet přidán do konkrétního sloupečku a objekt s několika povinnými atributy persistován do DB.

```
Layout layout = (Layout) request.getAttribute(WebKeys.LAYOUT);  
LayoutTypePortlet layoutTypePortlet = (LayoutTypePortlet) layout.getLayoutType();  
String portletId = layoutTypePortlet.addPortletId(userId, „portletId_WAR_webappcontext“,  
„column-1“, -1, false);  
LayoutLocalServiceUtil.updateLayout(layout.getGroupId(), layout.isPrivateLayout(),  
layout.getLayoutId(), layout.getTypeSettings());
```

Kód 14 Programové přidání portletu

10 Hook plugin Liferay

Rozšíření pomocí Hook pluginu realizuje dodatečné akce na základě jistých událostí nebo přepisuje stávající prvky implementované v ROOT aplikaci Liferay jako jsou JSP či properties soubory. Ve složce */WEB-INF* se nachází soubor *liferay-hook.xml*, obsahující soubory, jež chceme přepsat vlastními.

Následující úryvek přepisuje vybrané položky v lokalizačním souboru a vybrané položky z hlavního konfiguračního souboru. Dále definuje složku s vlastními JSP, které chceme přepsat. Do této složky vložíme soubory stejného názvu se stejnou adresářovou strukturou. Hook se v okamžiku nasazení postará o nahrazení původního souboru s vytvořením záložní kopie pro vrácení do původního stavu po odebrání tohoto zásuvného modulu. Nemusíme se proto bát, ztracení originálního obrazu portálu. V případě, že bychom chtěli modifikovat portál v rámci konkrétního portletu, je možné konfigurační soubor *liferay-hook.xml* vytvořit přímo v portletu a dále postupovat stejně, jako v případě autonomního Hook projektu.

```
<hook>
  <language-properties>Language.properties</language-properties>
  <language-properties>portal.properties</language-properties>
  <custom-jsp-dir>/META-INF/custom_jsps</custom-jsp-dir>
</hook>
```

Kód 15 Modifikace portálu pomocí Hook pluginu

10.1 Aplikační adaptér

Speciálním případem pluginu Hook, je aplikační adaptér, umožňující aplikovat danou změnu pouze pro vybrané Site nastavitelné přes administrační rozhraní. Příznak označující aplikační adaptér je obsažen v definici Hook pluginu. Jedná se o tag *custom-jsp-global* nastavený na hodnotu *false*. Jak již název napovídá, rozsah platnosti se vztahuje pouze na JSP soubory. Při psaní globálních Hook rozšíření pro JSP, je vytvořena záloha původního souboru s koncovkou *.portal.jsp*. Chceme-li tento soubor vložit do našeho vlastního JSP souboru, přejímající název originálního souboru, použijeme první variantu v příkladu níže. V případě aplikačního adaptéru je však nutné použít druhou variantu bez přidané koncovky *.portal* a s explicitně nastaveným atributem *useCustomPage* na hodnotu *false*, určující původ souboru. V případě aplikačního adaptéru je náš soubor pojmenován s koncovkou **{jméno_Hook_pluginu}.jsp* a je tedy nutné, rozlišit jaký název dostal originální portál soubor. Tento mechanismus umožňuje

zachování stávající funkcionality bez nutnosti přepisu všech závislých komponent a lze tak pouze doplnit či obalit fragment našimi změnami [Camarero, 2011].

```
<%@ taglib uri="http://liferay.com/tld/util" prefix="liferay-util" %>

<!-- Varianta pro globální Hook -->
<liferay-util:include page="/html/portlet/navigation/view.portal.jsp" />

<!-- Varianta pro aplikační adaptér -->
<liferay-util:include page="/html/portlet/navigation/view.jsp"
    useCustomPage="false" />
```

Kód 16 Zakomponování původního JSP portálu

Aktivace adaptéru je prováděna přes Control panels – Site administration – Configuration – Site settings. Nevýhodou je nemožnost úpravy JSP souborů ve složce */ROOT/html/taglib*, kde se nachází mnoho klíčových fragmentů.

10.2 Přepsání zabudovaných servisních tříd

Další možnosti úpravy Liferay portálu je reimplementace metod v zabudované servisní vrstvě spravující objekty dodávané s výchozí instalací. Patří k nim například uživatel, role, layout, organizace a další. Element *service-type* z příkladu níže definuje servisní třídu, kterou chceme přepsat a *service-impl* označuje naši implementační třídu dědící z obalující třídy zdrojové servisy (např. *UserLocalServiceWrapper*). Jelikož obalující třída pouze převolává metody z implementační vrstvy, lze v naší implementaci pouze přepsat potřebné metody, čímž efektivně upravíme často využívané komponenty.

```
<service>
    <service-type>
        com.liferay.portal.service.UserLocalService
    </service-type>
    <service-impl>
        net.evrem.hook.service.CustomUserServiceLocalImpl
    </service-impl>
</service>
```

Kód 17 Nahrazení interní servisní třídy

10.3 Pre a post akce

Liferay poskytuje několik událostí, při jejichž volání lze před nebo po, zavolat vlastní akci. Toto se hodí například před uskutečněním přihlášení uživatele pro dodatečné kontroly oprávnění či po odhlášení, kdy lze například uvolnit některá data spojená s aktuálním uživatelem. Akci je nutné zaregistrovat do souboru *portal.properties* ve stylu

`login.events.post=net.evrem.hook.LoginPostAction`, kde třída `LoginPostAction` dědí z `com.liferay.portal.kernel.events.Action` a implementuje jedinou metodu s dostupnými objekty `HttpServletRequest` a `HttpServletResponse`, se kterými lze právě na tomto místě pracovat.

10.4 Filtry

Další možností interakce s příchozími požadavky a odchozími odpovědi jsou servlet filtry. Ty lze namapovat na konkrétní URL, nadefinovat jaké filtry musí být zavolány před aktuálním filtrem a uvést samotnou referenci na obslužnou třídu s výkonným kódem. Ta musí implementovat rozhraní `javax.servlet.Filter`, poskytující metody `init`, `doFilter`, `destroy`. Pokud potřebujeme získat konfigurační statické parametry, využijeme metodu `init` s objektem `FilterConfig` obsahující námi nadefinované parametry ze servlet kontejneru. Metoda `doFilter` obsahuje hlavní logiku. Ve většině případů by zpracování mělo vypadat následovně.

1. Analyzuj požadavek
2. Libovolně doplň hlavičky a parametry požadavku/odpovědi
3. Libovolně obal vlastním chováním požadavek/odpověď
4. Deleguj požadavek do dalšího filtru nebo ukonči celý řetězec následných filtrů a přeruš aktuální zpracování

Po ukončení požadavku je zavolána metoda `destroy` pro vyčištění a uvolnění zdrojů před zničením instance filtru.

11 Ext plugin Liferay

V okamžiku kdy všechny předchozí typy pluginů při modifikaci portálu selžou nebo nejsou dostačující, můžeme se obrátit na Ext plugin. Ten nám umožňuje téměř kompletně přepsat Liferay portál nebo využít jeho jádro. Je nutné mít na paměti, že takovéto zásahy nepatří k běžným úpravám a **využívány by měly být co nejméně** a vždy je vhodné prověřit, zdali daná úprava nejde provést přes Hook plugin. Vývoj je v tomto případě na rozdíl od ostatních modulů výrazně složitější, zdlouhavější, avšak může umožnit například dočasnou opravu chyb v aktuální verzi Liferay, komunikaci s vnitřním API nebo obohacení klíčových tříd vlastním výkonným kódem. Nevýhodou je nutnost kompletní revize po migraci na novou verzi portálu. Důvodem je předpoklad změny v jejich struktuře vůči námi upraveným třídám. Jelikož se jedná o opensource produkt, bylo by možné stáhnout zdrojové kódy a požadované úpravy jednoduše zapracovat a zkompileované jar vložit na příslušná místa portálové instalace. Tímto úkonem bychom docílili toho samého, co lze přes Ext plugin, nicméně bychom nepoznali, co vše jsme v daném projektu změnili. Následná migrace, ladění chyb a obecně správa našeho projektu by se výrazně prodražila. Ačkoliv není separace našich změn stejně intuitivní jako v případě Hook pluginu, kdy je úprava nasazena v běhovém kontejneru jako další aplikace, práce s Ext je stále pohodlnější a kód je lépe oddělený, než v případě úpravy přes zdrojové kódy popsané výše [Chen et al., 2013].

Nejběžnější případy užití tohoto zásuvného modulu jsou:

- Přepsání *portal.properties*, které nejdou přepsat pomocí Hook pluginu
- Modifikování Struts akcí
- Vlastní implementace tříd deklarovaných v Liferay Spring souborech
- Přepsání JSP souborů uvedených v *portal.properties* a dalších, jež nejdou modifikovat přes Hook
- Přepsání tříd a konfiguračních XML souborů v jádře portálu

Ext projekt je složen z devíti podmodulů, přepisujících konkrétní části portálu. Nahrazující soubor musí být umístěn ve stejné lokaci v příslušném projektu jako originální soubor.

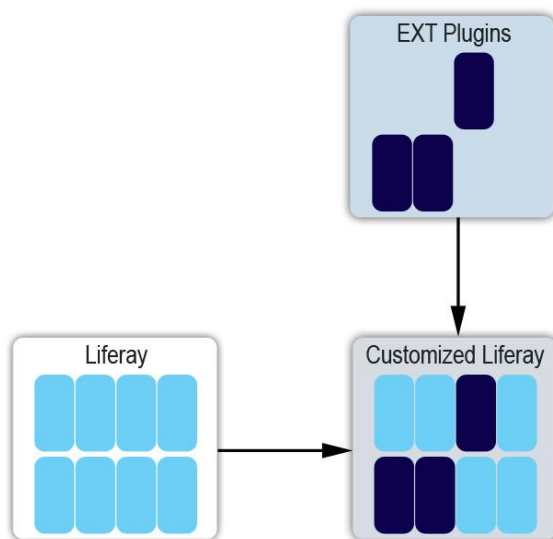
Struktura a účel projektů:

- **ext** – Hlavní webový projekt obsahující většinu ostatních závislostí. Po zkompilování je z tohoto projektu WAR soubor, určený pro nasazení na server, kde přepíše potřebné soubory v sobě obsažené.
- **ext-impl** – Tato lokace slouží pro přepis tříd a některých konfiguračních souborů. Často bývá tento projekt nejvíce využíváný, jelikož přepisuje zdrojový kód v *portal-impl.jar*, což je jádro portálu umístěné v */tomcat/webapps/ROOT/WEB-INF/lib*. Často využívané soubory jsou také *portal-ext.properties*, *system-ext.properties* nebo *Language-ext.properties*, přepisující položky v nastavení v Liferay jádru.
- **ext-lib-global** – Globální závislosti dostupné v rámci aplikačního/webového serveru.
- **ext-lib-portal** – Závislosti potřebné v portálové instalaci. Obvykle se jedná o závislosti využívané v našem Ext-impl modulu.
- **ext-service** – Závislosti, jež mají být dostupné ostatním pluginům. Obsahuje třídy generované nástrojem Service Builder, jež slouží především jako rozhraní pro volání. Projekt reflektuje strukturu *portal-service.jar* umístěném */tomcat/lib*.
- **ext-web** – Zde lze umístit vlastní JSP, HTML, JavaScript soubory, obrázky a mnoho konfiguračních XML souborů v celé instalaci portálu */tomcat/webapps/ROOT*.
- **ext-util-java** – Projekt, jehož protějškem je *util-java.jar* umístěném v */tomcat/webapps/ROOT/WEB-INF/lib*. Zde nalezneme mnoho pomocných JAVA tříd, využívaných napříč portálem a našimi pluginy.
- **ext-util-taglib** – Knihovna JSP tagů a pomocných tříd využívaných v naší view vrstvě. Strukturu a kompletní seznam třídy lze nalézt v *util-taglibs.jar* umístěném v */tomcat/webapps/ROOT/WEB-INF/lib*.
- **ext-util-bridges** – Posledním projekt implementuje životní cyklus portletů pro různé programovací jazyky. Možnosti, které poskytuje, nalezneme uvnitř *util-bridges.jar* umístěném stejně jako ostatní závislosti v */tomcat/webapps/ROOT/WEB-INF/lib*.

11.1 Nasazení pluginu

Ext plugin je ve způsobu použití odlišný od ostatních typů zásuvných modulů. Po kompilaci našeho projektu, dostaneme dva archivy webové aplikace WAR. Pokud

využíváme Liferay IDE, máme k dispozici nástroje pro zjednodušení celého postupu. I přesto je dobré vědět jak se s Ext pluginem zachází, z důvodů jeho provázanosti s naší instalací a nutností tato místa při vývoji navštěvovat a analyzovat.



Kód 18 Modifikace Liferay portálu skrze Ext plugin

Převzato z [Sezov, 2011]

1. WAR soubory překopírujeme do složky deploy, jež se stará o rozbalování archivů do `/tomcat/webapps` kde běží všechny naše aplikace.
2. Spustíme tomcat a necháme ho ziniclizovat a rozbalit naše Ext soubory.
3. Restartujeme tomcat, aby se projevíly změny v instalaci

Chceme-li postup opakovat, je nutné odebrat soubory, které jsme v předchozím kroku do instalace portálu nasadili. Bohužel toto nelze jednoduše přes odstranění aplikace z `/tomcat/webapps` jako v případě ostatních pluginů. Existují skripty, jež nám naši instalaci uvedou do původního stavu [Telcik, 2013]. Pokud ovšem aktualizaci neděláme často, lze vyčištění provést manuálně.

1. Odstraníme soubor `/tomcat/lib/ext/ext-{jméno_projektu}-ext-service.jar`
2. Odstraníme soubory z lokace `/tomcat/webapps/ROOT/WEB-INF/lib`
`ext-{jméno_projektu}--ext-util-taglib.jar`
`ext-{jméno_projektu}-ext-util-java.jar`
`ext-{jméno_projektu}-ext-util-bridges.jar`
`ext-{jméno_projektu}-ext-impl.jar`
`{jméno_projektu}-ext-util-bridges.jar`

{jméno_projektu}-ext-util-java.jar

{jméno_projektu}-ext-util-taglib.jar

+ Všechny závislosti uvedené v **ext-lib-portal** projektu

3. Odstraníme adresáře

/tomcat/temp

/tomcat/webapps/{jméno_projektu}-ext

/tomcat/webapps/{jméno_projektu}-ext-web

4. Odstraníme soubory */tomcat/webapps/ROOT/WEB-INF*

ext-{jméno_projektu}-ext.xml – tento soubor obsahuje všechny vlastní soubory

+ Další soubory použité v **ext-web** projektu

Závěrečným doporučením, kterým by se měl vývojář řídit je co nejvíce využívat mechanismu dědění z Liferay tříd a přepisovat pouze konkrétní metody způsobem, kdy vykonáme vlastní kód a provoláme metodu, jež přepisujeme pomocí `super.methodName()`. Tímto snížíme riziko ztráty funkcionality při přechodu na novou verzi na minimum. Postup lze ovšem aplikovat pouze u tříd, zaregistrovaných v konfiguračních souborech jako například Struts akce [Sezov, 2011].

12 Funkční popis aplikace

Aplikace Evrem slouží k upomínkování, organizování, tvorbu TODO listů, budování vlastní zdi s poznámkami nebo čistě jen jako kategorizovatelný zápisník. Zaměřuje se na uživatelskou přívětivost a moderní frontendové technologie. Segment trhu je zacílený na mladé osoby co hledají jednoduchost, pestrost a interaktivitu. Hlavní motivací je nezapomenout. Uživatel vloží upomínku s roční platností, a přestože aplikaci nebude aktivně využívat, v den události obdrží email s vlastnoručně nadefinovanými parametry. Ačkoliv podobné aplikace existují a to převážně na mobilních platformách, velmi často si data udržují pouze v interní paměti a v okamžiku ztráty či reinstalace telefonu jsou data ztracena. Vyšší funkce jako export dat bývá velice často podmíněn zakoupením licence. V této aplikaci jsou všechny funkce zdarma, pokud se uživatel rozhodne stáhnout svá data a aplikaci dále nepoužívat, provede export jednoduše do formátu *.xlsx tedy Excelu. Silnou stránkou aplikace je interaktivní zeď poznámek, kterou si uživatel může mechanismem Drag-and-Drop spravovat a zeď se mu transformuje za pomoci detekce kolize jednotlivých bloků. Díky portletové architektuře lze znovu využívat jednotlivé komponenty a v budoucí verzi nebude problém zajistit prémiovým uživatelům různé možnosti či uspořádání dle svých preferencí. V dnešní době již byla velká část nápadů realizována. Klíčem k úspěchu však zůstává finální provedení produktu a jeho propagace.

13 Výběr JS frameworku

Liferay ve své klientské vrstvě využívá framework YUI verze 3 od Yahoo, který tvoří základnu pro komponentovou knihovnu Alloy UI. Koncem srpna 2014 oznámilo Yahoo ukončení vývoje frameworku YUI. Důvodem bylo odchýlení od současných trendů. Ty jsou tvořeny single page aplikacemi, izomorfními aplikacemi tvořené pomocí Node.js, nástroji pro správu závislostí jakými jsou NPM či Bower, díky kterým lze celá klientská aplikace sestavit za pomoci úkolovacích nástrojů jako Grunt či Gulp a dalšími nezbytnými nástroji pro testování nebo pro transpilaci kódu, správu závislostí, oddělování scope (jmenný prostor s daným rozsahem platnosti) a mnoho dalších funkcí, které lze řešit nástroji jako Browserify nebo Webpack. Vývoj YUI trval téměř 10 let a přestože se jednalo o velice komplexní knihovnu, bylo jasné, že v současné podobě nemůže splňovat konkurenceschopnost a být kompatibilní se zmíněnými nástroji [Lecomte, 2014].

Velice populárním JS frameworkem v komunitě LFR je zcela bezkonkurenčně AngularJS. Je velice jednoduchý pro prvotní inicializaci a téměř hned dokážeme vytvořit model komunikující s template/view vrstvou (HTML elementy), díky mechanismu Two-way data binding. Ten nám zajišťuje propagování změn z view do modelu na straně klienta, ale i opačným směrem z modelu do view. Na počátku vývoje šetří tento přístup mnoho času a technických prostředků. Velice jednoduše lze přidávat nové komponenty controller (JS objekt s vlastním jmenným prostorem scope, navázaný na konkrétní HTML prvek), hierarchicky je zanořovat a následně z nich dědit. Každý controller má svůj vlastní scope, který nám odděluje jednotlivé jmenné prostory, čímž se nám uchová čistota v globálním jmenném prostoru. Není proto nutné využívat dalších nástrojů jako je CommonsJS pro vymezení jednotlivých prostorů a správu závislostí. Angular také nabízí velice účinnou, ale zároveň nebezpečnou zbraň, prolínající se jak JS vrstvou tak template vrstvou. Jedná se o funkci `$scope.watch(angularExpression, callback)`. Tato funkce zajišťuje kontrolu zadaného výrazu, kde při každé změně je zavolán callback se starou a novou hodnotou ve vstupních parametrech. Výhody jsou jasné. Jde zejména o rychlou reakci na konkrétní změny zadanou funkcí v typickém příkladu validací. Problém nastává v okamžiku, kdy proměnná scope obsahuje desítky instancí watcher a v callback funkci jsou měněny hodnoty, na kterých jsou opět navěšeny objekty watcher. Situace začne být velice nepřehledná a ladění chyb v takovémto systému může zabrat hodiny i dny. Two-way data binding nese další obrovskou daň v podobě Dirty checking

mechanismu, který kontroluje změny v celé hierarchii objektů controller respektive proměnných scope a reaguje na ně příslušnou aktualizací. Celý cyklus se volá vždy minimálně dvakrát, pro kontrolu, že byly všechny změny zpropagovány a jiná událost mezitím nezměnila hodnotu, již ověřené proměnné. Pro případ, že by se někde vytvořila smyčka v našich objektech watcher je nastaven limit opakování nad danou proměnou na 10 iterací. Výsledkem je velká výkonová náročnost, která způsobuje nemalé problémy i moderně vybaveným PC, především u větších aplikací [Panda, 2014]. Dalším problémem je velká učicí křivka při tvorbě direktiv a klíčových prvků nezbytných při vývoji.

13.1 ReactJS

Na problémy zmíněné v předchozím odstavci výborně odpovídá ReactJS s architektonickým vzorem Flux. Jedná se o relativně novou technologii pocházející z dílny Facebook. Nejen Facebook, ale i Instagram přešly na tuto technologii s relativně rychlým nasazením. Díky jednoduchosti rychle sklouzává do oblíby a povědomí dalších velkých firem. Jeho architektura poskytuje vysoký výkon, který je demonstrován například na staré hře Wolfenstein 3D. ReactJS skládá obrazovku z mnoha bloků s obrázky, generových jádrem Meteor a za běhu vyměňuje pouze potřebné segmenty. Další praktickou ukázkou může být desktopový textový editor Atom, který běží v prohlížečovém enginu, poskytující stejné funkce jako populární editor Sublime text opět napsaný v ReactJS.

Nesporná výhoda je jeho jednoduchost a krátká učicí křivka. Ačkoli se jedná o koncept poskytující možnosti složitých JS frameworků, jeho implementace je přímočará a přehledná. Tvorba komponent se stává základním stavebním kamenem, což přispívá k vysoké modularitě a znovupoužitelnosti. V aplikaci je využita nadstavbová syntaxe JSX, umožňující psaní HTML tagů přímo v jazyce JavaScript, zajišťující srozumitelnost, čitelnost kódu a hlavně zpřístupnění některých funkcí z ECMAScript 6. Po naučení několika málo funkcí a principů lze velice rychle tvořit komponenty, jejichž možnosti vysoce převyšují JSTL, kde lze jen velmi krkolomným způsobem vytvořit to samé co v ReactJS. Pokud navíc máme vysoce dynamickou aplikaci, kde je téměř každý element interaktivní nebo generován na základě modelu, zjistíme, že provázanost s JS stranou majoritním způsobem ovlivňuje výslednou stránku a statické tagy jsou zastíněny business logikou napsanou v JS. V takovémto případě dává smysl sjednotit oba koncepty do jedné roviny.

Základem ReactJS je využití virtuálního DOM (Document Object Model), který tvoří jakousi nadstavbu nad skutečným DOM. Jedná se o pouhé JavaScript objekty, které si drží stav a referenci vůči skutečnému DOM. To nám mimo jiné překlenuje rozdíly napříč různými prohlížeči a jejich verzemi. Implementace virtuálního DOM odladuje některé chyby v událostním (event) systému, které jsou známy ve starších prohlížečích. Hlavní výhoda je však ukryta v propagaci změn a aktualizaci pouze nezbytné měněné části, jelikož jakákoliv úprava skutečného DOM je v tomto ohledu velice náročná.

V každé vykreslovací fázi je heuristicky spočítáno, jaké nejmenší kroky musí být provedeny, aby se zpropagoval aktuální stav do reálného DOM. Logika vyhodnotí všechny uzly a hierarchicky níže postavené komponenty a překreslí potřebné části DOM. Tento úkon je proveden pouze jednou, díky virtuální podobě všech elementů, zjišťující všechny nezbytné kroky pro finální stav. Mechanismus se nazývá Reconciliation neboli Diff algoritmus. Minimum kroků potřebných pro transformaci jednoho stromu nodů je problém o složitosti $O(n^3)$ kde n je počet nodů ve stromu. Při 1000 nodech vykreslených ve stromu je zapotřebí miliarda porovnání pro finální transformaci. Takové množství operací nejsou nynější běžné procesory schopné zvládnout pod 1 vteřinu. React implementoval heuristicky založený algoritmus o složitosti $O(n)$, který vychází z následujících předpokladů:

1. Komponenty stejného typu generují podobné stromy a komponenty různých typů generují naprosto odlišné (`` je různý od `<div>`, `<DataTable>` je různý od `<Header>`).
2. Je možné elementům poskytnout unikátní klíč stabilní napříč překreslením.

Díky těmto předpokladům je docíleno rapidního zrychlení. Porovnávací algoritmus u elementů ověří, zdali jsou stejného typu a pokud ne, odstraní starý prvek a vloží nový. Pokud jsou stejného typu, zkoumá odlišnosti ve vnitřní struktuře a nahradí tak například pouze hodnotu v atributu `class`, namísto překreslení celého stromu potomků. V případě React komponent, které jsou stavové, máme k dispozici události `component[Will/Did]ReceiveProps`, které jsou při překreslování volány a je již na nás jakým způsobem budeme interagovat s nově přijímanými daty. Ačkoliv se nejedná o stoprocentně kompatibilní algoritmus aplikovatelný na všechny případy, výrazně urychluje většinu změnových operací nad DOM. V případě nedostatečného výkonu, knihovna nabízí další nástroje a mechanismy pro jeho optimalizaci [Chedeau, 2013].

13.2 Flux

Flux je architektonický vzor, jenž ve svém kódu využívá Facebook. Nejedná se o knihovnu, ale o pouhý přístup jakým způsobem psát klientský kód, tak aby byl přehledný a dobře strukturovaný. Stojí na principu Unidirectional flow, které těží z jednotného postupu akcí.

Všechny akce vznikají ve View nebo na serveru, ať už se jedná o uživatelskou interakci, počáteční inicializaci či jinou událost vyvolávající změnu. Při jakékoliv akci přesahující platnost komponenty volají vrstvu Actions. Ta představuje pomocnou vrstvu pro vydefinování všech dostupných akcí. Zde specifikujeme parametry, typ a unikátní označení konstantou v kontextu komponenty application dispatcher.

```
registerNotes: function(notes) {  
    AppDispatcher.handleViewAction({  
        actionType: FilterConstants.REGISTER_NOTES,  
        notes: notes  
    });  
}
```

Kód 19 Definice akce pro dispatcher

Dispatcher se stará o delegování všech akcí na příslušné callback funkce zaregistrované ve Store vrstvě. V případě, že využíváme více komponent typu Store, dispatcher deleguje akci na všechny takové komponenty. Obslužná činnost bude zavolána, pouze pokud je zaregistrována prostřednictvím switch konstrukturu rozlišených dle typu akce (unikátní konstanta). Dispatcher také umožňuje čekání na vykonání jiné akce. V některých případech potřebujeme zavolat akci teprve, až po dokončení jiné například ve více různých Store.

Store je jakési úložiště dat a zároveň vrstva operující nad těmito daty. Komunikuje se serverem a zajišťuje konzistenci a jednotný obraz dat v celé aplikaci. Po vykonání callback funkce je odeslána změna a komponenty, poslouchající na danou událost jsou překresleny s aktuálními daty. Do Store vede cesta vždy přes Dispatcher respektive přes Actions. Není jiná možnost jak změnit data v této singleton struktuře. Naopak získávání dat ze Store je umožněno pomocí veřejných metod odkazujících na privátní proměnné uvnitř JS souboru, na něž si drží referenci pouze objekt Store. Čerstvá data zpravidla konzumuje controller-view, což je hierarchicky nadřazená komponenta poslouchající na globální událost změny (change event.), na jehož pokyn aktualizuje svůj

vnitřní stav a překreslí všechny hierarchicky níže postavené komponenty s aktuálními daty předávanými skrze props (atributy tagu)[Wheeler, 2014].

```
function getNotesState() {
    return {notes: FilterStore.getNotes()};
}

var FilterContainer = React.createClass({
  getInitialState: function() {
    return getNotesState();
  },
  componentDidMount: function() {
    FilterStore.addChangeListener(this._onChange);
  },
  componentWillUnmount: function() {
    FilterStore.removeChangeListener(this._onChange);
  },
  render: function(){
    return (<DataGrid notes={this.state.notes} />);
  },
  _onChange: function() {
    this.setState(getNotesState());
  }
});
..
//Nadřazená komponenta zavolá vykreslení controller view do příslušného DOM elementu
React.renderComponent(<FilterContainer />, document.querySelector('#filter-container'));
```

Kód 20 Flux controller-view

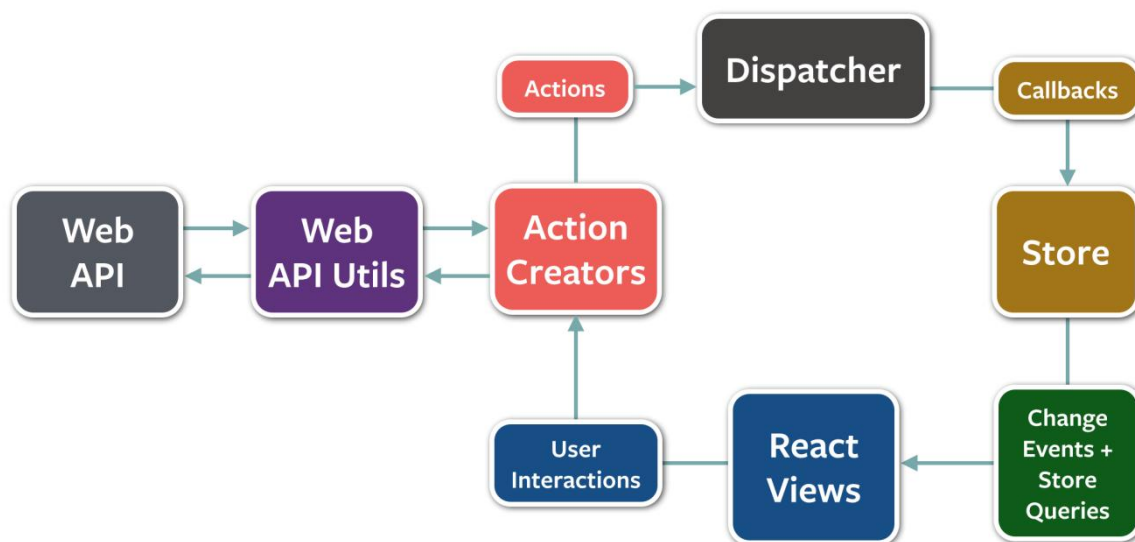
```
//Privátní část - definice modelu, interních metod
var _notes = {};
//Veřejný interface pro konzumaci modelu
var FilterStore = merge(EventEmitter.prototype, {
  getNotes: function(){
    return _note;
  },
  emitChange: function() {
    this.emit(GlobalConstants.CHANGE_EVENT);
  },
  addChangeListener: function(callback) {
    this.on(GlobalConstants.CHANGE_EVENT, callback);
  },
  removeChangeListener: function(callback) {
    this.removeListener(GlobalConstants.CHANGE_EVENT, callback);
  }
});
//Registrace obslužných metod pro změnu modelu
AppDispatcher.register(function(payload) {
  var action = payload.action;

  switch(action.actionType) {
    case FilterConstants.REGISTER_NOTES:
      _notes = action.notes;
      break;
    default:
      return true;
  }

  FilterStore.emitChange();
  return true;
});
```

Kód 21 Flux store

Celý princip elegantně zpřehledňuje všechny události, které se v aplikaci dějí a je velice snadné v takovéto struktuře odladovat chyby a škálovat do větších celků. Dále se tímto vyvarujeme nepředvídatelnému sledu akcí a jejich provázanosti, které nám v případě two-data binding systému způsobují nemalé starosti [Steigerwald, 2014].



Obrázek 8 Unidirectional flow

Převzato z [Facebook, 2015]

14 Architektura aplikace Evrem

Před započítím vývoje je nutné položit si otázky, zda tvoříme portlety pro Marketplace tedy širokou veřejnost, na kolik budeme využívat zabudované funkce portálu a hlavně jak budou vypadat a co mají umět. Je důležité ujasnit si vyhlídky do budoucna, jelikož je i Liferay každým rokem technologicky dál a dál. Snažit se využívat pouze technologie, které nyní využívá stabilní verze, by nás mohlo omezovat ve využití plného potenciálu všech dostupných frameworků na trhu a za rok bychom mohli zjistit, že je nová verze Liferay již využívá a my bychom zůstali uzamčení se zastaralými technologiemi, jelikož migrace našeho projektu na jiné, by pravděpodobně byla již nemožná. Pokud víme, že je pro nás některá knihovna zcela klíčová a potřebujeme sledovat a pravidelně aktualizovat její verze, je na místě použít oficiální zdroj, který je možné kdykoliv vyměnit či aktualizovat.

Příkladem může být nevyužití SASS a Compass, jež v aktuální verzi Liferay nepodporují mnoho funkcí oproti oficiální verzi. Dalším prvkem může být zvolení serverových knihoven jako Spring web MVC, usnadňující práci s požadavky/odpovědi v prostředí portletového kontejneru. Použití populárních tříd typu controller, kde lze snadno mapovat příchozí požadavky a parametry prostřednictvím anotací, ale i využít aplikační kontext pro inicializaci Bean a dalších prostředků. Počáteční čas pro nastavení a pochopení při integraci těchto nástrojů může být delší, jelikož chybí oficiální dokumentace, avšak v dlouhodobém horizontu, výhody silně převyšují nevýhody setrvání s výchozími nástroji.

14.1 Klientská část aplikace

V dnešní době již prakticky nenalezneme moderní aplikaci bez použitého jazyku JavaScript. Není divu, že velká část aplikace je napsána právě v něm. JSP je využito pouze při vykreslovací fázi, na inicializaci počátečních dat do příslušného objektu Store, kam patří i URL generované přes taglib portlet.

```
<portlet:resourceURL id="saveCoordinates" var="saveCoordinatesURL" />
```

Kód 22 Tvorba URL v JSP

Toto je v této aplikaci jediný účel JSP, ostatní funkce a tělo portletu je vygenerováno přes ReactJS do prázdného kontejneru ve view. Každý portlet má zpravidla pouze jedno view (*view.jsp*) a nejdůležitější soubor **bundle.js*. Tento JS soubor

obsahuje veškerou logiku a šablonu daného portletu napsané v JS. Toto odstínění od Liferay způsobu psaní, nám umožňuje vysokou variabilitu vzhledu, interaktivity, ale i propojení s moderními frontendovými knihovnami. Psaní logiky přes scriptlety v JSP neshledávám vhodné v aplikaci vykazující vysokou interaktivitu a dynamiku. Tento princip není u moderních aplikací považován za „best practice“, ačkoliv pro využití Liferay funkcionalit bývá velmi často nezbytným. Ztráta portability napříč portály není v tomto projektu důležitá.

Pro účely aplikace, jejíž převážná část vyžaduje propojení s JS je na místě použít technologii, která nepoužívá oddělenou šablonu od JS, nýbrž prolíná oba koncepty dohromady. Kód níže zobrazuje počáteční inicializaci portletu, kdy v okamžiku kdy je portlet načten a připraven, zaregistrujeme data do Store a vykreslíme hlavní komponentu, obsahující samotné view portletu.

```
/** @jsx React.DOM */
var UpcomingActions = require('./actions/upcoming-actions');
var UpcomingContainer = require('./components/upcoming-container.jsx');

Liferay.Portlet.ready(
  function(portletId, node) {
    if(portletId.indexOf('upcomingportlet_WAR_upcomingportlet') !== -1){
      var initialData = JSON.parse(jQuery('#data-upcoming').text());
      UpcomingActions.registerUrls(initialData.urls);
      UpcomingActions.registerNotes(initialData.notes);

      React.renderComponent(<UpcomingContainer />,
        document.querySelector('#upcoming-container'));
    }
  }
);
```

Kód 23 Inicializace view vrstvy

Jak již bylo zmíněno u architektury Flux, všechny akce prochází z View, Actions přes Dispatcher až do Store, kde v případě serverové akce je zaslán AJAX požadavek, který je odchycen specifickou instancí třídy Spring web MVC controller, vracejícím odpověď. Cílová URL adresa je použita z počáteční inicializace, tedy adresa obsahující všechny náležitosti potřebné k identifikaci portletu, metody, informaci o kešování, layoutu či módu portletu.

Portlety sdílejí společné JS knihovny a CSS styly prostřednictvím Theme. Základní proměnné obsahující reference na knihovny, jsou přístupné z globálního window scope, čímž je snížena velikost výsledných JS balíčků. Závislosti se neduplikují a soubor *bundle.js* obsahuje pouze kód související s portletem.

Ačkoliv Liferay umožňuje použití SASS a Compass vestavěném přímo v Liferay IDE, zde je využita vlastní struktura a kompilátor, který umožňuje použití nejnovějších funkcí v CSS. Toto odstínění nám umožňuje využívat vlastní adresář pro všechny frontendové soubory, které musejí projít zpracováním a teprve poté jsou rozkopírovány do příslušných umístění a hlavně použití nejnovějších funkcí.

V případě CSS je veškerý vlastní kód zkompileován do souboru *custom.css*, což je ve výchozím nastavení jediný soubor, kam by měly být vkládány uživatelské styly. Do Theme jsou balíky zkopírovány po provedení kompilace, minifikace a sloučení do jednoho souboru. Tyto úkony jsou zajišťovány pomocí správně nakonfigurovaného úkolovacího manažera Gulp, běžícího na Node.js.

Společně s nástrojem Webpack, jehož velkou výhodou je integrovaný CommonsJS jsou obstarávány balíky pro JS část. Může být nastaven pro práci s různými typy souborů, což zajišťuje tzv. loader. Pomocí výrazu řekneme, že soubor s danou koncovkou, má být předkompilován příslušným nástrojem loader a teprve poté jsou zpracovány závislosti, které jsou v kódu využity a pokud se nejedná o globální závislosti speciálně vydefinované jako externals, jsou přibaleny do výsledného balíku a jsou pouze dostupné v daném rozsahu. Ve výsledku máme čistější globální scope s oddělenými prostory.

Gulp nám dále zpracovává CSS styly respektive nadstavbu SASS, která nám umožňuje dynamiku při psaní stylů, znovu použitelnost, definování proměnných, jednoduché funkce, ale i hierarchické zanořování. Ruku v ruce s ním jde Compass, což je funkční knihovna, pro psaní kompatibilních stylů napříč prohlížeči. Nemusíme se tak starat o psaní moderních stylů pro všechna prohlížečová jádra. Jednoduše použijeme Compass funkci, která při kompilaci vygeneruje všechny styly pro dostupné prohlížečová jádra. Použití tohoto nástroje je podmíněno nainstalováním běhového prostředí Ruby s gemy Compass a Sass. Ukázkový kód níže znázorňuje dědičnost, použití funkcí či vlastních proměnných.

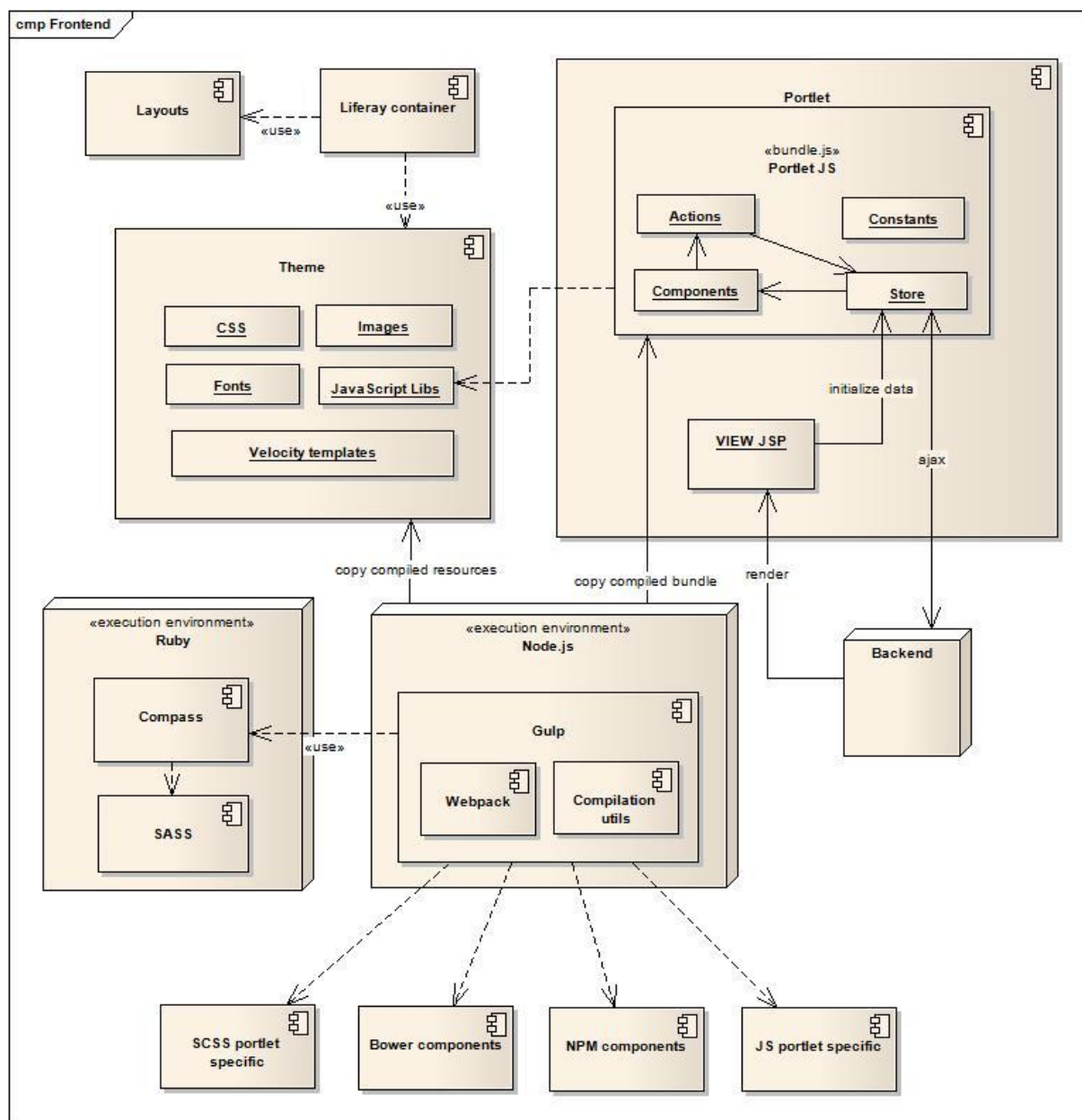
```
#isdone-input{
    @extend .main-input;
    @include user-select(none);
    @include hover-transition;
    &.isdone-checked{
        background-color: $greenColor;
    }
    &.isdone-unchecked{
        background-color: $rudeColor;
    }
}
```

Kód 24 Compass a SASS použití

Gulp se v neposlední řadě stará o správu závislostí z Bower a NPM. Tyto populární balíčkovací manažeři nám poskytují standardní způsob komponování a distribuci knihoven, což je výhodné pro jednotné použití. Bower se liší od NPM hierarchií závislostí, která je plochá a optimalizovaná pro vysoký výkon, zatímco NPM má závislosti zanořené a může obsahovat několik verzí knihoven. Jak již z krátkého popisu vyplývá, oba dva jsou profilovány pro jiný účel, ačkoliv je mnoho modulů dostupných v obou repositářích. Dalo by se říci, že Bower poskytuje závislosti ucelené přímo pro klienta, kdežto NPM počítá s následnou kompilací.

V Theme pluginu je využit šablonovací systém Apache Velocity s výchozí strukturou zděděnou z nadřazeného Theme. V aplikaci je využit Hook pouze pro drobné modifikace původního kódu jako je například JSP s licenčními podmínkami nebo zamezení přístupu do ovládacích panelů běžným uživatelům. V šablonách je dobré dbát veliké opatrnosti. Pokud smažeme nějaký zdánlivě nedůležitý prvek, můžeme se připravit o některé zabudované funkcionality, proto doporučuji co nejmenší zásahy v těchto souborech. Nevyhnul jsem se však například úpravě navigačního panelu, o ikonu profilu s vlastní dropdown funkcí, přidání dalších knihoven do hlavičky (nová Google reCaptcha) nebo skrytí některých funkcí nepřihlášenému uživateli.

Celé schéma klientské části je znázorněno na diagramu níže.



Obrázek 9 Schéma klientské části

14.2 Serverová část aplikace

Serverová část je rozdělena do několika modulů. Pro komunikaci s klientskou částí využíváme portlety, které obsahují logické celky zajišťující konkrétní činnosti. Portlety obsahují třídy controller, jež mají namapované metody na ID příchozí akce. Ve většině případů je komunikace zajišťována prostřednictvím AJAX volání typu Resource. Typický controller je vidět níže – při render fázi (překreslení obrazovky) jsou dotažena data, jež jsou vložena do modelu a je vrácena hodnota odpovídající názvu JSP. Na frontendu je vykreslena obrazovka se získanými daty z modelu. Uživatel založí novou poznámku a po stisknutí tlačítka uložit je zavolána metoda typu Resource, která přijme data ve formátu

JSON. Deserializuje je do DTO objektu a uloží prostřednictvím servisní vrstvy. Po dokončení vrátí v proměnné payload aktuální poznámku, případně chybové hlášky v serializovaném stavu JSON.

```
@Controller
@RequestMapping(value = "VIEW")
public class NewNoteController {

    @RequestMapping
    public String view(RenderRequest request, ModelMap map) {
        //.. getData ..
        map.addAttribute("periods", JsonSerializer.toJson(periods));
        map.addAttribute("colors", JsonSerializer.toJson(colors));
        return "view";
    }

    @RequestMapping("saveNote")
    public void saveNote(@RequestParam("jsonNote") String jsonNote, ResourceRequest request,
        ResourceResponse response) {
        NoteFormModel note = convertToFormModel(jsonNote);
        try {
            note = NoteLocalServiceUtil
                .saveNote(note, PortalUtil.getUserId(request));
        } catch (Exception e) {
            log.error("Error during saving note", e);
        }
        AjaxResponse<NoteFormModel> resObject = new AjaxResponse<NoteFormModel>();
        resObject.setPayload(note);
        response.getWriter().write(resObject.toJson());
    }
}
```

Kód 25 Spring MVC controller – render a resource požadavek

Pro sdílené akce je vytvořen commons-function-portlet, který je staticky umístěn, na každé stránce portálu ve skryté podobě. Nastavení statického portletu provedeme pomocí položky `layout.static.portlets.all` v *portal-ext.properties* souboru, kde jako hodnotu vložíme Fully Qualified Portlet Id nebo kompletní unikátní název (`portletId_WAR_webapplicationcontext`). Tento portlet dále obsahuje časované rutiny, s využitím zabudovaného řešení v Liferay. Jednoduchým přidáním položky v *liferay-portlet.xml* spouštíme rutinu dle vydefinované časové prodlevy nebo Cron výrazu. Jediné co musí třída časovače (scheduler) splňovat je implementace rozhraní `MessageListener`.

```
<scheduler-entry>
  <scheduler-event-listener-class>
    net.evrem.portlet.commonsfunction.scheduler.NotificationScheduler
  </scheduler-event-listener-class>
  <trigger>
    <simple>
      <simple-trigger-value>5</simple-trigger-value>
      <time-unit>minute</time-unit>
    </simple>
  </trigger>
</scheduler-entry>
```

```
</trigger>
</scheduler-entry>
```

Kód 26 Nadefinovaný Liferay scheduler

V automaticky spouštěných rutinách potřebujeme téměř vždy přistupovat k veškerým datům. Liferay neobsahuje žádnou informaci o tom, že by měl mít časovač práva do celé servisní vrstvy potažmo ke všem datům. Takovéto nastavení je nutné provést explicitně. Uživatelské ID administrátora máme uloženo v *portal-ext.properties*, pomocí něhož si získáme objekt uživatele a všechna jeho oprávnění nastavíme vláknu, pod kterým rutinu spustíme.

```
@Override
public void receive(Message message) throws MessageListenerException {
    User user = UserLocalServiceUtil.getUserById(
        Long.valueOf(PropsUtil.get("evrem.admin.userid")));
    PermissionChecker permissionChecker = PermissionCheckerFactoryUtil.create(user);
    PermissionThreadLocal.setPermissionChecker(permissionChecker);

    runMyJob();
}
```

Kód 27 Programové přidání administrátorských práv danému vláknu

Pro snazší přístup ke sdíleným funkcionalitám má commons-function-portlet nastaveny Friendly URL. Díky tomu není nutné generovat adresy taglibem v JSP, ale lze je jednoduše tvořit připojením ***/-/common/resourceId?parametry*** za aktuální adresu.

Další společný modul je tvořen projektem commons, poskytující pomocné třídy, konstanty, konvertory, třídy pro emailování spolu s Velocity šablonami a třídy pro exporty dat do Excelu skrze Apache POI.

Předávání dat mezi veřejnými portlety a service portletem má na starosti projekt Dtos. Zde jsou nadefinovány DTO (Data Transfer Object) objekty pro přenos dat mezi různými vrstvami uvnitř aplikace. Z frontendu přijmeme data ve formátu JSON, která odpovídají strukturálně modelu formulářů. Pomocí technologie Jackson deserializujeme řetězec do DTO a to předáme do servisní vrstvy. Je nutné, aby projekt respektive výsledné JAR s DTO objekty, bylo umístěno pouze ve složce ***/lib/ext*** ve webovém kontejneru pro zachování jediné definice třídy ve stejném kontextu jako *service-portlet-service.jar*. Pokud by každý portlet obsahoval závislost na tomto jar pak by předávání mezi různými třídami ClassLoader skončilo výjimkou *ClassNotFoundException*. Při volání service portletu, který je uložen taktéž v ***/lib/ext***, příslušný ClassLoader nehledá

závislosti uvnitř nadeployovaných aplikací (portletů), nýbrž v jeho vlastním kontextu webového kontejneru čerpajícím knihovny právě ze zmíněné složky */lib/ext*.

Servisní vrstva generována přes Service Builder, je defaultně nastavena na datový zdroj *LiferayPool*, který obsahuje všechna data o uživateli, layoutech, společnostech, právech, rolích a mnoho dalších. Jedná se o obrovské schéma čítající přes 180 tabulek. Z výkonnostních důvodů, ale i lepší správy a separace, je využít vlastní datový zdroj. Liferay tuto možnost nezakazuje, avšak primárně je uzpůsoben pro využití jednoho databázového schématu. Využití jiného datového zdroje je ne zcela přímočarou záležitostí. V service-portlet projektu je nutné v *ext-spring.xml* nadefinovat vlastní instance transaction manager, session factory a v našem případě JNDI datový zdroj, který je vydefinován v *context.xml* webového kontejneru. Transaction manager je aspektem aplikován na naše servisní třídy. Tyto Bean objekty je nutné zapsat ke každé entitě generované přes Service Builder v souboru *service.xml*.

V příkladu níže jsou znázorněny dvě entity, první z nich je plnohodnotná doménová entita s nadefinovaným databázovým zdrojem, session factory a manažerem transakcí. Druhá entita je pouze servisní vrstvou a proto u ní SB negeneruje třídy spojené s persistencí.

```
<!-- Doménová entita se servisní vrstvou -->
<entity name="Note" local-service="true" remote-service="false"
  table="note" data-source="evremDataSource"
  session-factory="evremSessionFactory" tx-manager="evremTransactionManager">

  <column name="noteId" type="long" primary="true" db-name="note_id" />
  <column name="text" type="String" db-name="text"/>
  <!-- ostatní sloupce -->
  <finder name="RemindDate" return-type="Collection"
    where="hasReminder=1 AND isDeleted=0">
    <finder-column name="userId" />
  </finder>
</entity>
<!-- Pouze lokální servisní vrstva -->
<entity name="SupportNote" local-service="true" remote-service="false"></entity>
```

Kód 28 Entity definované v service builder definici

Service builder se mimo jiné stará o kešování entit z databáze a výsledků vyhledávání. V některých případech se nám toto nastavení nehodí. Máme možnost napevno nastavit kešování v definici entity pomocí atributu *cache-enabled* nebo lze explicitně přepsat v souboru *service-ext.properties* ve formátu:

```
value.object.entity.cache.enabled.net.evrem.service.model.Note=false
```

```
value.object.finder.cache.enabled.net.evrem.service.model.Note=false
```

Zasílání emailů je realizováno přes zabudované API, poskytující zjednodušené odesílání na základě konfigurace spravované přes administrátorské rozhraní. Ačkoli mnoho návodů uvádí konfiguraci přes *portal-ext.properties* od Liferay verze 6.2, se nastavení z tohoto souboru, nahraje pouze při počátečním spuštění. Poté je již veškerá správa možná pouze přes ovládací panel. Email lze odeslat pomocí následujících dvou řádků kódu. První dva parametry jsou objekty z `javax.mail` `InternetAddress`, textový řetězec předmět, text emailu a rozlišení zda se jedná o HTML formát či strohý text. Použití je zde velice jednoduché. Velkou nevýhodou je ovšem nemožnost zasílání obrázků z důvodů chybějící podpory pro `MimeMultipart`.

```
MailMessage mailMessage = new MailMessage(from, to, subject, body, true);
MailServiceUtil.sendEmail(mailMessage);
```

Kód 29 Zasílání emailů přes zabudované komponenty

14.2.1 Apache Maven

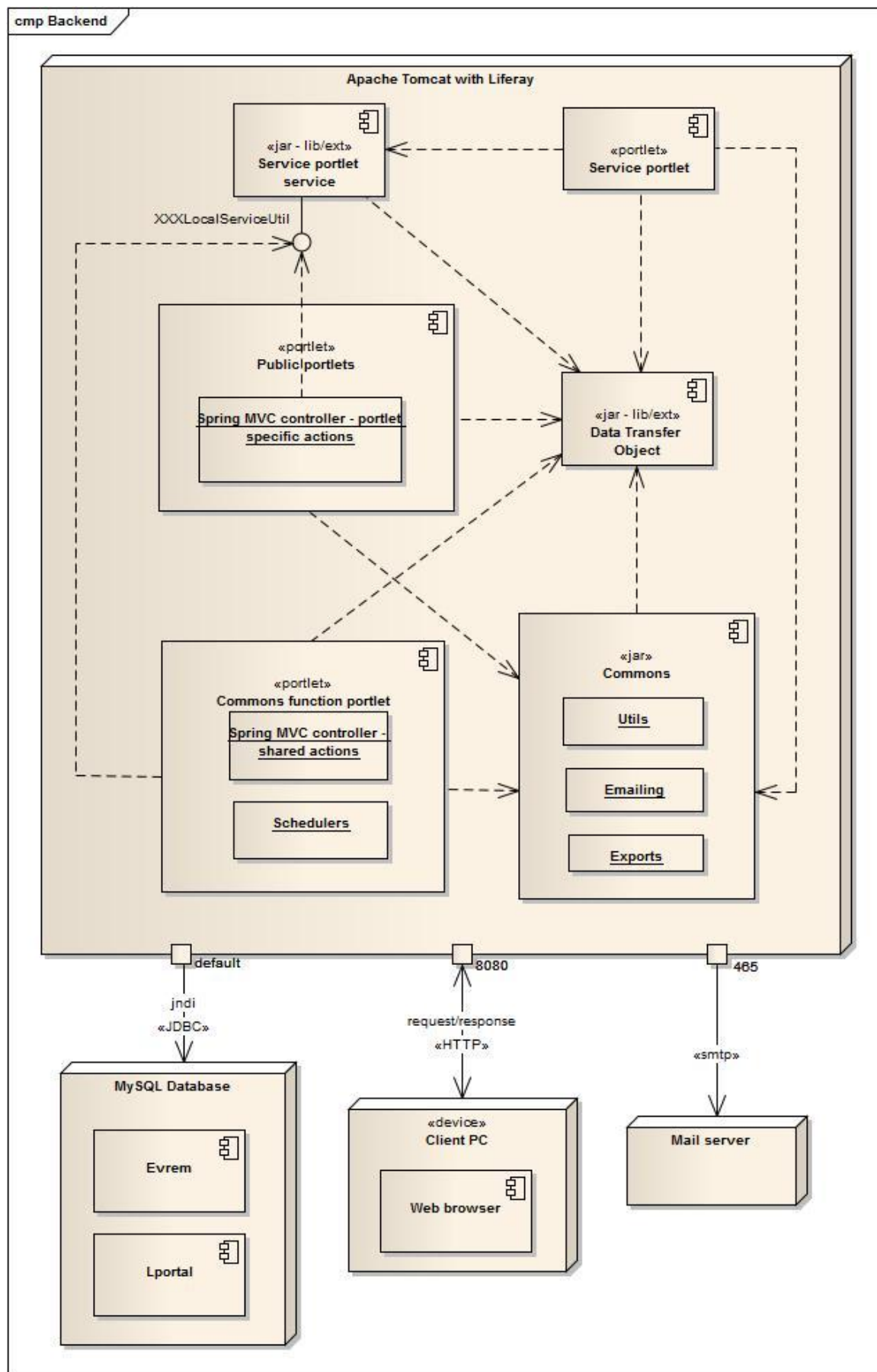
Nástroj pro správu projektu Apache Maven jsem si zvolil kvůli jeho popularitě a předchozí zkušenosti. Pro správné fungování je třeba nastavit cesty ke klíčovým složkám v adresáři staženého balíčku Liferay s vybraným webovým kontejnerem. Dále je nutné nastavit Maven repositáře, ve kterých se nachází potřebné závislosti. Tato nastavení náleží spíše k danému vývojáři než k projektu, proto jsem je zavedl do konfiguračního souboru *settings.xml*, který slouží jako globální nastavení napříč projekty a není tak nutné uvádět proměnné jako je lokace běhového prostředí do *pom.xml*, což je konfigurační soubor projektu ve vztahu k nástroji Maven.

```
<liferay.version>6.2.1</liferay.version>
<liferay.maven.plugin.version>6.2.1</liferay.maven.plugin.version>
<liferay.auto.deploy.dir>c:\liferay\deploy</liferay.auto.deploy.dir>
<liferay.app.server.dir>c:\liferay\tomcat</liferay.app.server.dir>
<liferay.app.server.deploy.dir>c:\liferay\tomcat\webapps</liferay.app.server.deploy.dir>
<liferay.app.server.lib.global.dir>c:\liferay\tomcat\lib\ext</liferay.app.server.lib.global.dir>
<liferay.app.server.portal.dir>c:\liferay\tomcat\webapps\ROOT</liferay.app.server.portal.dir>
```

Kód 30 Properties nastavené v settings.xml v Maven

Téměř každý vývojář při prvním kontaktu s portálem musí zvýšit paměť pro build, což je nastaveno pomocí systémové proměnné **MAVEN_OPTS** s adekvátním nastavením např. `-Xmx1500m -XX:MaxPermSize=500m`.

Schéma a rozvržením zmíněných modulů na serveru lze vidět na diagramu níže.



Obrázek 10 Schéma serverové části

14.3 Administrací nastavení

Nespornou výhodou portletů je jejich znovupoužitelnost, kdy lze stejný portlet umístit na více stránek nebo vícekrát na jednu stránku (více instancí). Tento prvek je využit v případě new-note-portlet, což umožňuje na každé stránce vytvářet novou poznámku, aniž by bylo nutné zásahu do kódu. Samozřejmostí je tvorba nových stránek zobrazených přes navigaci tohoto rozhraní. Lze je tak odebírat a přidávat za běhu aplikace.

Dále je využita zabudovaná politika hesel, jež se stará o validaci nastavené složitosti u nového hesla, expiraci, obnovu a mnoho dalších. Z hlediska bezpečnosti je novým uživatelům přiřazena vlastní role s přístupem ke konkrétním portletům, stránkám, s pouze čtenářskými právy bez možnosti úpravy portletů a jiných pokročilých zásahů. Běžnému uživateli se tak stránky jeví jako normální webová aplikace bez možnosti jakékoliv personalizace a správy.

Mezi další využití konfigurační položky patří parametry pro nastavení emailového serveru, výchozí adresa po přihlášení a odhlášení, změna loga nebo propojení se sociální sítí Facebook.

14.4 Použité technologie

Následující technologie hrály klíčovou roli při vývoji aplikace.

Tabulka 2 aplikované technologie

Serverová část	Klientská část	Vývojové a ostatní nástroje
Liferay 6.2	React (JSX)	Apache Tomcat
Spring framework	Flux	Apache Maven
Hibernate	jQuery	Node.js
Jackson	Gridster.js	Ruby
Apache POI	Fontawesome	JDK 7
JSP	DateTime picker	Gulp
Log4j	moment.js	NPM
Velocity	SASS	Bower
Portlet, Servlet API	Compass	Eclipse + Liferay IDE
Mail API	Fullcalendar	MySQL
Apache Commons	Twitter Bootstrap	

15 Výhody a nevýhody použití platformy Liferay

Nevýhody

1. Vysoké hardwarové nároky
2. Velká učící křivka
3. Chyby v aktuální verzi komplikující vývoj
4. Uzamčenost na použitých technologiích
5. Horší flexibilita vývoje ve specifických případech
6. Často nadbytečný kód, zatěžující klientskou část
7. Mnoho špatně zdokumentovaných úseků
8. Delší doba startování Liferay portálu

Výhody

1. Modularita rozsáhlých projektů
2. Oproti konkurenci lépe zdokumentovaný kód
3. Široká základna uživatelů a referenčních projektů
4. Dlouhá historie
5. Mnoho certifikovaných vývojových společností
6. Kompletní verze v open source variantě
7. Transparentní kód a nový vývoj
8. Patří k lídrům na trhu
9. Rozšíření pro kompletní modifikaci celého portálu
10. Mnoho zabudovaných nástrojů a aplikací
11. Spolupráce a integrace s moderními technologiemi a trendy na trhu
12. Několik vydaných literatur a ještě více blogů a odborných článků
13. Obsáhlé komunitní fórum s mnoha odborníky a vývojáři
14. Cílení na responzivitu s mobilními platformami
15. Neustálé inovace

16 Shrnutí výsledků

Tato práce představuje portály od jejich vnějších charakteristik po ty vnitřní. Vymezila pojmy Content Management System, portál, portlety a definovala jejich vlastnosti a funkci. Konkurence na trhu portálových řešení s každým rokem narůstá a je důležité neustále sledovat moderní trendy a pružně na ně reagovat. Velice důležitá je historie produktu, která je mnohdy rozhodující při volbě do korporátního prostředí. Liferay se řadí mezi jedničky na trhu a implementuje nejen portletové standardy, ale řadu dalších s širokou kompatibilitou napříč různými technologiemi.

Začínající programátor na této platformě musí pochopit mnoho principů a často si osvojit nové technologie, jež jsou integrované v Liferay portálu. Tato práce pomůže nahlédnout pod pokličku vývoje zásuvných modulů a demonstruje, co vše lze v portálu udělat či jak lze modifikovat. Klíčové kusy kódu pomohou k lepšímu pochopení vykládané problematiky stejně jako ilustrované či převzaté diagramy.

Aplikace pro upomínkování, organizování a zaznamenávání obsahuje celkem 8 portletů, z nichž každý zastává logicky ucelenou část stránky. Každý z nich obsahuje robustní klientskou základnu postavenou na JavaScriptové technologii React s architektonickým vzorem Flux. Tento přístup odemknul vysoce variabilní a moderní možnosti na klientské straně s dobře škálovatelným a výkonným kódem. Jedním z principů portletů bývá jejich portabilita mezi různými portály. V praxi je toto možné pouze po dodržení striktních pravidel limitujících v mnoha ohledech. Tento aspekt aplikace nezohledňuje, jelikož se v praxi téměř nesetkáváme s přenosem portletů do jiných implementací portálových řešení. Aplikace je po hlavní stránce funkční, avšak některé části budou ještě dále rozvíjeny, než bude možné uvedení do produkčního prostředí.

Počáteční analýza každé nové funkcionality by si měla klást otázky typu, jak moc bude daná část klíčová a jaké jsou nároky na její flexibilitu. Liferay usnadňuje často řešené problémy vlastní implementací, díky které není nutné řešit mnohá složitá zákoutí a je rovnou možné psát business logiku, která je předmětem změny. Pokud vyžadujeme specifické vlastnosti, často narážíme na problém kdy Liferay implementuje pouze základní a běžně používané případy a nám tak nezbyvá nic jiného než vymýšlet složitá řešení, aby daná věc fungovala. V těchto případech bývá výhodné použít již od začátku oficiální technologii s všeobecně známými postupy a zevrubnou dokumentací.

17 Závěry a doporučení

Cílem práce bylo představit technologii portálů a uvést čtenáře do základní problematiky. V první části byly popsány začátky webových aplikací pro správu obsahu. Ty se při svém vývoji transformovali až do moderních CMS řešení zajišťující dynamické publikování interaktivního obsahu s mnoha nástroji pro verzování, emailování, administraci nebo například kooperaci s ostatními uživateli. Práce ukázala propojení těchto systémů s portály a šla více do hloubky v jeho vnitřním fungování.

Dodavatelé portálů společně tvoří a následně implementují oficiální standardy, jež zajišťují portabilitu a jednotné řešení portletů. V případě platformy Liferay je nutné nahlédnout dále. Portlety jsou pouze základní rozšíření, která jsou prohloubena o mnohé zabudované funkcionality, jež se mohou časem stát standardem. Čtenáři jsou představeny rozšíření pro změnu vzhledu, rozložení portletů, neinvazivní úpravy zabudovaných služeb, ale také pokročilou úpravu jádra Liferay portálu.

Sílu Liferay portálu spatřuji v jeho široké základně uživatelů, která pouze implikuje kvalitu, otevřenost a možnosti jaké přináší. Mnoho klíčových aspektů je zde již vyřešeno, otestováno a vývojář má tak možnost přistoupit k tvorbě samotných aplikačních fragmentů bez nutnosti řešit správu uživatelů, bezpečnost, nastavení serveru či integraci s dalšími službami.

V představené aplikaci lze vidět využití mnohých zabudovaných technologií, které Liferay nabízí, ale také integraci s frameworky třetích stran. I přestože se na prvním pohled může zdát platforma nepřístupná vlastním modifikacím, aplikace demonstruje pravý opak a předvádí využití moderních klientských technologií, jež jsou v této době čím dál tím více žádané.

Zabudované vývojové prostředí Liferay IDE v nástroji Eclipse výrazně usnadnilo práci, jelikož zajišťuje rychlé nasazování provedených změn a interaktivní nabídky při tvorbě pluginů. Další hojně využívanou funkcí je propojení Liferay životních cyklů pro kompilaci zásuvných modulů přímo z kontextových nabídek projektu v Eclipse. Liferay je původně uzpůsoben pro sestavovací nástroj Ant. Díky tomu je většina oficiální dokumentace psaná ve vztahu k tomuto softwaru, což občas způsobuje zavádějící informace a ve finále mnoho zbytečných hodin konfigurace navíc.

Nemalá část této práce se věnuje klientské části aplikace, která se sestává především z JavaScriptové části a kaskádových stylů. Aplikace demonstruje vytvoření DOM fragmentů skrze komponenty psané JSX syntaxí běžící v JS frameworku React. Tento

přístup mohu silně doporučit, jelikož se osvědčil při psaní transparentních a modulárně založených aplikací. Komponenty lze tvořit velice snadno. HTML elementy jsou přímo psány v JavaScriptovém souboru což výrazně přispívá k lepší transparentnosti sledu akcí, snazší dynamice a interaktivitě našich komponent.

Vzor Flux jsem shledal velice vhodným v jakékoliv aplikaci pracující s modelem v klientské vrstvě. V portletové aplikaci má každý portlet svoji vlastní Flux architekturu, čímž jsou jejich prostory dokonale odděleny a jsou vůči sobě nezávislé. Mohou se rozvíjet do větších rozměrů, aniž by se z nich stal těžko udržitelný kolos.

Tato sada technologií výrazně usnadnila vývoj, a přestože na začátku bylo záměrem využít zabudovanou technologii Alloy UI, její nepoužití se ukázalo jako jedno z nejlepších rozhodnutí. Z mého pohledu se jedná o velice kontroverzní rozhodnutí použít tuto technologii v moderních projektech, a proto bych ji použil pouze tam, kde nejsou kladeny velké nároky na UX, při vývoji pro Marketplace nebo při prototypování aplikací. Na druhou stranu je již integrovaná do samotné instalace a je prověřená tisíci uživateli a vývojáři. Podobným rozhodnutím bude vývojář čelit při vývoji na této platformě dnes a denně. Leckdy však ani důkladná analýza nezabrání špatné volbě.

18 Seznam použité literatury

- ANON. *Portlet Tutorial: Portlet Lifecycle* [online]. [Cit. 20.2.2015]. Dostupné z WWW: <<http://jsr286tutorial.blogspot.cz/p/portlet-lifecycle.html>>.
- APACHE. *Apache Solr* [online]. Dostupné z WWW: <<http://lucene.apache.org/solr/>>.
- APOORVA, P. *Overriding Events: Working with PreActions and PostActions in EXT - Blog - Liferay.com* [online]. 2013. [Cit. 7.2.2015]. Dostupné z WWW: <<http://www.liferay.com/web/apoorvaprakash/blog/-/blogs/overriding-events-working-with-preactions-and-postactions-in-ext>>.
- AUGÉ, R. & MATHEW, R.G. *Logical Architecture - Wiki - Liferay.com* [online]. 2012. [Cit. 20.2.2015]. Dostupné z WWW: <<http://www.liferay.com/community/wiki/-/wiki/1071674/Logical+Architecture>>.
- BACKBASE. *Backbase Portal* [online]. 2015. Dostupné z WWW: <<http://www.backbase.com/portal-software/technology>>.
- BASHAM, B. *Head First Servlets and JSP* 2nd ed. Beijing ; Sebastopol, [Calif.] : O'Reilly, 2008. ISBN 9780596516680.
- BHATT, S. *Liferay Portal Performance Best Practices*. Birmingham, U.K. : Packt Pub., 2013. ISBN 1299674003 9781299674004 1782163689 9781782163688 9781782163695 1782163697.
- CAMARERO, J. *Application Adapters - Wiki - Liferay.com* [online]. 2011. [Cit. 20.2.2015]. Dostupné z WWW: <<https://www.liferay.com/community/wiki/-/wiki/Main/Application+Adapters>>.
- CHEDEAU, C. *Performance Calendar » React's Diff Algorithm* [online]. 2013. [Cit. 20.2.2015]. Dostupné z WWW: <<http://calendar.perfplanet.com/2013/diff/>>.
- CHEN, X. & YUAN, J.X. *Liferay 6*. Birmingham : Packt Publishing, 2013. ISBN 9781782162346 1782162348 9781782162353 1782162356.
- COMMUNITY. *Adding Portlet on Page Programmatically* [online]. Program Creek. Dostupné z WWW: <<http://www.programcreek.com/java-api-examples/index.php?api=com.liferay.portal.model.LayoutTypePortlet>>.
- DALQUIST, E. *Portlet Request Lifecycle* [online]. 2013. Dostupné z WWW: <<https://wiki.jasig.org/display/PLT/Portlet+Request+Lifecycle>>.
- EPISERVER. *Web Content Management* [online]. 2015. Dostupné z WWW: <<http://www.episerver.com/web-content-management/>>.
- EXO. *eXo Community* [online]. 2015, eXo Community. [Cit. 20.2.2015]. Dostupné z WWW: <<http://community.exoplatform.com>>.
- FACEBOOK. *Facebook/flux* [online]. 2015, GitHub. [Cit. 20.2.2015]. Dostupné z WWW: <<https://github.com/facebook/flux>>.

- FACEBOOK. *Flux / Application Architecture for Building User Interfaces* [online]. 2014. [Cit. 20.2.2015]. Dostupné z WWW: <<http://facebook.github.io/flux/index.html>>.
- FERRER, J. *Liferay's Architecture: The Beginning of a Blog Series - Blog - Liferay.com* [online]. 2012. [Cit. 20.2.2015]. Dostupné z WWW: <<http://www.liferay.com/web/jorge.ferrer/blog/-/blogs/liferay-s-architecture-the-beginning-of-a-blog-series>>.
- HEPPER, S. & KÖTH, O. *What's New in the Java Portlet Specification V2.0 (JSR 286)?* [online]. 2008. [Cit. 20.2.2015]. Dostupné z WWW: <http://www.ibm.com/developerworks/websphere/library/techarticles/0803_hepper/0803_hepper.html>.
- HIMANSHU, J. *Liferay: Service Builder Concept* [online]. 2014. [Cit. 7.2.2015]. Dostupné z WWW: <<http://innovationliferay.blogspot.cz/p/service-builder-concept.html>>.
- HINZ, P. *Portals vs. Web CMS - What's the Difference?* [online]. CMSWire.com. [Cit. 20.2.2015]. Dostupné z WWW: <<http://www.cmswire.com/cms/web-cms/portals-vs-web-cms-whats-the-difference-013713.php>>.
- IBM. *IBM Digital Experience Analytics* [online]. 2014. [Cit. 20.2.2015]. Dostupné z WWW: <<http://www-01.ibm.com/software/collaboration/digitalexperience/analytics.html>>.
- JAVA COMMUNITY PROCESS. *JSR 168: Portlet Specification* [online]. 2003. Dostupné z WWW: <<https://jcp.org/ja/jsr/detail?id=168>>.
- JAVA COMMUNITY PROCESS. *JSR 286: Portlet Specification 2.0* [online]. 2008. Dostupné z WWW: <<https://jcp.org/ja/jsr/detail?id=286>>.
- KHARKOVSKI, R. *Software Costs* [online]. 2013, WhyWebSphere.com Blog. [Cit. 20.2.2015]. Dostupné z WWW: <<http://whywebsphere.com/2013/09/26/software-costs/>>.
- KOUBA, Š. *Zaklínadlo Jménem WebDAV / Interval.cz* [online]. 2003. [Cit. 20.2.2015]. Dostupné z WWW: <<http://interval.cz/clanky/zaklinadlo-jmenem-webdav/>>.
- LECOMTE, J. *Important Announcement Regarding YUI* [online]. 2014, Yahoo Engineering. [Cit. 20.2.2015]. Dostupné z WWW: <<http://yahooeng.tumblr.com/post/96098168666/important-announcement-regarding-yui>>.
- LIFERAY. *Liferay Portal 6.2 Developer's Guide - Development - Liferay.com* [online]. [Cit. 20.2.2015]. Dostupné z WWW: <<https://www.liferay.com/documentation/liferay-portal/6.2/development>>.
- LIFERAY. *Liferay Technical Specifications & Details - Liferay.com* [online]. 2014. [Cit. 20.2.2015]. Dostupné z WWW: <<https://www.liferay.com/products/liferay-portal/tech-specs>>.
- MCKAY, C. & FERRER, J. *Liferay Developer's Guide* [online]. Liferay, Inc., 2011.

- MCKENZIE, C. *JSR-286 Development Tutorial: An Introduction to Portlet Programming* [online]. 2013a, The Server Side. Dostupné z WWW: <<http://www.theserverside.com/tutorial/JSR-286-development-tutorial-An-introduction-to-portlet-programming>>.
- MCKENZIE, C. *JSR-286 Development Tutorial: Understanding the PortletSession* [online]. 2013b, The Server Side. Dostupné z WWW: <<http://www.theserverside.com/tutorial/JSR-286-development-tutorial-Understanding-the-PortletSession>>.
- MEERA, P. *Portlet Introduction / Portlet Technology Introduction - Blog - Liferay.com* [online]. 2014. [Cit. 20.2.2015]. Dostupné z WWW: <<http://www.liferay.com/web/meera.succes/blog/-/blogs/portlet-introduction-portlet-technology-introduction>>.
- MONTERO, A. & POTTER, J. *Scripting Language Portlets - Wiki - Liferay.com* [online]. 2010. [Cit. 20.2.2015]. Dostupné z WWW: <<https://www.liferay.com/community/wiki/-/wiki/Main/Scripting+languages+to+develop+portlets++>>.
- MURPHY, J., VALDES, R., PHIFER, G., TAY, G. & REVANG, M. *Magic Quadrant for Horizontal Portals* [online]. 2014. Dostupné z WWW: <<http://www.gartner.com>>.
- NILANG. *Spring MVC Portlet in Liferay* [online]. 2012, Tech Blog. [Cit. 20.2.2015]. Dostupné z WWW: <<http://www.opensource-techblog.com/2012/09/spring-mvc-portlet-in-liferay.html>>.
- ORACLE. *JD Edwards EnterpriseOne Portal Content Configuration Guide* [online]. [Cit. 20.2.2015]. Dostupné z WWW: <https://docs.oracle.com/cd/E24705_01/doc.91/e21055/intro_to_portlets.htm#EOPCG670>.
- PANDA, S. *Understanding Angular's \$apply() and \$digest()* [online]. 2014, SitePoint. [Cit. 20.2.2015]. Dostupné z WWW: <<http://www.sitepoint.com/understanding-angulars-apply-digest/>>.
- PAOLUCCI, N. *Flux Step By Step* [online]. 2014, Atlassian Blogs. [Cit. 20.2.2015]. Dostupné z WWW: <<http://blogs.atlassian.com/2014/08/flux-architecture-step-by-step/>>.
- PATEL, N. *Creating Service Layer in Service Builder in Liferay* [online]. 2013. [Cit. 7.2.2015]. Dostupné z WWW: <<http://www.opensource-techblog.com/2013/03/creating-service-layer-in-service.html>>.
- RAVINDRANATH, A. & FERRER, J. *Service Builder and Liferay Database Table Creation - Wiki - Liferay.com* [online]. 2011. [Cit. 7.2.2015]. Dostupné z WWW: <<http://www.liferay.com/community/wiki/-/wiki/1071674/Service+Builder+and+Liferay+Database+Table+Creation/maximized>>.
- ROCHA, Z. *AlloyUI 3 Released: Bye Bye YUI - Blog - Liferay.com* [online]. 2014. [Cit. 6.2.2015]. Dostupné z WWW: <<http://www.liferay.com/web/zeno.rocha/blog/-/blogs/alloyui-3-released-bye-bye-yui>>.

- SARIN, A. *Portlets in Action*. Shelter Island, NY : Manning, 2012. ISBN 9781935182542.
- SAYS, S. *Liferay Site Specific Customization - Application Adapter* [online]. 2014, Liferay Portal. [Cit. 20.2.2015]. Dostupné z WWW: <<https://sushilsaini.wordpress.com/2014/04/08/liferay-site-specific-customization-application-adapter/>>.
- SEZOV, R. *Liferay in Action: The Official Guide to Liferay Portal Development*. Shelter Island, NY : Manning, 2011. ISBN 9781935182825.
- SHIPLEY, R. *Content Management Software Review 2015 / Best CMS Systems* [online]. 2015, TopTenREVIEWS. [Cit. 20.2.2015]. Dostupné z WWW: <<http://cms-software-review.toptenreviews.com/>>.
- SPRING. *Spring Portlet MVC Framework* [online]. 2011. Dostupné z WWW: <<http://docs.spring.io/spring-framework/docs/3.0.7.RELEASE/reference/portlet.html>>.
- STEIGERWALD, D. *Proč Facebook React Zabil jQuery* [online]. 2014, Zdroják. [Cit. 20.2.2015]. Dostupné z WWW: <<http://www.zdrojak.cz/clanky/proc-facebook-react-zabil-jquery/>>.
- STEIGERWALD, D. *What's Wrong with Angular.js — Este.js Framework* [online]. Medium. [Cit. 22.2.2015]. Dostupné z WWW: <<https://medium.com/este-js-framework/whats-wrong-with-angular-js-97b0a787f903>>.
- SUBBU, A. *Java Transaction Service* [online]. 1999. Dostupné z WWW: <<https://www.subbu.org/articles/jts/JTS.html>>.
- SWAIM, B. & KHANCHANDANI, P. *Access Objects from Velocity - Wiki - Liferay.com* [online]. 2013. [Cit. 22.2.2015]. Dostupné z WWW: <<http://www.liferay.com/community/wiki/-/wiki/Main/Access+Objects+from+Velocity>>.
- TELCIK, T. *How Do I Undeploy a Liferay Portal 6 EXT Plugin* [online]. 2013. Dostupné z WWW: <<https://www.permeance.com.au/web/tim.telcik/home/-/blogs/how-do-i-undeploy-a-liferay-portal-6-ext-plugin>>.
- WHEELER, K. *What's Your Favorite MV* JavaScript Framework?* [online]. 2014. [Cit. 20.2.2015]. Dostupné z WWW: <<https://www.wedgies.com/question/54e42446c9fbad0f000003f8>>.
- WILLIAMS, S. *What Is a Portal?* [online]. Enterprise Content Management. Dostupné z WWW: <<http://www.contentmanager.eu.com/portal.htm>>.
- WILLIAMS, S. *What Is Web Content Management (WCMS)* [online]. Enterprise Content Management. Dostupné z WWW: <<http://www.contentmanager.eu.com/wcms.htm>>.
- YUAN, J.X. *Liferay Portal 6 Enterprise Intranets Build and Maintain Impressive Corporate Intranets with Liferay*. Birmingham, U.K. : Packt Pub., 2010. ISBN 9781849510394 1849510393.

YUAN, J.X., SAHAT, C. & ROGERS, S. *Liferay Portal 5.2 Systems Development Build Java-Based Custom Intranet Systems on Top of Liferay Portal*. Birmingham, U.K. : Packt Pub., 2009. ISBN 9781847194718 1847194710.

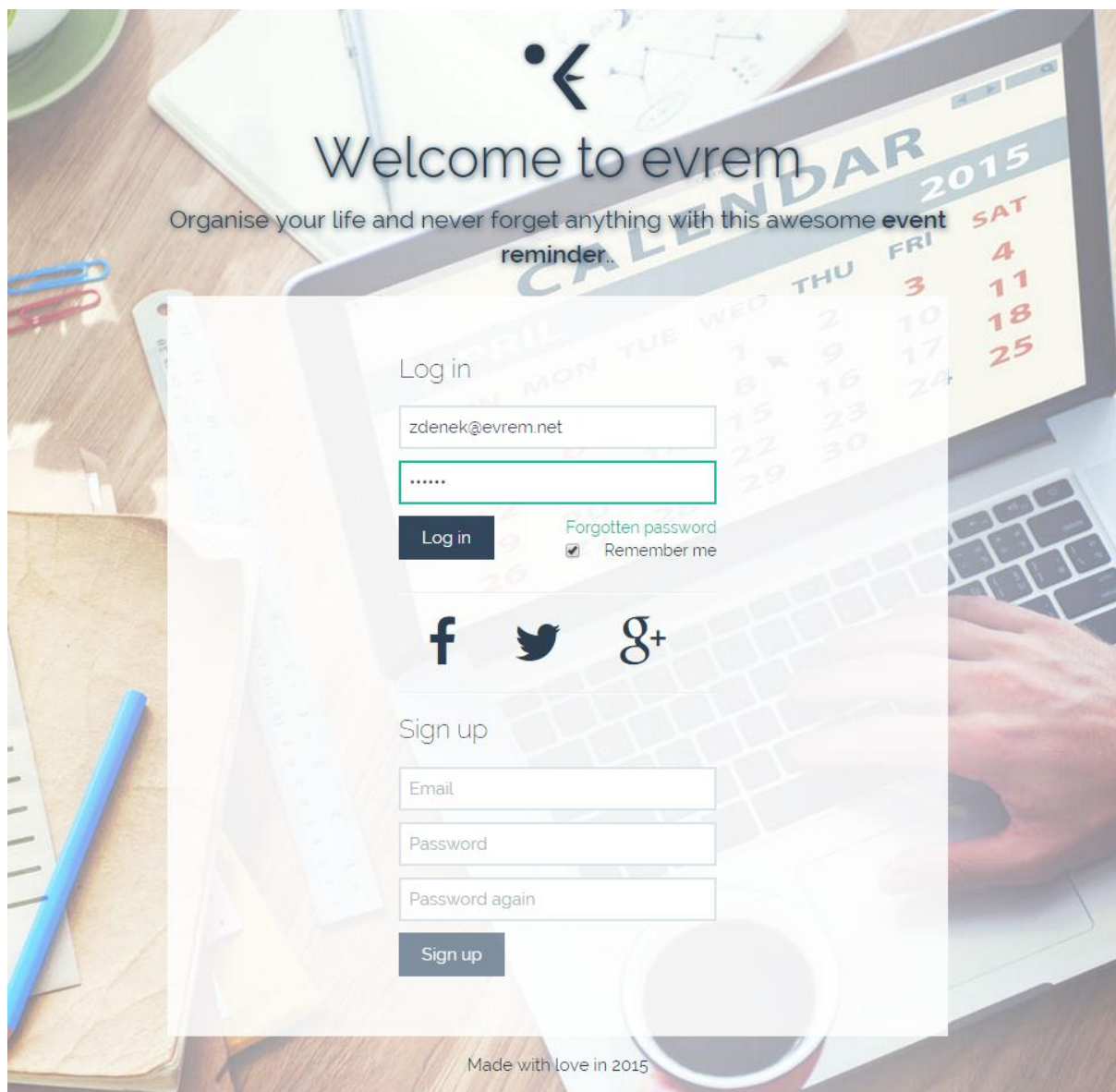
ZUBAIR, G. & GONZALEZ, C. *Portlet to Portlet Communication - Wiki - Liferay.com* [online]. 2014. [Cit. 20.2.2015]. Dostupné z WWW: <<https://www.liferay.com/community/wiki/-/wiki/Main/Portlet+to+Portlet+Communication>>.

19 Přílohy

Na přiloženém DVD jsou obsaženy zdrojové kódy aplikace společně se zkompilovanými archivy webové aplikace.

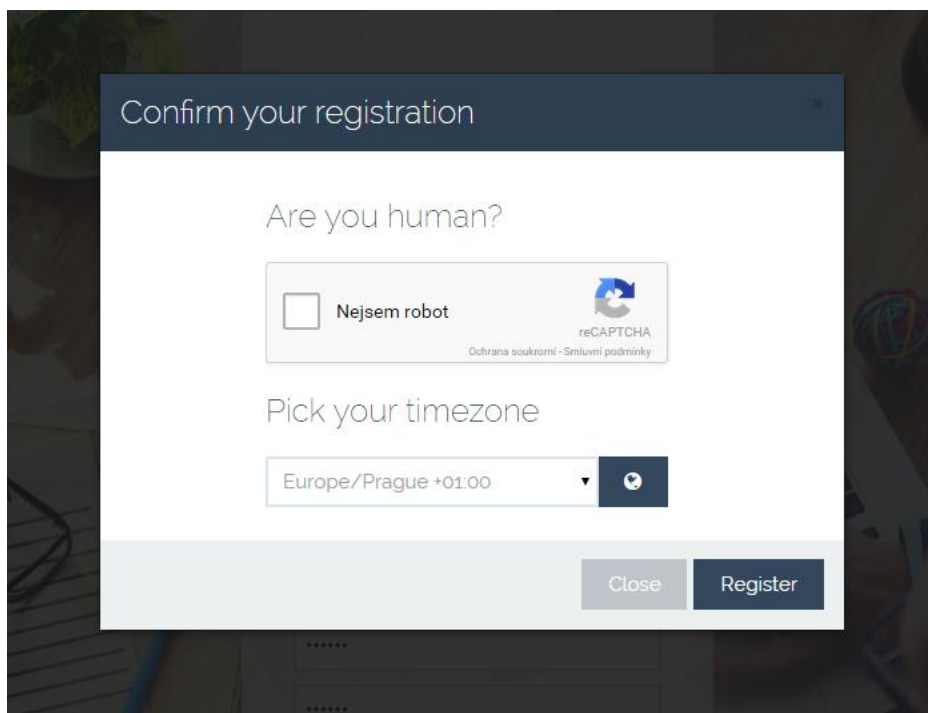
Následující obrázky zobrazují vytvořenou aplikaci pro organizování, běžící na platformě Liferay.

Přihlašovací obrazovka s rychlým registračním formulářem.



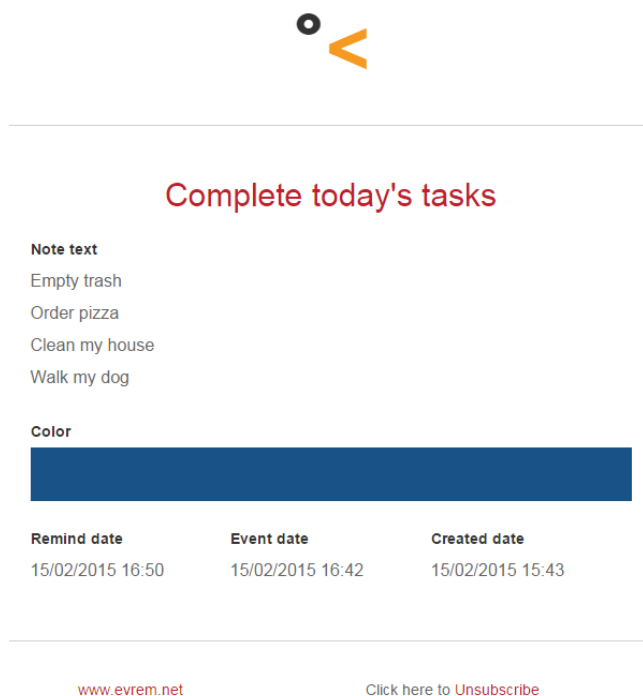
Obrázek 11 Ukázka aplikace - úvodní obrazovka

Výběr časového pásma pro časované zaslání upomínek a nová reCaptcha od Google.



Obrázek 12 Ukázka aplikace - potvrzení registrace

Emailové notifikace postavené na Velocity frameworku s responzivním zobrazením pro mobilní zařízení.



Remind date	Event date	Created date
15/02/2015 16:50	15/02/2015 16:42	15/02/2015 15:43

Obrázek 13 Ukázka aplikace - emailové notifikace

Interaktivní zed' poznámek s portletem pro jejich přidání a s navigačním panelem.

Obrázek 14 Ukázka aplikace - interaktivní zed' poznámek

Zobrazení časovaných událostí v členěném přehledu. V dolní části se nachází znovu použitý portlet pro přidání poznámek s aktivními validacemi.

Obrázek 15 Ukázka aplikace – úvodní přehled událostí

Kalendářová komponenta zobrazující události v měsíčním, týdenním nebo denním přehledu.

<

>

today

February 2015

month

week

day

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
16:21 Do my homework	20:43 Buy some vegetables			18:02 Dentist prevention	20:46 Go to school to...	12:29 Empty trash Order pizza Clean my house Walk my dog
15	16	17	18	19	20	21
20:44 Wash my clothes			20:45 Uncle's birthday			
22	23	24	25	26	27	28
			20:44 Mow the lawn		20:45 Prepare on presentation	

Obrázek 16 Ukázka aplikace - kalendář událostí

Filtrace poznámek zobrazující všechny poznámky v kompletním přehledu s možností exportu do Excelu nebo permanentního odstranění smazaných poznámek.

Color	Note text	Event time	Remind time	Repeat period	Done	Wall	Deleted	Creation date	Last modified
	Do my homework	08/02/2015 16:21						14/02/2015 19:39	14/02/2015 19:48
	Dentist prev	12/02/2015 18:02	11/02/2015 18:03	Half a year	✕	📅		14/02/2015 19:19	14/02/2015 19:19
	Mobile warr	14/09/2015 20:27				📅		14/02/2015 19:27	14/02/2015 19:27
	Used JAVA	19/02/2015 18:04				📅		14/02/2015 19:26	14/02/2015 19:26
	Gmail accou					📅		14/02/2015 19:21	14/02/2015 19:21
	Finish diplor	30/04/2015 12:21				📅		14/02/2015 19:28	14/02/2015 19:54
	Empty trash	14/02/2015 12:29				📅		14/02/2015 19:30	14/02/2015 19:30
	Mother's bir	13/06/2015 00:00	13/06/2015 08:00	Yearly	✕	📅		14/02/2015 19:38	
	Buy some v	09/02/2015 20:43						14/02/2015 19:43	14/02/2015 19:47
	Mow the lav	25/02/2015 20:44						14/02/2015 19:44	
More events...									

Gmail account

Login: john234

Password: walker1

Export

Remove permanently

New blank note

📁

✓

🕒

🔔

🔄

📝

📅

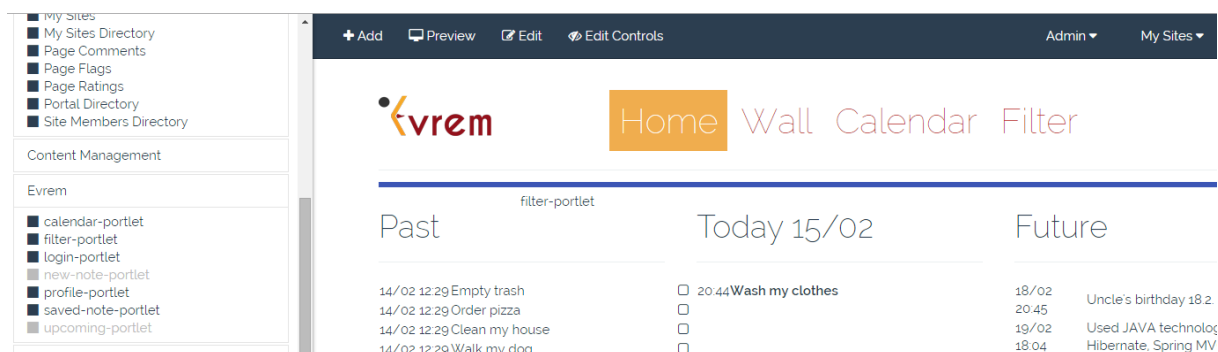
📱

✕

🔄

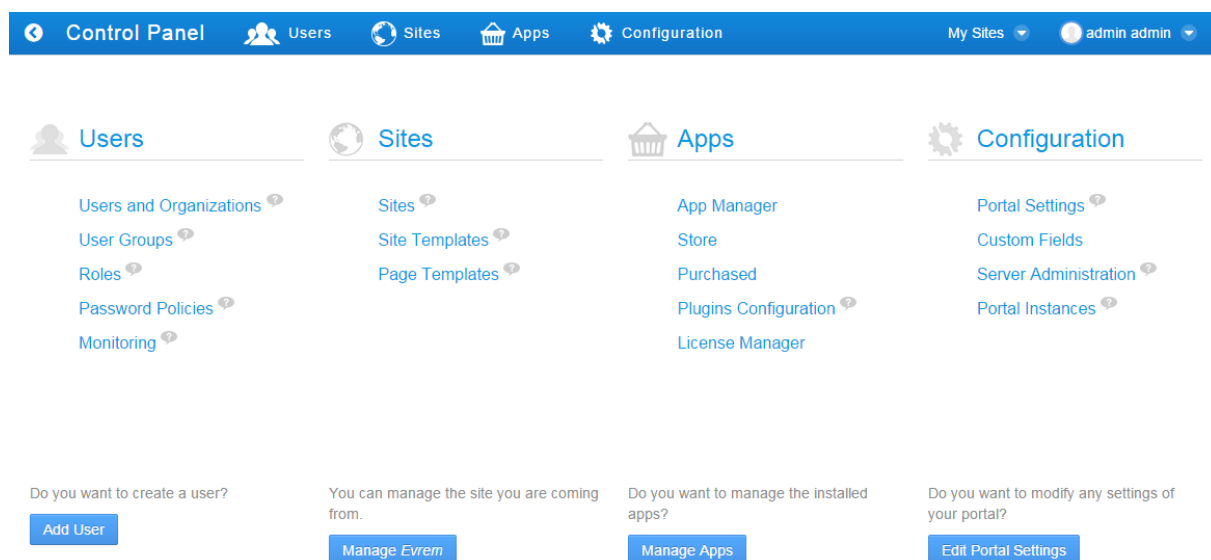
Obrázek 17 Ukázka aplikace - filtrace a zobrazení všech poznámek

Přidání portletu na stránku přes Drag-and-Drop mechanismus. Dále zde vidíme lištu Dockbar nabízející mnohé funkcionality pro uspořádání obsahu.



Obrázek 18 Ukázka aplikace - administrační panel a přidání portletu

Ovládací panel pro kompletní správu portálu.



Obrázek 19 Administrační rozhraní Liferay 6.2.

20 Zadání závěrečné práce

22.9.2014

Tisk zadání závěrečných prací



UNIVERZITA HRADEC KRÁLOVÉ

Fakulta informatiky a managementu

Rokitanského 62, 500 03 Hradec Králové, tel: 493 331 111, fax: 493 332 235

Zadání k závěrečné práci

Jméno a příjmení studenta:

Zdeněk Věcek

Obor studia:

Informační management (5)

Jméno a příjmení vedoucího práce:

Filip Malý

Název práce:

Vývoj portálových aplikací

Název práce v AJ:

Portal applications development

Podtitul práce:

Podtitul práce v AJ:

Cíl práce: Prozkoumat možnosti portálových aplikací a CMS systémů obecně. Shrnout základní principy a výhody užití Liferay portálu. Demonstrovat vývoj pluginů v dané technologii.

Osnova práce:

Úvod

CMS systémy

Portálové aplikace – portlety

Uživatelské rozhraní Liferay

Vývoj pluginů v prostředí Liferay

Závěr

Použitá literatura

Projednáno dne: **22. 9. 2014**

Podpis studenta

Podpis vedoucího práce