# C. Implementace, použité technologie

## Maven

Apache Maven je nástroj pro správu, řízení a automatizaci buildů aplikací. Ačkoliv je možné použít tento nástroj pro projekty psané v různých programovacích jazycích, podporován je převážně jazyk [Java](http://cs.wikipedia.org/wiki/Java_(programovac%C3%AD_jazyk)). Název [maven](http://cs.wikipedia.org/w/index.php?title=Maven&action=edit&redlink=1) pochází z [jidiš](http://cs.wikipedia.org/wiki/Jidi%C5%A1) a znamená „znalec“. Maven byl vytvořen jako nástroj pro zjednodušení buildů pro projekt [Jakarta Turbine](http://cs.wikipedia.org/w/index.php?title=Jakarta_Turbine&action=edit&redlink=1). Hlavním impulzem pro vznik byla snaha o standardizaci a znovupoužitelnost buildovacích [skriptů](http://cs.wikipedia.org/wiki/Skript_(program)), která v tehdy používaném nástroji [Apache Ant](http://cs.wikipedia.org/wiki/Apache_Ant) nebyla plně podporována. [[1]](http://cs.wikipedia.org/wiki/Apache_Maven" \l "cite_note-maven.org-2)

## Reflexe

Reflexe je schopnost [programovacího jazyka](http://cs.wikipedia.org/wiki/Programovac%C3%AD_jazyk) zjistit za běhu informace o určitém [objektu](http://cs.wikipedia.org/wiki/Objektov%C4%9B_orientovan%C3%A9_programov%C3%A1n%C3%AD). Obecně, poněvač není jen [OOP](http://cs.wikipedia.org/wiki/OOP), je to schopnost zjistit informace o programu a jeho syntaktické struktuře. V objektově orientovaném programování je program rozdělen do tříd, kdy jednotlivá třída popisuje vnitřní strukturu objektu a jeho vnější rozhraní. Na základně tříd je možné tvořit jednotlivé objekty. Některé jazyky mají schopnost za běhu zjistit informace o daném programu. Tato schopnost se nazývá reflexe, s jejíž pomocí lze získat za běhu programu informace o typu objektu. V objektově orientovaném programování se dá říci, že vše je objekt, tak je tedy objektem i třída a jiné datové typy, o kterých lze zjistit požadované informace. [2]

Výhodou reflexe například je, že programátor nemusí dopředu vědět, jaké parametry / metody daná třída / objekt má, ale dovede si je získat za chodu programu a podle potřeby spustit, resp. Jinak využít. Dává mu tím do rukou značně silný nástroj pro zobecnění kódu, silou porovnatelný ukazatelům v jazyku C. (kupř. reflexi využívá mnoho moderních programátorských nástrojů, jako je třeba Spring)

## Spring

Jaká je filozofie Springu? Nejdůležitějším slovem v souvislosti se Springem je: kontext. Jádro Springu je postaveno na využití návrhového vzoru [*Inversion of Control*](http://cs.wikipedia.org/w/index.php?title=Inversion_of_Control&action=edit&redlink=1) a je označován jako IoC kontejner. Tento návrhový vzor funguje na principu přesunutí zodpovědnosti za vytvoření a provázání objektů z aplikace na framework. Objekty lze získat prostřednictvím [*vkladání závislostí*](http://cs.wikipedia.org/w/index.php?title=Dependency_Injection&action=edit&redlink=1) *(dependency injection)*, což je speciální případ Inversion of Control. Dependency Injection řeší vlastní způsob vložení objektů. Základní tři způsoby vložení objektů jsou Setter Injection, Constructor Injection a Interface Injection. Objekty vytvořené kontejnerem jsou nazývány [*Beans*](http://cs.wikipedia.org/wiki/Java_Bean). Objekty jsou frameworkem vytvořeny typicky na základě načtení konfiguračního souboru ve formátu XML, který obsahuje definice těchto Beans.[wiki]

Jinými slovy, Spring Framework je založen zhruba na následovné filozofii: Mezi objekty jsou různé vztahy (*dependencies,* tj. odkazy na jedné instance na instanci jinou). Pojďme tento model zkonstruovat tak, že nejdříve vytvoříme všechny potřebné objekty (bez vzájemných vztahů) a pak v druhém kroku vytvořme tyto vztahy. Nebo ještě lépe: vytvořme tyto vztahy až v okamžiku, když je budeme potřebovat. Vytváření vztahů se v Spring hantýrce říká *wiring*.

(Normálně je to tak, že jednotlivé instance, pokud obsahují odkazy na jiné, tak si je obvykle sami vytvoří, např. v rámci konstruktoru. Vnitřní instance je tak „uzavřená“ ve vnější instanci a není k ní zvenčí přístup, co může mít řadu nevýhod)

Spring Framework se nezabývá řešením již vyřešených problémů. Místo toho využívá prověřených a dobře fungujících existujících open-source nástrojů, které v sobě integruje. Tím se stává jejich použití často jednodušším.[wiki]

## Řešení přístupu do databáze

S přístupem do DB je možné se vypořádat všelijak. Dostupné je množství řešení, která můžeme rozdělit do následujících kategorií.

1. Níže-úrovňové, tedy ty, které navazují s databázovým serverem spojení, a jednotlivé operace převodu mezi světem javovských instancí a DB se komunikují v podobě SQL dotazů, které konstruuje uživatel.
2. Výše-úrovňové Tyto frameworky odstiňují uživatele (programátora) od detailů komunikace s DB. Některé i od samotné tvorby SQL dotazů. Mně osobně jsou sympatičtější technologie typu A. Jsme totiž názoru, že jednoduchost znamená minimálními prostředky popsat realitu. Pokud je však sama realita komplikovaná, snaha zjednodušit její popis nutně vede k zahalování, odstiňování vnitřních funkčních částí. Výhodou je zjednodušené používání pokud vše funguje v pořádku. V okamžiku když však systém někde uvízne, dolehne na uživatele všechna tíha, o kterou si práci ulehčil. Narazí totiž na nepřehledné předivo nejasně viditelných vztahů schovaných pod kapotou. Požadavky na produktivitu však neumožňují příliš hluboko zacházet do přístupu typu A, a tedy dá se říci, že uživatel je nepřímo nucen volit přístupy typu B. Ideálním výchozím bodem pro přístup typu B je dostatečně dlouhá práce přístupem typu A, kdy uživatele – programátora nakonec unaví donekonečna vykonávat rutinné úkony a práci si nakonec zjednoduší. Touto cestou prakticky tyto frameworky vznikají. Proti tomuto tak řečeno „přirozenému“ vývoji není prakticky závažných námitek. Problém nastává v okamžiku, když do „zjednodušeného“ systému má nastoupit nezkušený uživatel. Ten, obrazně řečeno platí pomyslný dluh za uživatele, co daný systém sestrojil. Nastává tak situace, když pro stávajícího obyvatele světa IT technologií je život čím dále, tím jednodušší, zatímco pro nově začínající programátory je vstup do tohoto světa čím – dále složitější. Situace není nepodobná světu byznysu, kdy pro majetné společnosti je čím dál jednodušší dále bohatnout, zatím co pro začínající podnikatele je situace se uchytit čím dále namáhavější. Tato monopolizace vlastnictví IT poznání – podobně jako v ostatních odvětvích implikuje 2 základní přístupy, jak se s ní vyrovnat. 1. Přístup vede k „instantnímu přizpůsobení“ tj. jakémusi zrychlenému ztotožnění se modelem. Cenou je povrchnější znalost systému, která se však projeví až později v krizových situacích. Ty pochopitelně „hasí“ zkušení matadoři, upevňujíc svoje monopolní postavení. Druhý přístup spočívá ve snaze o hlubší pochopení systému, což je však časově náročné a stejně rozdíl oproti matadorům se stírá jen částečně. Mají prostě již moc velký náskok. Naštěstí, na rozdíl od byznys-přirovnání, vědomosti není možné v plné míře zdědit, ale jen v omezené míře předávat dál, čím je zabezpečeno jaké – také znovunastolení rovnosti příležitostí v případě matadorova úmrtí. To je výhoda těch, co jsou v čele peletonu.

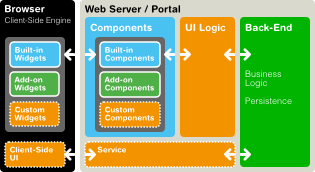
Jelikož není na škodu, když absolventská slouži i jako edukační prostředek (mám na mysli pro toho, kdo ji píše) v projektu jsem zvolil stezku A.

# Vaadin

Co je to Vaadin? Vaadin je „Java web developement Framework“ určený ke vytváření a udržování komplexních webovských aplikací. Podporuje 2 rozdílné programovací přístupy: server-side a klient-side. Nejsilnější stránkou Vaadinu je však právě tvorba aplikací pro serverovou stranu. Kde programátor prakticky využívá stejné nástroje jako při tvorbě jiných desk-topových aplikací. Nemusí se tedy vůbec seznamovat s detaily překladu do JavaScriptu a html, od kterých je úplně odstíněn.

V minulosti byla jistá propast mezi programátory „klasickými“ a programátory web. Stránek. Pokud klasický programátor chtěl udělat webovskou aplikaci, musel opustit jemu známý programovací jazyk a naučit se něco pro web charakteristických technologií (PHP, JavaScript a pod.). Jeho produktivita tedy šla dolů a nemohl se zaměřit plně jen na logiku aplikace. Vaadin se pokouší tuto propast překlenout a umožnit tvorbu web aplikací i programátorům používajícím jazyk Java.

Vaadin se tak stará o obsluhu uživatelského rozhraní a AJAX-ovou komunikaci mezi browserem a serverem. Viz. Obrázek 1.1.



Obr. 1.1

Tento obrázek ilustruje základní architekturu server-side aplikací. Architektura pozůstává ze server-side frameworku a klient-side engine, která běží na browseru, poskytující uživatelské rozhraní a doručující uživatelské akce na server. Uživatelské rozhraní aplikace běží jako Java Servlet Session na Java aplikačním serveru a klient-side engine jako JavaScript.

Protože klient-side engine je vykonávána jako JavaScript v browseru, nejsou pro použití Vaadinu nutné žádné dodatečné plug-iny do browseru. Toto poskytuje výhodu oproti jiným Framework-ům využívajících plug-iny (Flash, Java Applets, etc.. ).

Vaadin je závislý od podpory pro GWT (Google Web Toolkit), která je však běžná pro široké spektrum browserů. Takže developer se nemusí vůbec starat o podporu ze strany toho-kterého prohlížeče.

Při tvorbě aplikací pro klientskou stranu vaadin využívá GWT, ktorý poskytuje compiler z Javy do JavaScriptu, který běží na prohlížeči. Tedy tak, nebo tak developer přichází do kontaktu jedině s Javou.

Vaadin taky podporuje taky jasné oddělení mezi strukturou uživatelského rozhraní a jeho vzhledem, a umožnuje je vyvíjet separátně, nezávisle na sobě. Vaadin řídí vzhled stránek kompletně skrz tzv. *témata* které využívají CSS dokumenty, případně šablony HTML stránek.

Základní třídou ze které se ve Vaadinu vychází je třída, která dědí z třídy *com.vaadin.ui.UI*. UI je částí webovské stránky ve které Vaadinovská aplikace běží. UI je sdružená se uživatelovým *session*, které je vytvořeno pro každého uživatele pracujícího s aplikací.

Základní metodou této třídy je *init*(), která se spouští automaticky při prvním vstupu do aplikace.

## Trochu historie

Vaadin je víceméně produktem plynulého vývoje myšlenky vývojového prostředí pro tvorbu webových aplikací. Tento vývoj řekněme, že začal v roku 2000 produktem skupiny IT Mill. Tehdy měl jeho předchůdce název Millstone library. Tato měle široké využití v rámci společnosti IT Mill, na tvorbu stránek pro své klienty. Další generace, IT Mill Toolkit Release 4 v roce 2006 zavedla kompletně nový systém (engine) založený na AJAX-u. Tento umožňoval tvorbu aplikací bez starostí o komunikaci mezi klientem a serverem. Dalším milníkem byla verze IT Mill Toolkit Release 5, která pokročila ještě hlouběji do AJAX-u a klientská část bylo kompletně přepsána pomocí GWT, Google Web Toolkit. Od tohot momentu bylo možné použít Javu jak na serverové, tak i klientské straně. K šestému vydání IT Mill Toolkit-u nedošlo, nicméně v roce 2009 bylo vydáno pod novým jménem jako Vaadin 6. Zde nastává exploze uživatelů frameworku. Vydání Vaadin 7 v roce 2012 bylo zatím posledním krokem. Ve Vaadinu 7 došlo k úplnému zahrnutí GWT.

## Architektura Vaadinu

Se zaměřením aplikací na serverově straně. Nasledující obrázek ukazuje základní vztahy mezi aktéry komunikace. Vaadin Framework pozůstává ze API serverové strany, API klientské části a množství komponent uživatelského rozhraní na obou stranách. Témat, ovládajících vzhled a data modelu, implementaci logiky systému a vázání dat přímo do komponent systému. Klientská část zase zahrnuje Vaadin kompilátor, umožňující kompilaci Javy do JavaScriptu.



Serverová část aplikace běží jako servlet na Java web serveru, obsluhujíc http žádosti. Zvyčejně k tomuto účelu bývá využit VaadinServlet. Server přijímá požadavky od klienta a interpretuje je jako události dané klientské session. Události jsou asociovány s komponenty uživatelského rozhraní a doručovány k listenerům těchto událostí. Pokud klient způsobí změnu na serverové části, servlet je předá internetovému prohlížeči, který vytvoří odezvu pro uživatele aplikace. Tuto odezvu zachytí engine na klientské části a použije je na provedení změn internetové stránku, kterou uživatel zrovna prohlíží.

## Technologické pozadí, HTML a Javascript, CSS

Téměř celý web je postaven na technologiích HTML, který definuje strukturu stránky. Definuje jak grafickou i hierarchickou strukturu textu navíc umožňuje vkládání odkazů a obrázků. Vaadin používá XHTML, který je syntakticky přísnější. Používá verzi HTML 5.

JavaScript na druhé straně je programovací jazyk, který pracuje v součinnosti s HTML stránkami a je možné ho do nich implementovat. JavaScript může manipulovat s HTML stránkou skrz DOM (Document Object Model). Klient-side engine a klient.side widgets jsou zkompilovány právě do JavaScriptu, pomocí Vaadin Klient Compiler-u.

Vaadin zhusta skrývá použití HTML, dovolujíc se programátorovi koncentrovat na logiku stránek. Při aplikacích pro serverovou část UI je vyvíjeno pomocí UI komponent a přeloženo pomocí client-side engine do podoby HTML stránky.

Z webových technologií je převzeto také tvorba stylu stránek. Používá se k tomu všeobecně rozšířený jazyk CSS (cascade style sheet).

Technologie Sass(syntactically awesome stylesheets), také využívaná ve Vaadinu je rozšířením CSS. Umožňuje použití proměnných, v-hnízdění a mnoho dalších syntaktických črt, které dělají použití CSS přehlednější. Vaadin již má přichystané základní docela dizajnově propracované témata, kterých výhodou kromě toho že uživatel nemusí začínat tvorbu vzhledu stránek „na zelené louce“, ale např změnou, či rozvíjením jejich motivů je, že umožňují graficky identifikovat technologii. Z tohoto důvodu mnoho uživatelů základní témata nijak nemění.

## AJAX

Znamená Asynchronous JAvaScript and XML je technologie pro tvorbu webových aplikací podobným desktopovým aplikacím. Klasický přístup umožňuje načítat HTML stránku jen jako celek. Podstata této technologie spočívá v odesílání klientových požadavků na server asynchronně, tj. bez čekání na odezvu (tj. stránka běží dál, jako by se nic nedělo). Po přijetí odezvy ze serveru se příslušně upraví buďto celá stránka, nebo jen její část. Nedojde tedy ke „zmrznutí“ stánky, zatím co čeká na výstup ze severu.

Asynchronní žádost v AJAXu umožňuje třída *XHTMlHttpRequest* v JavaScriptu.

## Psaní aplikací pro serverovou stranu

Aplikace serverové strany běží jako Java Servlet ve servlet kontejneru. Nicméně Java Servlet API je skryto za frameworkem. Uživatelské rozhraní je implementováno jako třída UI, která musí být vždy vytvořená.

### Základní prvky aplikací UI

Reprezentuje HTML fragment, ve kterém aplikace běží ve webovské stránce. Běžný postup je, že hlavní třída aplikace dědí z UI.java. Do této třídy se pak vkládá obsah stránky. UI je původně zobrazovací pole spojené s uživatelským session dané aplikace. Běžně když uživatel otevře novou stránku s URL daného UI, vytvoří se automaticky nová instance třídy UI a asociovaný objekt „Page“. Toto všechno je asociováno se session.

### Page

Je objekt asociovaný se UI, reprezentuje webovou stránku jako i okno prohlížeče, kde běží UI. Může být přístupné skrz Page.getCurrent(), nebo pomocí UI.getCurrent().getPage().

### Vaadin Session

Reprezentuje uživatelský session. Začíná okamžikem, kdy prvně otevře UI, nebo spustí Vaadinovskou aplikaci. Končí buďto ukončením aplikace, nebo vypršením času session. Bude využito kupř. na rozeznání běžného uživatele od administrátora.

### Navigace

Vaadinovské aplikace nemají navigaci jaká je běžná u normálních stránek. Tyto běžně běží na jenom 1 stránce podobně jako všechny aplikace založeny na Ajaxu. Běžné je, že aplikace mají vícero View, mezi kterými uživatel přepíná. Tuto službu dělá ve Vaadinu třída Navigator. Views spravované pomocí něho automatický získávají specifické URI, které může být použito na přímou navigaci, skrz příkazový řádek v browseru.

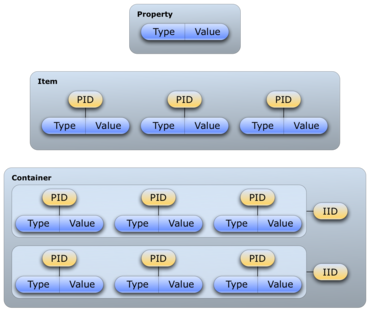
## Komponenty uživatelského rozhraní

Uživatelské rozhraní pozůstává z komponent, které jsou součástí aplikace. Jsou ukládány v hierarchickém uspořádání do tzv. *Layout* komponent, a hlavním, kořenovým layoutem, který je implicitně zastoupen v třídě UI. Nejdůležitější „koncové“ komponenty Vaadinu jsou co do typu prakticky totožné s komponenty knihovny AWT. Button, Table, TexField, ComboBox, Window, etc… Uživatel – programátor si může na základě existujících komponent sám vytvářet nové komponenty

## Vázání komponent a dat(Data *binding*)

Jednou z filosofií ve Vaadinu je odstraňovat prostředníky, co to jde. To se projevuje i v široké podpoře vázání dat s komponenty. Z pohledu MVP je to zkratka, nicméně se s MVP úplně nevylučuje (ale posouvá samotný model směrem k modelu MVC). Filozofie je ta, že změna dat, tak jak je vidí na obrazovce uživatel vede k přímé změně dat v modelu. Jsou tři vnořené úrovně hierarchie v data modelu Vaadinu. *Property, item* a *container*. Při použití analogie s datovým modelem tyto položky zodpovídají buňce, řádku resp. celé databázové tabulce.

*Data binding* představuje způsob, jak efektivně propojit View s Modelem. Vlastnost fieldu je přímo propojená s databází, a tak změna v daném poli se přímo projeví do změny hodnoty v databázi, nebo proměnné, která je na dané pole navázaná. Tento koncept, stojí na pojmech *Container*, *Property a item* kde container představuje pole hodnot, které se můžou vázat ke komplikovanější komponentě, např. ComboBoxu. Property je naopak jednoduchou hodnotou, která se váže k danému políčku. Tento koncept bude hodně využívaný i praktické části této absolventské práce. Vztah mezi *Item, property* a *containerem* je znázorněn na dalším obrázku.



### Zapojení aplikace („deploying“)

Vaadinovské aplikace jsou zapojované jako Java webovské aplikace, které můžou obsahovat množství servletů a statické zdroje, jako např. obrázky a HTML soubory. Takováto webovská aplikace je běžně zabalená do WAR souboru (s koncovkou .war). je to podtyp balíčku typu JAR (Java ARchive) a podobně jako obyčejný Jar soubor je in War ZIP-komprimovaný, se specifickou strukturou obsahu.

V hantýrce Java – Servletů *webová aplikace* značí kolekce *servletů* a *portletů,* JSP a statických HTML souborů a různých jiných zdrojových souborů, které tvoří aplikaci.