|  |  |  |
| --- | --- | --- |
| **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. | | |
| Hradec Králové | duben 2015 | |
| Prohlášení:  Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury. | | | |
| V Hradci Králové dne 4.1.2015 | | 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

Annotation

Title: Portal application development

The

Obsah

[1 Úvod 8](#_Toc400805906)

Seznam obrázků

**Nenalezena položka seznamu obrázků.**

Seznam tabulek

**Nenalezena položka seznamu obrázků.**

# Úvod

Portálové aplikace…

# Výběr JS frameworku

Liferay v celé své klientské vrstvě využívá frameworku YUI verze 3 od Yahoo, který tvoří základnu pro Alloy UI. Koncem srpna 2014 Yahoo oznámilo ukončení vývoje frameworku YUI. Důvodem bylo odchýlení od současných trendů, které jsou tvořeny single page aplikacemi, isymorfními aplikacemi tvořené pomocí Node.js, ale i nástrojema pro správu závislostí jakou jsou NPM či Bower, díky kterým lze celá klientská aplikace sestavit za pomocí taskovacích nástrojů jako Grunt či Gulp a dalšími nezbytnými nástroji, které jsou nezbytné pro transpilaci kódu, resolvování závislostí, oddělování scope 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 existovat, aby splňovala konkurenceschopnost a byla kompatibilní se zmíněnými nástroji.

Velice populárním JS frameworkem v komunitě LFR je zcela bezkonkurenčně AngularJS. Je velice jednoduchý pro první inicializaci a téměř hned dokážeme vytvořit model komunikující s template/view vrstvou, jelikož zde funguje mechanismus two-way data binding. Tento mechanismus zajišťuje propagování změn z view do modelu na straně clienta, tak z modelu do view. Na počátku vývoje šetří tento přístup mnoho času a technických prostředků. Velice jednoduše lze přidávat nové controllery a hierarchicky je zanořovat a následně z nich dědit. Každý controller má svůj vlastní scope, který nám oddělení jednotlivé jméné prostory a čímž se nám uchová čistota v globálním scope a není tedy nutné využití dalších nástrojů jako je CommonsJS pro vymezení oddělených scope. Angular však také nabízí velice účinnou, ale zároveň nebezpečnou zbraň, která se prolíná jak JS vrstvou tak template vrstvou. Jedná se o direktivu scope.watch(angularExpression, callback). Tato funkce zajišťuje kontrolu zadaného expression kde při každé změně je zavolán callback se starou a novou hodnotou ve vstupních parametrech. Výhody jsou zcela nesporné a jde zejmená o rychlou reakci na konkrétní změny nějakou funkcí. Problém nastává v okamžiku, kdy se ve scopu nachází desítky watcherů, kdy v callbacku jsou měněny hodnoty, na kterých jsou opět navěšeny watchery. Situace začne být velice nepřehledná a debug chyb v takovémto systému může zabrat hodiny i dny. Two-way databinding nese další obrovskou daň v podobě dirty checking mechanismu, který kontroluje změny v celé hierarchii scopů a reaguje na ně příslušnou aktualizací. Celý cyklus se volá vždy minimálně dvakrát pro kontrolu, že byly všechny změny zpropagovány a že jiný event mezitím nezměnil hodnotu již ověřené proměnné. Pro případ, že by se někde vytvořila smyčka v našich watcherech je nastaven limit opakování nad danou proměnou na 10 iterací. Výsledkem je velká výkonová náročnost, která způsobuje nemalé problémy i moderně vybaveným PC předveším však u větších aplikací. Dalším problémem je velká učící křivka při tvorbě direktiv a dalších klíčových prvků nezbytných při vývoji.

## ReactJS

Na problémy zmíněné v předchozím odstavci výborně odpovídá ReactJS s architektonickým vzorem Flux. Jedná se o relativně novou technologie pocházející z dílny Facebook, na němž také běží a rychle sklouzává do obliby a povědomí dalších velkých firem. Díky jeho architektuře poskytující vysoký výkon, který je demonstrován například na staré hře Wolfenstein 3D, který je jako html stránka renderovaná přes ReactJS, generována jádrem Meteor. Další praktickou ukázkou může být desktopový textový editor Atom, který běží v prohlížeči a poskytuje stejné funkce jako populární editor Sublime text. Nesporná výhoda je jeho jednoduchost a malá učící křívka. Ačkoli se jedná o koncept poskytující možnosti složitých js frameworků, jeho implementace je přímočará a přehledná. Tvorba komponent se tak stává základním stavebním kamenem což přispívá k vysoké modularitě a znovupoužitelnosti. Pro naše potřeby využíváme nadstavbovou syntaxi JSX, umožňující psaní html tagů přímo v javascriptu, což opět usnadňuje srozumitelnost a čitelnost kódu. 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 je generován dynamicky zjistíme, že se z naší JSP template stává více javascript než značkovací jazyk poskytující HTML kostru aplikace. Tento problém nám opět řeší ReactJS.

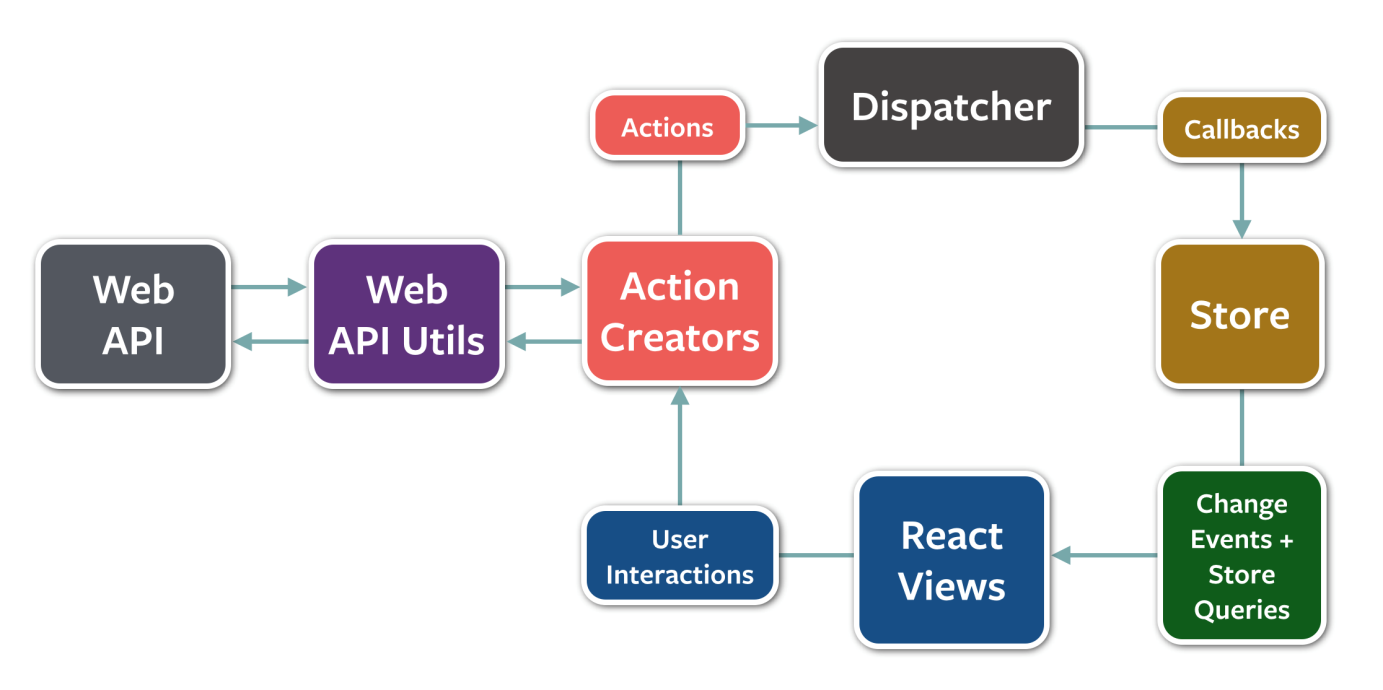
Klíč k tomuto cíli tkví ve využití virtuálního DOM, který tvoří jakousi nádstavbu nad skutečným DOMem, což nám mimo jiné překlenuje rozdíly napříč různými prohlížeči a jejich verzemi. Implementace virtuálního DOM odlaďuje některé chyby v event systému, které jsou známy ve starších prohlížečích. Hlavní výhoda je však ukryta v propagaci změn a v aktualizaci pouze nezbytné části, která se skutečně změnila, jelikož jakákoliv úprava skutečného DOMu je velice náročná záležitost a tak je nutné, aby zásahy byly co nejmenší a nejméně časté. Virtualní DOM jsou pohé javascriptové objekty, které si drží stav a referenci vůči skutečnému DOMu. V každé render fázi je heuristicky spočítáno, jaké nejmenší kroky musí být provedeny, aby se zpropagoval aktuální stav do DOMu, jsou vyhodnoceny všechny uzly a komponenty potomků v hierarchii a jsou překresleny potřebné části DOMu. Toto překreslení je však provedeno pouze jednou, díky virtuálnímu DOMu, kde jsou provedeny všechny nezbytné kroky pro zjištění finálního stavu.

## 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. Stojí na principu Unidirectional flow, které těží z jednotného postupu akcí. Všechny akce vznikají na View, které volá vrstvu Actions, což představuje pomocnou vrstvu pro vydefinování akce, jejich parametrů, typu a označení konstantou v kontextu aplikačního dispatcheru. Jeho funkcí je delegování všech akcí na příslušné callbacky zaregistrované ve Store vrstvě. Store je jakési úložiště dat a zároveň vrstva operující nad těmito daty či komunikující se serverem, což nám zajišťuje konzistenci a jednotný obraz dat v celé aplikaci. Po vykonání callbacku je emitnuta změna a komponenty, které poslouchají na danou událost jsou překresleny s aktuálními daty.

Tento princip nám elegantně zpřehleďnuje všechny události, které se v aplikaci dějí a je velice snadné v takovéto struktuře odlaďovat chyby. Dále se tímto vyvarujeme nepředvídatelnému sledů akcí a jejich provázanosti, které nám v případě Angularu způsobovaly nemalé starosti.

Tento vzor jsem shledal velice vhodnými v portletové aplikaci, kdy každý portlet má svoji vlastní Flux architekturu čímž jsou portlety dokonale odděleny a jsou vůči sobě nezávislé.

****

# Architektura aplikace Evrem

## Frontend

V dnešní době již prakticky nenalezneme moderní aplikaci bez javascriptu. Není proto divu, že velká část aplikace je napsána právě v něm. JSP tak využíváme pouze při render fázi, na inicializaci počátečních dat do příslušných Stores daných portletů, kam patří i URL generované přes taglib <portlet:resourceURL id=*"saveCoordinates"* /> .

Toto je 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 javascriptový soubor obsahuje veškerou logiku a template daného portletu napsané v ReactJS. Toto odstínění od Liferay způsobu psaní nám umožňuje vysokou variabilitu vzhledu, interaktivity, ale i propojení s moderními frontendovými knihovnami. Aplikace vykazuje vysokou interaktivitu proto je na místě použít technologii, která nepoužívá oddělenou šablonu od javascriptu, 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'));  }  }  ); |

Jak již bylo zmíněno u architektury Flux, všechny akce prochází od View, Actions přes Dispatcher až do Store, kde v případě serverové akce je zaslán AJAX request, který je odchycen Spring web MVC controllerem, vracejícím response. Cílová URL adresa je použita z počáteční inicializace, tedy adresa obsahující všechny náležitosti potřebné k identifikaci portletu, metody, informaci o cachování, layoutu či módu portletu (VIEW, EDIT, HELP).

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

Portlety sdílejí společné js knihovny a CSS styly prostřednictvím Theme. Základní proměnné jsou přístupné z globálního window scope, čímž je snížena velikost výsledných balíčků pro každý portlet a tak nejsou externí závislosti v průběhu kompilace přibaleny.

Ačkoliv Liferay umožňuje použití SASS a Compass vestavěném přímo v Liferay IDE, využívám vlastní strukturu a kompilátor, který mi umožňuje použití nejnovějších funkcí v CSS. Dále jsem schopen vyčlenit veškerý vlastní kód do umístění, kde se nachází veškeré komponenty předkompilované Gulpem. Do těchto složek jsou umístěny pouze soubory se stejným názvem jako je ve vygenerované struktuře Theme, jelikož takový styl nahradí původní styl v rodičovském souboru. Výsledkem se stane soubor původní soubor s modifikacemi provedenými ve zdrojovém css/scss.

Do theme jsou balíky kopírovány po provedení kompilace, minifikace a následném sloučení do jednoho souboru. Tyto úkony jsou zajišťovány pomocí taskovacího managera Gulp běžící na Node.js.

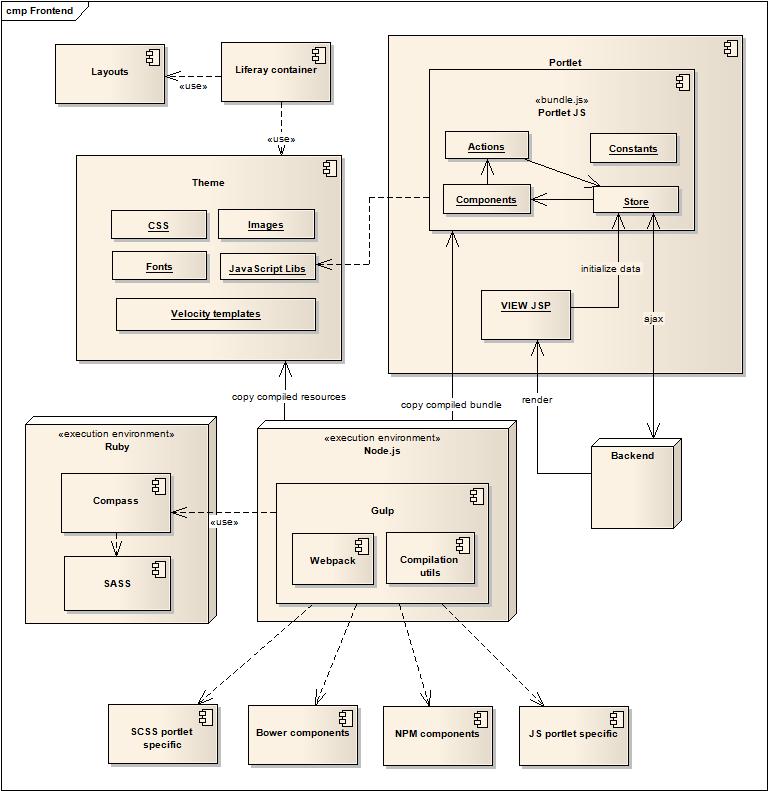
Společně s webpackem, jehož velkou výhodou je integrovaný CommonsJS jsou obstarávány balíky pro JS část. Webpack lze nastavit pro práci s různými typy souborů, čemuž se říká loadery. Pomocí expression řekneme, že daný soubor má být předkompilován příslušným loaderem a teprve poté jsou zpracovány závislosti, 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. V ruku v ruce se SASS jde Compass což je funkční knihovna, pro psaní cross browser kompatibilních stylů. Nemusíme se tak starat o psaní moderních stylů pro všechna prohlížečová jádra, ale jednoduše použijeme Compass funkci nebo mixin, který po kompilaci vygeneruje všechny styly pro dostupné prohlížeče. Pro používání tohoto nástroje je nutné mít nainstalované běhové prostředí Ruby s gemy Compass a Sass. Ukázkový kód lze vidět níže, znázorňující dědičnost, použití funkcí, vlastní mixinů či proměnných.

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

Gulp se nám v neposlední řadě stará o správu závislostí z Bower a NPM. Tyto populární package managers nám poskytují standardní způsob balíčkování knihoven, což je výhodné pro jednotné použití. Bower se liší od NPM hierarchí 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.

V Theme využíváme šablonovací systém Apache Velocity. Zde využíváme výchozí strukturu s co možná nejmenšími zásahy. Šablony tvoří kompletní strukturu webové stránky od hlavního html tagu až po obsahy jednotlivých portletů. Nachází se zde mnoho předdefinovaných fragmentů, které lze změnit pouze za pomocí pluginu hook. V naší aplikaci nebyl takový zásah potřeba. Některým úpravám jsem se však nevyhnul. Šlo o úpravu navigační panelu, o ikonu profilu s vlastní dropdown funkcí, přidání dalších knihoven do hlavičky (nová Google reCaptcha) nebo skrytí některých funkcí nepřihlášenému uživateli. Zásahy v této části jsou velice citlivé na správné vykreslování portletu, ale i se zvýšením verze Liferay je více než jistá kompletní revize všech templates.



## ReactJS

# Zdroje

Flux architecture - https://scotch.io/tutorials/getting-to-know-flux-the-react-js-architecture

Yahoo anouncement - http://yahooeng.tumblr.com/post/96098168666/important-announcement-regarding-yui

AngularJS dirty checking - http://www.sitepoint.com/understanding-angulars-apply-digest/

ReactJS Diff mechanism - http://calendar.perfplanet.com/2013/diff/

