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

Diplomová práce

Vaadin - přechod na novou technologii existujícího portfolia produktů

Plzeň, 2014 Tomáš Kubový

Prohlášení

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

V Plzni dne ……. ……

Podpis

Poděkování

Na tomto místě bych rád poděkoval svému vedoucímu práce Ing. Jiřímu Kalovi za praktické rady, výbornou spolupráci a výstižné poznámky při tvorbě.

Abstract

Vaadin - conversion of an existing products portfolio into a new technology

The main aim of this diploma thesis is getting to know its readers with the Vaadin 7 Java Web Tool, and an implementation of several visual components for a presentation layer of web applications. To obtain a more productive application development, it is necessary to simplify some basic components as determine their single appearance and make them as simple as possible for using and to avoid unnecessary mistakes. Among the created elements belong tables, dialogues, or mechanism of creating forms for getting user data together with automatic validations. This dissertation mainly comes into being in virtue of gradual crossing developed applications at Marbes Consulting s.r.o. from existing technology to a new one that is just Framework Vaadin 7. Created components will be tested on the web module serving for administrative knowledge and generating CVs of employees using a template, which is part of the system for administration of human resources.

Abstrakt

Hlavním cílem této diplomové práce je seznámení s Java webovým nástrojem Vaadin 7 a implementací vizuálních komponent pro prezentační vrstvu internetových aplikací. K dosažení produktivnějšího vývoje aplikací je nutné několik základních komponent zjednodušit, stanovit jim jednotný vzhled a připravit je tak, aby bylo jejich použití co nejjednodušší a zabránilo zbytečným chybám. Mezi vytvořené prvky patří například tabulky, dialogová okna nebo mechanizmus tvorby formulářů pro získávání uživatelských dat spolu s automatickou validací. Tato práce vzniká především z důvodu postupného přecházení aplikací vyvíjených ve společnosti Marbes Consulting s.r.o. ze stávající technologie na novou, kterou je právě framework Vaadin 7. Vytvořené komponenty budou otestovány na webovém modulu sloužícím pro správu znalostí a generování životopisů zaměstnanců podle zadané šablony, jenž je součástí systému pro správu lidských zdrojů.

Obsah

1 Úvod

Společnost Marbes Consulting s.r.o. již řadu let vyvíjí komplexní řešení pro úřady státní správy a samosprávy. Řešení se skládá z řady aplikací, které jsou všechny postaveny na třívrtsvé architektuře. Aplikační server je vyvíjen v jazyce Java za podpory frameworku Java Spring. Klientská část je vyvíjena v jazyce Delphi a jedná se tak o klasické Windows Desktopové aplikace. Stále větší počet zákazníků vyžaduje webového klienta, avšak zároveň trvají na komfortu a možnostech, které jim nabízely původní desktopové aplikace. Po testování řady technologií a frameworků určených k vývoji webových aplikací byl jako vhodný kandidát ve firmě zvolen komponentový Framework Vaadin 7. Jednou z významných výhod tohoto frameworku je to, že vývoj probíhá v jazyce Java, takže je jednodušší získat nové vývojáře a také odpadá nutnost vyvíjet ve dvou jazycích a tedy i vývojových prostředích. Jde tak o zjednodušení na všech úrovních, a to jak technologických, tak i organizačních.

Důvodů pro změnu programovacího jazyka klientské části aplikací ve společnosti Marbes Consulting s.r.o. se za několik let vývoje nashromáždilo mnoho. V jazyku Delphi 2007, který je ve společnosti nyní používán, zcela chybí podpora nástrojů na sdílení zdrojového kódu (SNV, GIT). Editor zdrojového kódu již neodpovídá dnešním požadavkům na rychlost a funkce. Není podporována znaková sada Unicode (až v Deplhi 2009, ovšem z důvodu používání knihoven třetích stran nelze jednoduše přejít na novou verzi). Z technických důvodu vzniká omezení u rozsáhlých aplikací, kde dochází k vyčerpání limitu velikosti oblasti (64KB) obsahující tzv. „resource“ řetězce, které se v Delphi používají také k definici rozložení prvků každého formuláře (soubory typu .DFM). Možným řešením by bylo rozdělení aplikace do více modulů, ovšem tento proces je velmi složitý a v kombinaci s některými komponentami třetích stran dokonce nerealizovatelný. Problém nastává i ze strany hledání vývojářů, kteří umí a také chtějí vyvíjet v tomto jazyce, jelikož nezískávají perspektivní znalosti a zkušenosti pro svůj další profesní rozvoj a růst.

Cílem této práce je vývoj systémového jádra a především vizuálních komponent nad frameworkem Vaadin 7, jež postupně nahradí stávající vývojovou technologii. Grafické rozhraní ve Vaadin se skládá z množiny komponent vytvořených v programovacím jazyce Java. Hlavní částí práce je tedy implementace klíčových komponent navržených pro jednoduché použití, které budou základními prvky většiny aplikací, urychlí jejich

vývoj, a dále stanoví jednotný styl vzhledu na základě stávajících trendů v designu webových aplikací. Jedná se jak o běžné komponenty, jako jsou tabulky, tlačítka či textová pole, tak i o třídy pro tvorbu formulářů a validaci dat.

Součástí práce je ověření funkčnosti komponent a současné rozšíření aplikace pro správu znalostí a generování životopisů zaměstnanců. Z tohoto důvodu obsahuje práce i seznámení se stávajícími technologiemi používanými pro vývoj internetových aplikací spolu s nástroji pro jejich správu a údržbu.

2 Architektura a technologie aplikací

V této kapitole budou stručně uvedeny technologie používané ve společnosti Marbes, které jsou nezbytné pro vývoj, správu a údržbu aplikací a jednotlivých komponent sdílených více aplikacemi. Jedná se o nástroje, bez kterých by produkce softwaru, v dnešní době zaměřená především na rychlost vývoje, nebyla téměř reálná.

Mezi tyto základní technologie patří především platforma Java Enterprise Edition určená pro vývoj a provoz podnikových aplikací a informačních systémů. Dále nástroj Apache Maven pro správu, řízení a automatizaci sestavování aplikací. Poslední zmíněnou technologií bude systém pro správu zdrojových kódů a verzí.

2.1 Java Enterprise Edition

Informace o Java Enterprise Edition jsou čerpány především ze zdroje [6].

V dnešním podnikovém světě jsou neustále zvyšovány požadavky na rychlost a funkčnost softwaru oproti uplynulé době. Aplikace potřebují přistupovat ke sdíleným datům, zpracovávat obchodní logiku a vše přehledně prezentovat uživateli, ať již na počítači, nebo na mobilním telefonu. V poslední době jsou po aplikacích čím dál více vyžadovány funkce pro získání geografické pozice uživatele, komunikace s externími systémy nebo online synchronizace dat se službami třetích stran. Toho všeho se společnosti snaží dosáhnout za použití standardních robustních technologií, které tuto zátěž zvládnou a navíc umožní neustálé rozšiřování při snaze o zachování přehlednosti a minimalizace nákladů. A právě k tomu byl navržen framework Java EE.

První návrh Java Enterprise Edition vznikl na konci devadesátých let a přinesl do jazyku Java robustní softwarovou platformu pro vývoj podnikových aplikací. Byl zaměřen především na přední zájem společností v té době, tedy na tvorbu distribuovaných komponent. Hlavní myšlenkou bylo zjednodušení tvorby programu pomocí propojených částí s různou funkčností a vlastní zodpovědností za svou práci. Tomuto způsobu Java EE definuje rozhraní umožňující jednoduchou záměnu komponent se stejnou funkcí, ale například s rozdílným výkonem. Během vývoje se musel framework adaptovat na nová technická řešení, jakým je kupříkladu komunikace pomocí webových služeb. Postupně tedy začala platforma Java EE přebírat odpovědnost za nově přicházející technické požadavky pomocí nových standardů, díky čemuž se stávala

bohatším, jednodušším a lehčeji použitelným nástrojem pro vývoj komplexních aplikací. V současné době je nejnovější verzí Java EE 7.

Svoji výhodu má rovněž díky bezplatné licenci jádra a rozšiřujících komponent, čímž umožňuje využívat všechny své služby zdarma jak samostatným uživatelům, tak i komerčním softwarovým firmám.

2.1.1 Základní vlastnosti

Java Enterprise Edition (neboli Java EE, dříve označovaná jako Java 2 Enterprise Edition) je součást platformy Java určená pro vývoj a provoz podnikových aplikací a informačních systémů. Rozšiřuje platformu Java SE (Java Standard Edition) o podporu pro tvorbu webových aplikací, webových služeb a distribuovaných vícevrstvých aplikací. Jejím cílem je poskytnout vývojáři infrastrukturu usnadňující jejich vývoj.

Jak již bylo zmíněno výše, platforma Java EE definuje množství služeb nezbytných k tomu, aby aplikace byla škálovatelná, robustní, bezpečná a hlavně udržitelná. Služba pro bezpečnost se stará o to, aby uživatelé v systému byli těmi, jimiž tvrdí, že jsou, a aby měli přístup pouze tam, kam mají povoleno. Přístup do databáze je další základní službou, díky které můžou aplikace získávat a ukládat svá data. Právě k této funkci je nezbytná služba transakcí napomáhající k udržení konzistence dat a k jejich aktualizaci v logickém pořadí, čímž zaručí správný a zapsaný výsledek. Tyto služby jsou používané stejným způsobem, jako jsou například ve standardní edici jazyka Java užívány kolekce namísto vlastnoručně napsaných zřetězených seznamů nebo rozptylových tabulek.

2.1.2 Základní služby

Definici základních služeb předepisuje Java EE API. Jednotlivé implementace rozhraní mohou být, a většinou jsou, vyvíjeny různými společnostmi, díky čemuž má autor možnost výběru, jaké implementaci dá přednost.

- *Java Persistence API*: Standardní API pro objektově-relační mapování (ORM). Pomocí tohoto rozhraní nabízí Java EE komunikaci s databází, kdy pro různé typy databází stačí pouze vyměnit ovladač k dané databázi, zatímco programový kód zůstává beze změny.
- *Security services*: Java autentifikační a autorizační služba (JAAS) umožňuje kontrolu nad přihlašováním a přístupem uživatelů.

- *Web services*: Java EE nabízí podporu pro webové služby SOAP (Simple Object Access Protocol) a REST (Representational State Transfer). Umožňuje jednoduché vytvoření jak klientské strany, tak strany serverové, jež nabízí webové služby jiným aplikacím.
- *Dependency Injection*: Služba vedoucí k přehlednější struktuře aplikace tím, že nabízí možnost vytvoření instance objektu ve třídě pomocí definovaného XML souboru. Díky tomu lze snadněji měnit komponenty aplikace.

V předchozím seznamu je uveden pouze náhled na nabízené služby platformou Java EE. Jejich detailní popis je nad rámec této práce. Mezi další běžně používané a stejně důležité služby patří například Java Transaction API (řízení transakcí), JavaMail (možnost odesílání emailů), Java Message Service (komunikace komponent pomocí zpráv) a mnoho dalších služeb zjednodušujících vývoj i údržbu podnikových aplikací.

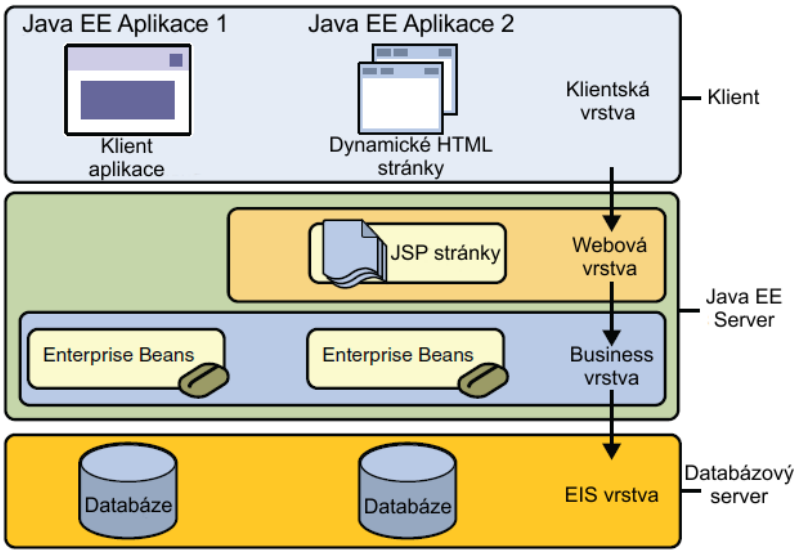
2.1.3 Běhové prostředí – Aplikační server

Serverová aplikace zastřešující všechny knihovny, které dle specifikací Java EE platformy zajišťují požadovanou funkcionalitu, je označována pojmem aplikační server. Tyto knihovny implementují veškerá API obsažená v Java EE. Kromě toho aplikační server poskytuje další klasické služby, jako např. administrátorskou konzoli, logování atp. Mezi nejznámější implementace aplikačního serveru patří například JBoss (od firmy Red Hat), GlassFish (referenční implementace od Sun) nebo TomCat (od Adobe).

2.1.4 Architektura aplikací v Java EE

Základním stavebním kamenem aplikací v Java EE je vícevrstvá architektura [7]. Při jejím použití jsou komponenty rozděleny podle svého typu do oblastí zvané vrstvy, jež často bývají instalovány na různých počítačích. Java EE používá tzv. čtyřvrstvou architekturu, jejíž jednotlivé části jsou vidět na následujícím obrázku 1.

Java EE platforma se zaměřuje na vývoj ve webové a business vrstvě běžící na aplikačním serveru. Jelikož tyto dvě vrstvy obvykle běží na stejném stroji, jsou někdy dohromady označovány jako střední vrstva.



Obrázek 1: Vrstvy architektury Java EE

Klientská vrstva (Client Tier)

Klientskou vrstvu představuje aplikace (jinak nazývána jako klient), která ani nemusí být napsaná v jazyce Java. Dva základní typy klientů jsou webový prohlížeč a klientská aplikace. Hlavní úlohou klienta je zasílat požadavky střední vrstvě, přijímat od ní odpovědi a prezentovat je uživateli.

Každý z těchto dvou typů klientů tuto funkci však provádí odlišným způsobem. Webový prohlížeč komunikuje s webovou vrstvou pomocí HTTP protokolu. Webová vrstva následně interpretuje klientské požadavky business vrstvě, jež provede jejich obsluhu. Odpověď pak putuje zpět stejným způsobem.

Klientská aplikace může rovněž komunikovat s webovou vrstvou. Na rozdíl od webového prohlížeče ji však může přeskočit a komunikovat přímo s komponentami v business vrstvě.

Webová vrstva (Web Tier)

Webová vrstva je tvořena především tzv. Java Servlety. Tyto komponenty zpracovávají klientské požadavky a generují odpověď. Ta je následně zaslána zpět do webového prohlížeče klienta. V naprosté většině případů se jedná o HTTP komunikaci. Java Servlety jsou však schopny komunikovat pomocí jakéhokoliv protokolu založeného na principu požadavek-odpověď (též request-response). Při zpracování odpovědi komunikuje webová vrstva s business vrstvou, od níž získá vypočtené výsledky operace.

Aplikační business vrstva (Business Tier)

Zde leží jádro celé aplikace. Veškerá logika a funkcionalita by měla být uložena v této vrstvě, konkrétně v EJB komponentách (objekty implementované vývojářem zajišťující vlastní aplikační logiku systému). Ty přijímají požadavky od klientské a webové vrstvy a na jejich základě pracují se zdroji z EIS vrstvy (viz následující odstavec). Následně zasílají odpověď zpět klientské, resp. webové vrstvě.

Enterprise Information System vrstva

Tato vrstva představuje veškeré externí systémy, jejichž funkcionalitu nebo data Enterprise aplikace využívá. Může se jednat například o ERP systém, databázový systém atp. Komunikace s EIS je především zajišťována přes aplikační vrstvu.

2.2 Apache Maven

Projekt Maven odstartoval pokus, který by vedl ke zjednodušení sestavení jednoho projektu firmy Apache [4]. Ten se skládal z několika komponent, kdy každá z nich měla svůj vlastní originální sestavovací ANT skript (vždy nepatrně odlišný od ostatních) a knihoven získávaných ze společného úložiště. A právě tento složitý postup pro adaptaci změn vedl k vytvoření nástroje, jenž zjednodušuje proces sestavení a údržby aplikací v jazyce Java.

Cílem Maven není ovšem pouze usnadnění procesu sestavování. Mezi další hlavní záměry definované jeho tvůrci náleží jednotnost systému sestavení, poskytování informací o projektu nebo možnost transparentního přidávání nových funkcí [12]. Dále se zaměřuje na nasměrování programátora k tomu, aby používal současné ověřené principy pro dosažení lepšího a jednodušeji udržitelného projektu. Jedná se především o strukturu rozsáhlé aplikace, v níž jsou uživatelé nabádáni např. k ukládání testů ve shodné stromové, avšak paralelní struktuře.

2.2.1 Jednotný systém sestavení programu

Maven umožňuje sestavení projektu použitím svého tzv. projektového objektového modelu (POM), což je soubor ve formátu XML definující jednotlivé části projektu a jeho závislosti na externích knihovnách a nástrojích. Tento dokument se nachází v kořenovém adresáři projektu a je pojmenován `pom.xml`. Pokud je aplikace složena z více dílčích projektů nebo modulů, každý z nich má svůj vlastní dokument dědící vlastnosti

od nadřazeného a přidává další vlastní položky. Díky této struktuře je možné sestavit celý projekt jednoduše jedním příkazem.

POM soubor

Každý POM soubor obsahuje identifikátor modulu spolu s verzí, podle něhož se Maven rozhoduje, jakou verzi modulu pro sestavení projektu použije. Dále následuje seznam závislých externích knihoven použitých v modulu, jež jsou jednoznačně definovány popisujícími atributy. Maven pak knihovny automaticky vyhledá a nainstaluje. Samotné vyhledávání probíhá v definovaných úložištích (repozitářích), a to jak vlastních, tak i ve veřejně přístupném globálním úložišti společnosti Apache. Dalším prvkem dokumentu mohou být pluginy.

Plugin

Plugin je externí program, většinou třetích stran, implementující dané rozhraní, aby mohl být spouštěn během sestavení programu. Standardní je například plugin pro spuštění kompilace zdrojových kódů nebo plugin pro spuštění testů.

2.2.2 Životní cyklus sestavení

Maven umožňuje rozdělit proces sestavení do více fází. Tímto přístupem lze definovat fázi, v níž se má spustit požadovaný plugin nebo kdy má sestavení skončit. Například během vývoje není nutné v každém sestavení programu spouštět testy nebo vkládat program do instalačního balíku, často stačí pouhá kompilace zdrojových kódu. Tím lze ušetřit velké množství času, pokud probíhá sestavování často.

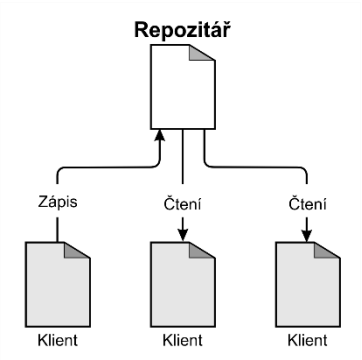
2.2.3 Poskytování kvalitních projektových informací

Další pozitivní vlastností tohoto nástroje je poskytování informací o projektu. Jedná se kupříkladu o výpis dílčích knihoven potřebných pro běh programu nebo generování sestavy výsledků jednotkových testů.

2.3 Správa implementace

Neoddělitelnou součástí vývoje aplikací je systém pro správu zdrojových kódů, který má za cíl usnadnit spolupráci více lidí na společných projektech. Skládá se především ze dvou základních prvků, a to z repositáře a pracovní kopie [11]. Repositář (Repozitory) je centrální databáze uložená na serveru obsahující soubory vyvíjeného projektu. Neuchovává se zde pouze poslední stav, ale všechny zapsané změny, jež jsou neustále dostupné všem uživatelům. Druhým prvkem je pracovní kopie (‘Working Copy‘), což je

kopie stavu projektu v centrálním úložišti určená k lokálním úpravám jednotlivými vývojáři. Systém pro správu implementace ilustruje následující obrázek 2.



Obrázek 2: Systém pro správu zdrojových kódů

2.3.1 Apache SVN

Apache Subversion (SVN) je jedním z mnoha systémů pro správu zdrojových kódů a verzí vzniklý jako náhrada za jiný hojně používaný systém, a to CVS (Concurrent Version System). SVN se snaží zachovat podobný způsob a styl práce, ale přitom odstranit jeho nedostatky, jimiž jsou například nemožnost přesunutí souboru nebo kopírování adresářů atd. Následující informace jsou čerpány z oficiální dokumentace SVN [1].

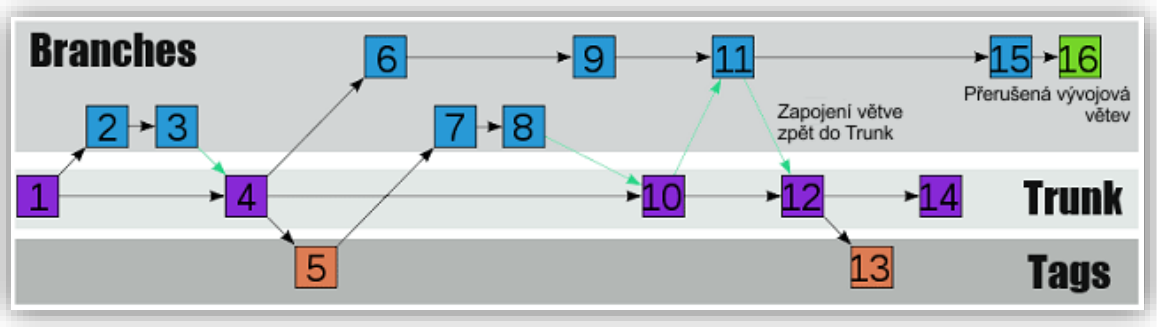
Běžný cyklus práce s úložištěm

1. Získání pracovní kopie z úložiště na klientskou stanici, kde budou probíhat změny.
2. Modifikace pracovní kopie, většinou změna zdrojového kódu.
3. Aktualizace – získání změn v centrálním úložišti, které uložil někdo jiný během vlastních úprav své kopie.
4. Další modifikace pracovní kopie.
5. Zjištění rozdílu pracovní kopie a stavu v úložišti.
6. Publikace změn v pracovní kopii do centrálního úložiště.

Každá publikace změn uložená do centrálního úložiště vytvoří novou verzi projektu mající jednoznačně definovaný stav, již je možné kdykoliv získat. V úložišti je vždy uložena jedna plná kopie dat a další verze jsou ukládány pouze v podobě změn od předchozích, než následuje další verze plná. Tento způsob podstatně redukuje velikost uložených dat v systému.

Organizace úložiště

V centrálním úložišti jsou většinou uloženy všechny projekty najednou. Je tedy nezbytně nutné zajistit nejen dokonalou zálohu, nejlépe několika způsoby, ale také udržovat v datech určitou strukturu. Díky tomu bude i po letech jasné, jak se projekt vyvíjel a jak vypadají verze pro jednotlivé zákazníky.



Obrázek 3: Ukázka organizace projektu v úložišti

Na obrázku 3 je zobrazena struktura a postupný vývoj úložiště jednoho projektu. Každý z projektů má v centrálním úložišti podobnou strukturu tří hlavních složek, do nichž jsou postupně ukládány změny. Každá změna má jednoznačně definované číslo revize (angl. revision), podle kterého je možné získat z úložiště projekt v přesně takovém stavu, v jakém byl při uložení do systému v daném časovém okamžiku.

- **Složka Trunk**: V této složce je vždy uložen nejaktuálnější stav projektu, jenž je postupně vyvíjen až do majoritní verze. Je to místo sloužící pro postupné zaznamenávání změn a nemělo by být označováno verzemi nebo jmény.
- **Složka Branches**: Jedná se o odbočení (větev) od hlavního vývoje. Například, pokud je požadováno místo pro experimentování nad aktuálním kódem nebo pro dosažení nějakého specifického funkčního cíle, probíhá vývoj nad touto větví. Tato odbočka by měla být také používána před vydáním verze, například pro opravu chyb nalezených při testování, aby byla oddělena hlavní vývojová linie.
- **Složka Tags**: Tagy jsou podobné odbočkám (větvím), ovšem neslouží pro ukládání změn. Jedná se o větve sloužící pouze pro čtení, jež uchovávají stav zdrojového kódu typicky v okamžicích vydání verze. Lze je opatřit vlastním názvem čitelným lépe, než je pouhé číslo revize.

2.4 Prezentační vrstva

Jak bylo uvedeno v odstavci 2.1.4, prezentační vrstva slouží k zajištění služeb uživatelského rozhraní [3]. Uživatelské rozhraní (UI – User interface) je závislé na platformě nebo operačním systému, na němž je aplikace provozována. Může se jednat o webového klienta, konzolovou aplikaci, mobilního klienta a další. Uživatelské rozhraní je tvořeno ovládacími prvky, pomocí nichž má uživatel možnost aplikaci obsluhovat (proto bývá také označováno zkratkou GUI – z angl.. Graphical User Interface). Implementace jednotlivých ovládacích prvků může být různě složitá pro rozdílné platformy, ovšem většinou platí, že čím složitější uživatelské rozhraní je, tím jeho funkce více závisí na konkrétní platformě a je obtížnější vytvořit takovou aplikaci, která by se na různých systémech chovala stejným způsobem a vypadala identicky. Například webové rozhraní má sice své ovládací prvky jednodušší proti desktopovým, ovšem na druhou stranu je zobrazitelné na všech platformách majících k dispozici webový prohlížeč.

2.4.1 Zodpovědnost vrstvy

Jako každá jiná vrstva má i prezentační definovanou zodpovědnost. Jak bylo uvedeno výše, jedná se především o služby uživatelského rozhraní. Prezentační vrstva by měla sloužit pouze pro zobrazování dat, zachycování uživatelských událostí (akce při stisknutí klávesy/tlačítka nebo pohybu myší atp.) a odesílání a přijímání HTTP požadavků. Nikdy by neměla měnit hodnoty dat nebo řídit vykonávání kódu. V takovém případě by ani nemělo smysl aplikaci ve vrstvách implementovat. Dodržení těchto omezení vede k jednoduššímu návrhu několika různých řešení zobrazujících data pro různé platformy.

2.4.2 Oddělení vrstev

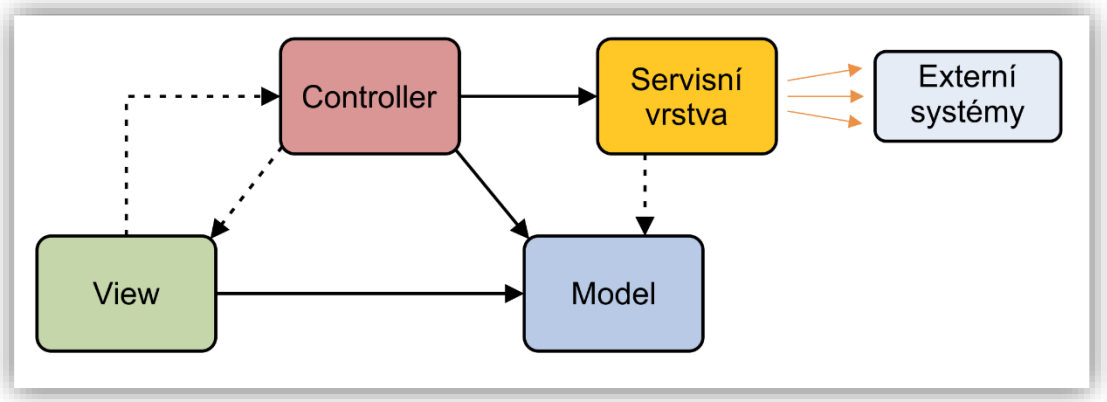
V rozsáhlých aplikacích je běžné, že obsahují jednu datovou vrstvu, jednu doménovou vrstvu a jednu nebo více vrstev prezentačních. Právě z toho důvodu je více než vhodné mít vrstvy dostatečně oddělené. Prezentační část by měla se servisními třídami komunikovat především přes rozhraní a obsahovat vlastní nezávislé datové modely. Stejně tak je i z druhé strany vrstva oddělena, a to klientem. Pokud se jedná o internetovou aplikaci, kdy je prezentační vrstva provozována na serveru, komunikace s ní probíhá pomocí tzv. tenkého klienta.

2.4.3 Tenký a tlustý klient

Tenký klient přistupuje k aplikaci přes webové grafické uživatelské rozhraní, v němž každá interakce uživatele může vynutit odeslání požadavku na webový server (v tomto případě k obsluze v prezentační vrstvě) [3]. Tato architektura vyžaduje trvalé připojení k webovému serveru, a proto není vhodná pro aplikace požadující použití bez připojení k síti. V takovém případě je většinou nutné přesunout část nebo celé vrstvy aplikace na klientský počítač. Klient je potom nazýván jako „tlustý klient“ a obsahuje uživatelské rozhraní, v němž není potřeba komunikace přes HTTP protokol nebo webové služby. Obsluhovat aplikace je poté typicky možné i bez internetového připojení.

2.4.4 Architektonický vzor prezentační vrstvy – MVC

Jedním z nejčastěji využívaných architektonických vzorů je označován zkratkou MVC [2]. Dělí prezentační vrstvu na tři logické části tak, aby je šlo upravovat samostatně a aby byl dopad změn na ostatní části co nejmenší. Tyto tři části se nazývají Model, View a Controller. MVC architektura je velmi podobná architektuře popsané v kapitole 2.1.4 a lze ji použít jednak pro celou aplikaci, tak po menších úpravách i pouze pro prezentační vrstvu oddělenou od business vrstvy, což bude uvažováno v našem návrhu (viz obrázek 4).



Obrázek 4: Architektura MVC prezentační vrstvy

- **Model:** Model je datová struktura obsahující data pro zobrazení v části View. Jeho součástí může také být nějaká základní doménová logika na datech, ovšem pouze pro prostředí Controlleru a View, jelikož ta samá a další logika musí být i v servisní vrstvě.

- **View:** Úkolem View (vzhled/pohled) je převádět data reprezentovaná modelem do podoby vhodné k interaktivní prezentaci uživateli. View může být například HTML šablona se vstupními poli naplněnými daty z modelu.
- **Controller:** Controller (kontrolér) je používán k manipulaci s daty. Slouží k transformaci datových objektů získaných ze servisní vrstvy na modely a naopak. Dále obsluhuje požadavky (události) vytvořené uživatelskou interakcí, například aktualizací modelu a předání informace View o hotové změně.
- **Servisní vrstva:** Servisní vrstva je implementována tak, aby byla nezávislá na konkrétní prezentační vrstvě. Díky tomu jde jednoduše prezentační vrstvy měnit nebo rozšiřovat o další řešení. Objekty servisní vrstvy jsou typicky skryté za rozhraním, aby bylo možné Controller testovat i bez použití reálných externích systémů.

Vzor MVC je obecný návod, jak by měla prezentační vrstva vypadat a jak by v principu měla fungovat. Její přesná implementace ovšem závisí na konkrétní technologii. Příkladem může být rozdílný způsob, jakým jsou vytvářeny instance objektů, zda má technologie podporu pro data-binding[1] nebo jestli jsou metody pro přístup k datům synchronní či asynchronní.

2.4.5 Komponentně orientovaná prezentační vrstva

Komponentový model na webové vrstvě přináší oproti MVC se standardním šablonovým systémem možnost elegantního znovupoužití mnoha částí kódu s různými daty bez jeho změny [10]. Tato možnost je samozřejmě z části proveditelná i s pomocí běžných šablon, ať již v podobě JSP stránek nebo pomocí šablon nástroje FreeMarker[2], ovšem ladění jejich funkčnosti na různých typech dat je daleko složitější a méně přehledné než u komponent. Správně navržené komponenty vedou k podstatně rychlejšímu a „přehlednějšímu“ vývoji.

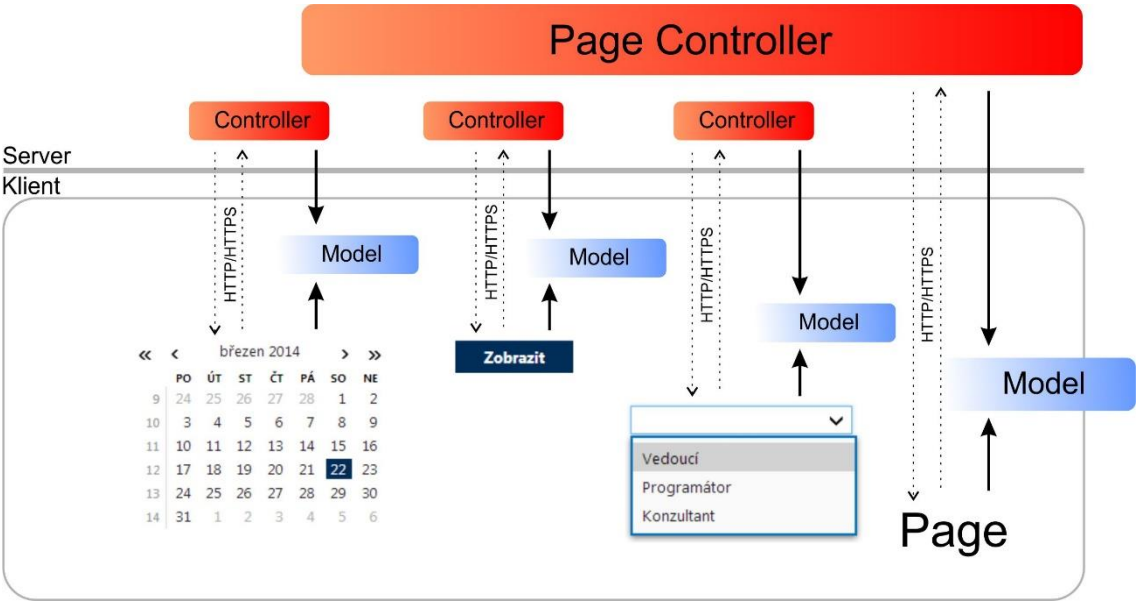
Pod pojmem komponenta si lze představit jednotlivé vizuální části webové stránky. Jedná se například o nadpis, odkaz, tabulku, obrázek a všechny další stavební prvky, které jsou potřebné k funkčnosti stránky. Každá taková komponenta „žije“ vlastní život. Má

[1] Svázání datového objektu s editačními poli pro automatickou synchronizaci dat

[2] FreeMarker je šablonovací systém pro jazyk Java založený na šablonách umožňujících generování čehokoliv od HTML po automaticky generované zdrojové kódy. Nejedná se o aplikaci pro koncové uživatele, je to nástroj pro generování výstupů uvnitř programů. (více informací na adrese http://freemarker.org)

vlastní kontrolér a vzhled, aby mohla být nezávislá na ostatních komponentách. Není nutné, aby jednotlivé komponenty o sobě věděly, pokud spolu nepotřebují komunikovat.

Vzhled webové stránky je tvořen většinou buď definováním struktury s informacemi o rozmístění komponent v XML souboru, nebo přímo ve zdrojovém kódu, kde jsou jednotlivé komponenty seskupovány do bloků (tzv. layoutů, které jsou mimo jiné také komponenty) a vytváří tak hierarchii, podle níž je při dotazu sestavena příslušná HTML odpověď pro prohlížeč. Komponenty jsou v layoutu rozmísťovány podle jeho vlastností a typu. Například tak, že všechny prvky v bloku budou vkládány do řádku nebo naopak do sloupce. Samozřejmostí správného zobrazení je definování ať již relativní nebo absolutní velikosti jednotlivých komponent.



Obrázek 5: Architektura komponentní prezentační vrstvy

Jak již bylo zmíněno, každá komponenta má svůj kontrolér, model a vzhled, jak je vidět na obrázku 5. Díky tomuto návrhu již není nezbytné, aby jedna adresa (URL) reprezentovala celou stránku. Může reprezentovat pouze její část nebo konkrétní akci. Není proto nutné po každé události načítat znovu celou stránku. Komponenta, na které událost vznikla, odešle vlastní dotaz na server a postará se o reakci na odpověď, například pouze aktualizací svého stavu. Ostatní komponenty nemusí vědět, že se jiná změnila a zůstávají ve stavu jako před akcí. Právě tento přístup komponentového frameworku umožňuje jednodušeji vytvářet tzv. bohaté internetové aplikace (RIA – Rich Internet Application), jež se svým vzhledem a interakcí s uživatelem přibližují aplikacím desktopovým, i když běží v internetovém prohlížeči.

3 Vaadin

Informace o frameworku Vaadin byly převzaty především z výjimečně kvalitní průvodní dokumentace [9].

Slovo Vaadin v překladu z finského jazyka znamená „požaduji“. A právě tím se snažili vývojáři vstoupit do širšího povědomí dalších uživatelů, když přejmenovali projekt založený v roce 2002 na toto slovo. Cílem nástroje Vaadin je totiž vystihnout požadavky programátorů a uživatelů. Programátoři chtějí především nástroj, pomocí něhož budou schopni efektivně vyvíjet internetové aplikace, jež jsou v poslední době na vzestupu. Požadují snadno udržitelný, platformně nezávislý jazyk, ale zároveň dostatečně výkonný na to, aby i starší servery zvládly jejich provoz. Na druhé straně přístup uživatelský požaduje především jednoduchost a přehlednost internetových stránek. To již není ovšem otázkou konkrétního programovacího jazyka, ale schopností vývojáře porozumět myšlení běžného uživatele a navrhnout mu přívětivé prostředí. I tento požadavek je Vaadin schopný pokrýt, především pomocí široké nabídky volně dostupných komponent, které lze na své stránky umístit, a využít tak otestované a žádané prvky.

Framework Vaadin je sada komponent a nástrojů pro snadný vývoj rozsáhlých webových aplikací v jazyce Java. Pro jednoduché internetové stránky nebo pro reklamní upoutávky není vhodný. Vaadin je vyvíjen především ke zjednodušení tvorby a údržby kvalitních webových uživatelských rozhraní a systémů, typicky intranetových aplikací. Hlavní myšlenkou je rozdělení jednotlivých komponent na dva odlišné modely, a to na stranu serverovou a stranu klientskou, kde právě strana serverová nabízí sílu vývoje. Díky ní je možné z části zapomenout na internetové stránky a programovat uživatelské rozhraní jako pro klasický počítačový Java program, ovšem podstatně jednodušeji než nabízí například Java Swing. Dokonce dovolí vytvořit webovou aplikaci bez znalosti jazyků HTML a JavaScript. Sice bez využití všech schopností Vaadin, ale plně funkční s klasickými potřebnými webovými komponentami, jako jsou obrázky, odkazy, formuláře a mnoho dalších.

Snahou a cílem Vaadin je potlačení limitů vlastních možností. Pokud z nějakého důvodu komponenta nepodporuje právě to, co uživatel vyžaduje, musí být jednoduché tuto funkčnost přidat nebo rozšířit.

Na rozdíl od frameworků jako Flash, Java Applets a podobných, klientská strana aplikace spouští v prohlížeči pouze JavaScript a není zapotřebí žádných speciálních pluginů. Vaadin využívá pro vykreslení uživatelského rozhraní framework GWT (Google Web Toolkit) [13]. Java zdrojový kód je následně překompilován do JavaScript. To výrazně usnadňuje vývoj a zároveň vede ke kompatibilitě mezi různými prohlížeči. Výsledné prvky jsou překládány do několika verzí kvůli rozdílným požadavkům a funkcím běžně dostupných prohlížečů, jako jsou Google Chrome, Mozilla Firefox, Internet Explorer a další, čímž v podstatě zaručují stejné zobrazení a dále odprostí programátora od nutnosti vytvářet vlastními silami několik verzí pro různé prohlížeče. GWT je vhodné pro tvorbu pokročilých komponent uživatelského rozhraní a implementují logiku, která má být vykonávána v prohlížeči. Takto vytvořené a přeložené komponenty umožňují rozšířit vlastní aplikaci o volně dostupné prvky třetích stran, tzv. add-ons („doplňky“).

Framework Vaadin definuje jednoznačnou linii mezi strukturou uživatelského rozhraní a jeho vzhledem, čímž dovoluje jejich oddělený vývoj. Souborům ovlivňujícím vizuální vzhled aplikací, se říká *themes* (motivy). Ty definují vzhled uživatelského rozhraní pomocí CSS a HTML šablon. Samotný Vaadin poskytuje základní motivy přímo v jádře frameworku a umožňuje jejich rozšíření či přepsání vlastními styly.

3.1 Historie

Vývoj Vaadin nejprve začal jako součást webového frameworku Millstone 3 s otevřenou licencí v roce 2002 [5], kdy představil klientskou část komunikující pomocí AJAX. Následně pokračoval vývoj odděleně jako komerční produkt a začátkem roku 2007 byla vydána nová verze pod názvem IT Mill Toolkit 4. Ta používala vlastní složitou JavaScript implementaci pro dynamické vykreslování, čímž se stala komplikovanou pro implementaci nových komponent. Proto ještě koncem roku 2007 byla tato implementace nahrazena nástrojem GWT. Ve stejném čase prošla licence změnou na open source Apache Licence 2.0 a první produkční verze byla vydána asi po roční beta verzi pod názvem IT Mill Toolkit 5.

Další milník vývoje přišel v květnu roku 2009, kdy byl současný název přejmenován na Vaadin Framework, jenž má být atraktivnější a má vést k rozšíření komunity zajímající se o tento produkt. Současně byla vydána verze Vaadin 6 a spuštěny komunitní stránky s podrobnou dokumentací pro začínající vývojáře. Poslední velkou změnu zaznamenal

framework v roce 2013 vydáním verze Vaadin 7 obsahující téměř 70 nových funkcí. V současnosti je nejaktuálnější verze Vaadin 7.1.11 nabízející například okamžité informování klientů o změně stavu na serveru.

3.1.1 Vaadin 6 a Vaadin 7

Autoři Vaadin 6 počítali s možností, že bude možné vyvíjet uživatelské rozhraní pro více klientských technologií (webové rozhraní, desktop a další), ačkoliv byl vyvinut pouze HTML klient. S příchodem verze Vaadin 7 byla tato možnost zrušena a odebrána z aplikačního rozhraní související části, čímž byla podstatně zjednodušena struktura [9]. Jak bylo uvedeno v předchozím odstavci, mezi šestou a sedmou verzí bylo změněno několik funkcí. Mezi ty hlavní patří urychlení vykreslování stránek, kdy nejsou pozice a velikosti komponent počítány pomocí Javascriptu, ale výhradně přes HTML 5 a definované hodnoty v kaskádních stylech (CSS). Kompletně byla přepracována obsluha formulářů, přidána podpora více záložek v prohlížeči nebo možnost vkládání vlastního JavaScriptu do klientských částí komponent jednoduchým oficiálním způsobem. Podstatně byl také zjednodušen vývoj nových komponent.

3.2 Porovnání s dalšími frameworky

V dnešní době existuje značná řada Java frameworků podporujících tvorbu webových aplikací. Často jsou jednotlivé nástroje velmi odlišné, typicky proto, že byly navrženy pro různé cíle. Vybrat si ovšem správný nástroj pro vlastní projekt není jednoduché. Rozhodnutí ovlivňuje několik faktorů, jako jsou rozšiřitelnost, rychlost vývoje, dokumentace jazyka, bezpečnost a další. Detailnější porovnání několika frameworků je dostupné na [8].

3.2.1 ZK framework

Framework ZK je stejně jako Vaadin založen na jazyku Java a uživatelským rozhraním skládajícím se z komponent. Tyto komponenty nejsou kompilovány do jazyka JavaScript, ale přímo do JQuery, čímž umožňuje jednoduchou integraci JQuery pluginu přímo do stránky. I zde je hlavní předností skrytá komunikace před programátorem mezi klientskou a serverovou stranou pomocí AJAX zpráv. ZK podobně jako Vaadin podporuje všechny hlavní návrhové vzory, především MVC, data-binding a další.

Výhody ZK

Výhodou tohoto frameworku je možnost definovat rozložení komponent na stránce jak zdrojovým kódem v jazyce Java, tak přes strukturu v souboru v XML formátu. Rozmístění komponent v aplikaci pomocí XML značek dovoluje vytvořit několik různých souborů se strukturami rozložení komponent. Díky tomu lze změnit vzhled aplikace bez zásahu do zdrojového kódu a nutnosti nového překladu programu. To vše vede k jednoduché implementaci tzv. „responzivního designu“, kdy jsou velikost a rozložení komponent dynamicky měněny za běhu podle velikosti okna, v němž je aplikace prohlížena.

Výhody Vaadin

Pozitivem pro Vaadin je rozhodně jeho bezplatné používání v plném rozsahu díky vydávání pod licencí Apache 2.0, zatímco ZK používá limitovanou komunitní a plnou komerční verzi. Avšak i komunitní verze dostačuje pro vývoj aplikací. Další výhodou je dostupnost několika základních motivů snažících se napodobit vzhled desktopových aplikací, od nichž je možné dědit styly s minimálními úpravami. ZK využívá vlastí nástroj pro vykreslování uživatelského rozhraní, zatímco Vaadin pracuje na rozšířeném frameworku GWT a obsahuje podstatně více komponent třetích stran jednoduše zapojitelných do vlastních aplikací.

3.2.2 Apache Wicket

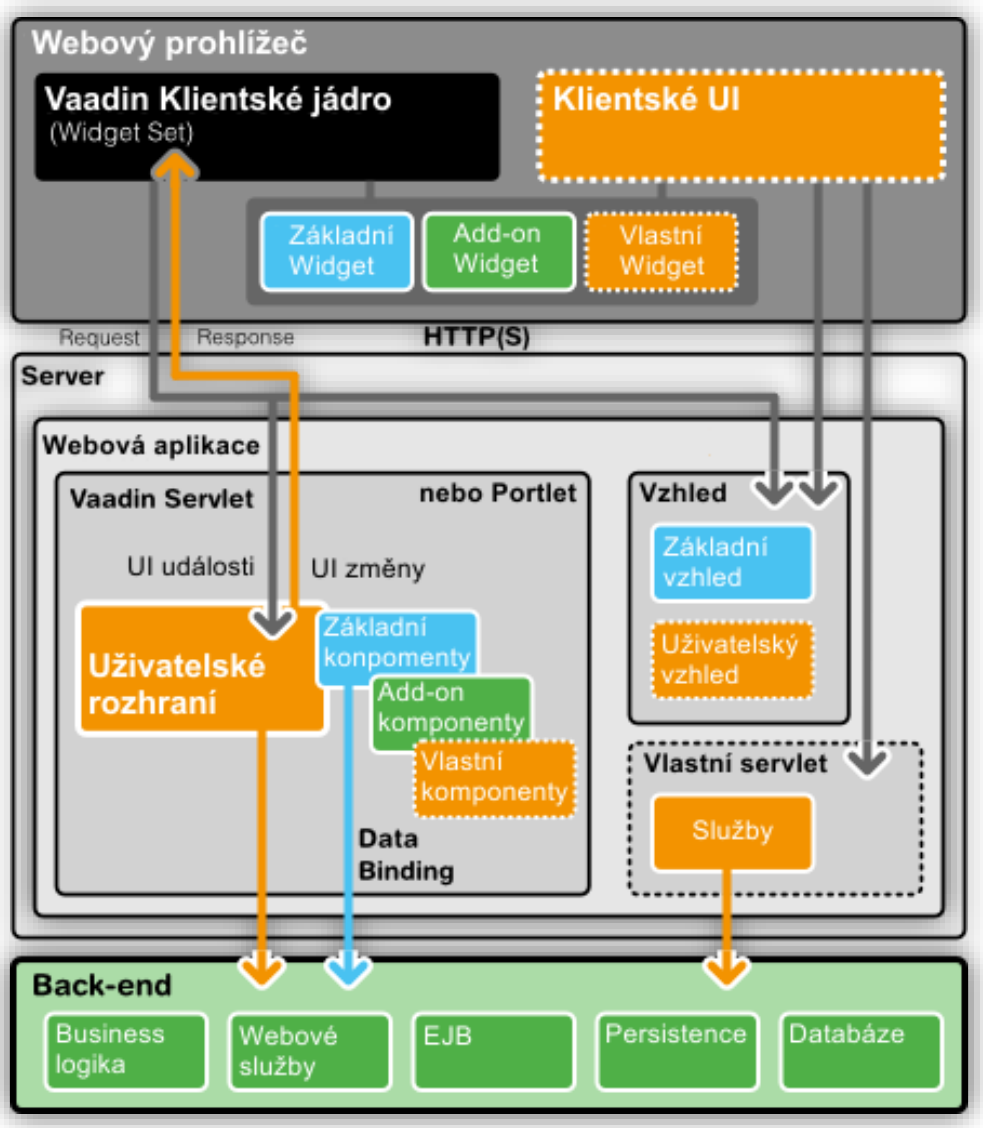
Apache Wicket je další komponentně řízený framework podobný Vaadin, jenž na rozdíl od ostatních využívá pro definování prvků HTML šablony s vlastními značkami a s možností AJAX komunikace. Tento přístup nutí používání a znalost více programovacích jazyků než jen Java, s níž si ve frameworku Vaadin ve většině případů programátor vystačí. Nejde samozřejmě pouze o znalost jazyků, ale i rychlost vývoje, jelikož ve Wicket je nezbytné napsat mnoho řádků navíc nejen v kódu Java, ale i v HTML, CSS nebo JavaScript.

3.2.3 Další frameworky

V současnosti existuje několik dalších frameworků pro vývoj webových Java aplikací, ovšem jejich detailní porovnávání by bylo vhodnější pro samostatné téma diplomové práce. Mezi další používané patří: Spring MVC, JSF, Struts, GWT, Grails a další.

Obecně má Vaadin nad ostatními frameworky výhodu především v komunitě vývojářů. Obsahuje velmi detailně propracovanou programátorskou dokumentaci, ve které jsou zmíněny všechny základní možné konstrukce nutné pro vývoj. Díky tomu je snadnější začít vytvářet aplikace. Dokumentace je v neustálém vývoji, stále přibývají návody a popis nových funkcí. Ve stejné kvalitě je i fórum, na němž je možné se obrátit na autory nebo odborníky s problémy. Ve většině případů je odpověď velmi rychlá a nápomocná.

3.3 Architektura Vaadin aplikací



Obrázek 6: Architektura aplikace ve Vaadin

Obrázek 6 zobrazuje architekturu webové aplikace vytvořené ve Vaadin. Skládá se ze serverové části, kde se nachází perzistentní a logická vrstva aplikace, logická část

komponent a jednotlivé komponenty. Druhou částí je klientská strana. Ta je překládána do jazyku JavaScript vykonávaného běžným prohlížečem internetových stránek. Uživatelova interakce je z části obsluhována přímo na klientské straně a pro složitější úkony odesílána pomocí asynchronních AJAX zpráv serverové straně. Ta zpracuje jednotlivé požadavky a vrátí stejným způsobem odpověď zpět prohlížeči, jenž se postará o aktualizaci dat.

Díky HTML, JavaScript a dalším technologiím skrytým před aplikační logikou, je možné na prohlížeč hledět jako na tenkého klienta. Ten zobrazuje uživatelské rozhraní a komunikuje se serverovou částí po nižších vrstvách. Logika uživatelského rozhraní běží na aplikačním serveru v podobě servletů spolu s aplikační logikou. Naproti tomu standardní architektura klient-server s klientskou aplikací může obsahovat spoustu aplikačně specifické komunikace mezi klientem a serverem. V podstatě odebrání prezentační části z aplikační architektury vede k efektivnějšímu přístupu dovolujícímu vytvářet tzv. RIA aplikace (Rich Internet Applications – „Bohaté internetové aplikace“) stejně interaktivní jako desktopové.

Hlavní části architektury a jejich funkce:

- **User interface**: Aplikace nabízí uživateli rozhraní pro interakci s aplikační logikou a daty. Jedná se o vrstvu reagující na vstupy uživatele s možností se zobrazit v prohlížeči nebo umístit do HTML stránky.
- **Komponenty/Widgety**: Uživatelské rozhraní se skládá z komponent vytvářených a rozmisťovaných v aplikaci. Každé komponentě odpovídá na klientské straně „widget“, podle nějž je vykreslena v prohlížeči. Přes widgety obsluhuje uživatel aplikaci.
- **Client-Side engine**: Prvek mající na starost vykreslování uživatelského rozhraní v prohlížeči pomocí widgetů. Komunikuje se serverovou částí výměnou HTTP a HTTPS požadavků a odpovědí.
- **Vaadin Servlet**: Třída obsluhuje požadavky od několika klientů. Každý uživatel má svůj datový prostor, který Servlet určí podle cookies přijatých v požadavku.
- **Themes**: Téma obsahující CSS styly aplikace a jednotlivých komponent. Dále zde mohou být umístěny HTML šablony definující vlastní rozložení komponent nebo soubory, jako například obrázky.

- **Back-end**: Aplikační vrstva aplikace oddělená od prezentační vrstvy.
- **Klientské aplikace**: Vaadin umožňuje vytvářet aplikační moduly běžící samostatně v prohlížeči. Moduly používají stejné widgety, témata a aplikační vrstvu jako serverové aplikace. Jsou vhodné pro náročná, rychle reagující uživatelské rozhraní, jakou jsou například burzovní aplikace.

3.4 Události a jejich obsluha

Vaadin nabízí událostně řízený model pro obsluhu uživatelské interakce. Po provedení nějaké akce uživatelem v uživatelském rozhraní, jako například stisknutí tlačítka nebo vybrání položky seznamu, je potřeba informovat server o změně stavu. Stejně jako další webové Java frameworky, tak i Vaadin používá návrhový vzor „Událost-Posluchač“ (také známý jako návrhový vzor „Observer“) pro doručení uživatelských dat do vrstvy s aplikační logikou. Tento princip je složen ze dvou částí, a to z objektu generujícího události a několika posluchačů, kteří čekají na událost. Po vzniku události informuje objekt všechny posluchače. Ve většině případů je posluchač pouze jeden.

Události mohou sloužit k různým účelům. Ve Vaadin je běžný účel reakce na události vzniklé v uživatelském rozhraní. Ovšem například správa relací může vytvářet zcela odlišné události, jako je vypršení času na interakci s aplikací vedoucí ke smazání uživatelovy relace, jelikož po zvolenou dobu neprovedl žádnou akci. Časové události mohou nastávat ve zvoleném čase nebo periodicky po uplynutí určité doby.

Aby mohl posluchač reagovat na událost, musí implementovat rozhraní obsahující metodu, jež je zdrojem události zavolána. Před tím se ovšem ještě musí posluchač zaregistrovat do seznamu, zpravidla pomocí metody `addListener()`. Většina komponent definuje vlastní rozhraní pro posluchače a vlastní typy událostí, jak je naznačeno v ukázce zdrojového kódu 1.

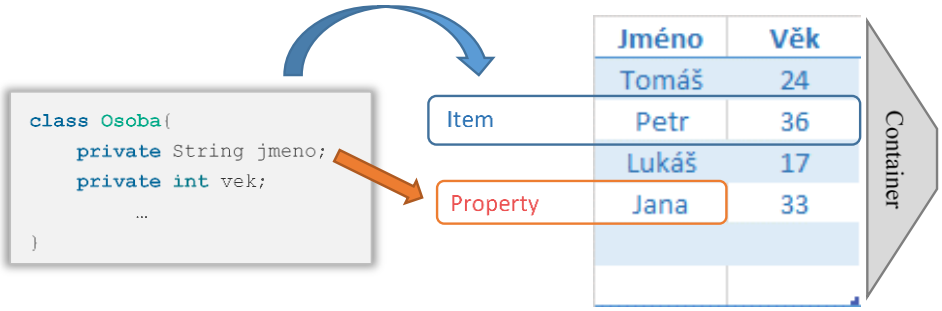
```
final Button button = new Button("Uložit");
button.addClickListener(new Button.ClickListener() {
    public void buttonClick(ClickEvent event) {
        button.setCaption("Uloženo");
    }
});
```

Zdrojový kód 1: Obsluha události pomocí anonymní třídy

3.5 Data binding

Datový model Vaadin je jeden ze stěžejních konceptů knihovny. Aby mohly vizuální komponenty přistupovat k modelu přímo, je definováno několik rozhraní zprostředkovávajících tuto funkčnost. Model dovoluje svázání konkrétních dat s konkrétním prvkem komponenty, který může data zobrazovat a jednoduše editovat. Hlavní konstrukční částí tohoto mechanizmu jsou tři zanořené úrovně v hierarchii datového modelu. Jedná se o Property, Item a Container. Analogicky si pod těmito pojmy lze představit buňku, řádek tabulky a samotnou tabulku.

Datový model je používán hlavně v jádře Vaadin komponent, především v komponentách zprostředkujících vstupní pole uživateli. Těmi jsou třídy implementující rozhraní `Field` nebo rozšiřují třídu `AbstractField`, jež definuje několik běžných vlastností. Právě tyto prvky je možné svázat mezi sebou. To přináší spoustu výhod, především jednoduchost a přehlednost vývoje formulářů na webových stránkách.



Obrázek 7: Ukázka využití data-binding

Na obrázku 7 je naznačeno základní použití návrhového vzoru data-binding. Pro ukázku si lze představit jednoduchou třídu „Osoba“, jež je datovým modelem. Ta obsahuje proměnné hodnoty `jmeno` a `vek`, jež musí mít pro správnou funkčnost definované příslušné metody typu get a set[3] podle Java konvencí. Proměnné hodnoty každé osoby představují jednotlivé `Property`, jimiž jsou, jak již bylo uvedeno, položky prvku `Item`. Ty v našem případě reprezentují jednu instanci osoby. Následuje naplnění kontejneru osobami a nastavení komponentě zobrazující seznam osob jako datový zdroj právě připravený kontejner. Tento mechanizmus nám zajistí transparentnost úprav. Pokud je přidána nebo odebrána osoba z kontejneru, automaticky se změna promítne na druhou

[3] Veřejné metody pro čtení a zápis hodnot do privátních proměnných třídy.

stranu, kde dojde k překreslení komponenty. Stejným způsobem, například při změně jména osoby v komponentě, dojde k úpravě dat v modelu.

S daty lze dále pracovat několika způsoby. Například může kontejner položky průběžně řadit, filtrovat či indexovat podle našich požadavků, a tím měnit pořadí vykreslení hodnot v komponentě. Na jednotlivé `Property` je možné přidat validátor, který zajistí kontrolu uživatelem zadaných dat, jako je správný formát emailové adresy nebo zadání celého čísla v dotazu věku osoby a ne textu. Dále může zprostředkovat tzv. lazy loading (postupné načítání dat), kdy jsou komponentě odeslány ze serveru pouze položky na první stránce seznamu a další až poté, kdy je uživatel požaduje (rolováním položek v seznamu).

3.6 Add-ons

Jádro knihovny Vaadin je navrženo k maximální možnosti jednoduchého rozšíření prvků třetích stran, ať již komponent, stylů nebo užitečných nástrojů. Add-on je doplněk, který může být jednoduše sdílen mezi projekty a rozšiřovat jejich funkčnost. Většinou se jedná o doplňky jiných vývojářů, kteří chtějí nabídnout svoje komponenty ostatním lidem a oprostit je tak od nutnosti vytvářet celé komponenty vlastními silami.

Pro zjednodušení sdílení doplňků bylo v roce 2010 autory Vaadin spuštěno úložiště Vaadin Directory, kam může každý umístit své vlastní implementace. Doplňky jsou tříděny do skupin podle funkčnosti a lze mezi nimi vyhledávat podle klíčových slov. Registrovaní uživatelé mají možnost ohodnotit prvek jednou až pěti hvězdami, čímž dávají ostatním najevo svoji spokojenost či nespokojenost s použitelností. Velmi jednoduchá je integrace do vlastního projektu. Pokud je projekt sestavován pomocí Maven, stačí pouze přidat závislost na doplněk do konfigurace projektu. Další možností je stáhnout JAR balík a připojit doplněk k projektu manuálně. Většina sdílených doplňků je přístupných s bezplatnou licencí, ovšem není to pravidlem. Pro některé je nutné licenci zakoupit.

Známým oficiálním placeným doplňkem jsou například Vaadin Chart). Jedná se o knihovnu tříd, pomocí nichž lze na stránky vkládat statistické grafy. Na výběr je několik standardních sloupcových či kruhových šablon pro zobrazení statistických dat nebo časových řad. Grafy jsou do stránky vkládány ve formátu SVG (anglicky Scalable Vector Graphics, škálovatelná vektorová grafika) a obsahují jednoduché efektní animace při načítání a interakci s uživatelem.

4 Vývoj komponent pro webový framework Vaadin 7

Framework Vaadin obsahuje již v základním balíku mnoho komponent, pomocí nichž lze bez sebemenších problémů vytvořit i složitější webovou aplikaci a dále má definovány tři motivy základního vzhledu. Všechna tato témata se snaží dodat stránkám podobu desktopových aplikací. I přesto je nutné při komerčním vývoji většinu záležitostí týkajících se vzhledu a funkčnosti změnit. Každá společnost zabývající se vývojem internetových aplikací má vlastní motivy, ať již se to týká barev tlačítek a odkazů nebo zaoblení okrajů jednotlivých komponent.

Cílem vytvoření vlastních nebo upravení komponent je definování jednotného stylu vzhledu všech komponent potřebných k vývoji aplikací a dále připravení programátorům takových prostředků, jež maximálně zefektivní a urychlí jejich práci. Jde o to, aby se vývojáři nemuseli zabývat stále stejnými oblastmi kódu, aby nemuseli u každé komponenty zadávat její barvu a velikost. Cílem je co nejvíce zapouzdřit komponenty tak, že je bude stačit pouze umístit do layoutu a zadat jejich nezbytné vlastnosti (například u tlačítka obsluhu stisku). Této části je dobré věnovat ve firmě dostatek času, jelikož při správném navržení komponent umožňujících jejich maximální znovupoužitelnost je urychlení vývoje v porovnání s psaním stránek v klasickém HTML obrovské.

V této kapitole bude následovat popis nutných a některých dokonce nezbytně nutných komponent k přechodu z desktopových na vývoj webových aplikací ve frameworku Vaadin 7. Nepůjde pouze o vizuální komponenty. Některé jsou ukryty jen v serverové části a svými mechanizmy napomáhají ke správnému běhu aplikace. Mezi těmi viditelnými budou jednak zmíněny například tabulky, tlačítka, vstupní pole, bez nichž by stránky nemohly existovat, a několik dalších prvků, jež jsou potřeba v každé aplikaci pracující s daty uživatelů, mezi něž lze zařadit především komponenty pro práci s formuláři. Je vhodné mít nástroj, který po předání datového objektu vygeneruje s minimálními změnami stránku s formulářem. V tomto formuláři jsou vytvořena vstupní pole, jimiž uživatel zadává data do systému. Na jednotlivá pole lze umístit validátory zaručující doménovou integritu dat (například že datum nástupu zaměstnance je dříve než datum ukončení pracovního poměru, a ne obráceně).

4.1 Základní vizuální komponenty

V této části budou uvedeny základní jednoduché vizuální komponenty nezbytné pro většinu nových stránek.

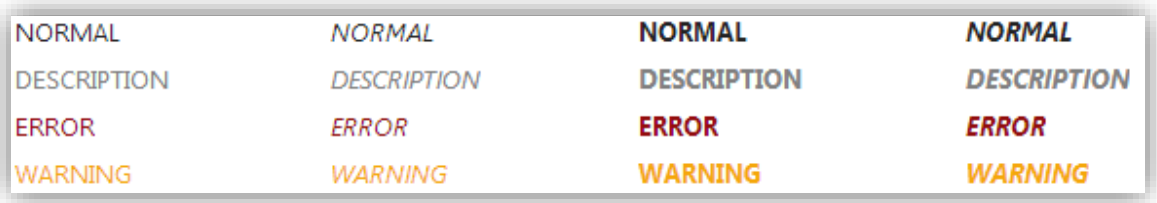
4.1.1 Popisek (Label)

Komponenta Label je ve Vaadin používána pro zobrazení textu. Prvek je do stránky vykreslen pouze pomocí jednoho HTML elementu DIV s definovanou CSS třídou. V základní variantě Label umožňuje pouze nastavení typu obsahu, jakým může být například běžný text, formátovaný text (lze vkládat znaky \n, \t a další) nebo HTML text, díky němuž lze vykreslit vlastní jednoduchý HTML obsah. Pro vývoj je nutné tuto komponentu rozšířit a definovat jí několik standardních stylů pro různé typy popisků.

Rozšíření

K jednoduššímu používání je třída Label rozšířena o metody, jimiž lze nastavit font textu, jako je tučné písmo nebo kurzíva. Další metoda slouží pro nastavení typu popisku. V každé aplikaci je typicky potřeba texty oddělit několika různými barvami, například zašedlý popisek, červená chybová zpráva nebo žluté varování. Všechny tyto vlastnosti se nastavují zadáním hodnoty výčtu do parametru obslužné metody, čímž způsobí přidání příslušné CSS třídy do elementu komponenty, a tedy změnu vzhledu definováním sady stylů.

Dalším rozšířením je přepínač `setEllipsis()`, který zakáže zalamování textu v elementu. Místo toho je popisek oříznut na velikost rodičovského elementu a zakončen třemi tečkami znázorňujícími jeho zkrácení. Na obrázku 8 je zobrazena ukázka několika možných typů popisků.



Obrázek 8: Různé typy popisků

Další zděděné typy

Zmíněná komponenta Label nastavuje pouze barvu a styl textu. Pro změnu velikosti textu jsou vytvořeny další třídy rozšiřující komponentu Label, aby mohly využívat

všechny její metody pro nastavování vzhledu. Vytvoření samostatných tříd pro různé typy popisků zlepšuje čitelnost zdrojového kódu aplikace.

- Header: Komponenta pro vložení textu představující nadpis s větší velikostí. Vzhledem k tomu, že se jedná o rozšíření normálního popisku, je do stránky vložen v podobě elementu DIV a postrádá tedy sémantickou informaci v textu, jakou by měl v případě například element H1.
- SubHeader: Podnadpis má podobně jako komponenta Header vetší velikost textu než má standardní popisek, ale zároveň menší než nadpis.
- SmallLabel: Zobrazuje text menší velikosti.

4.1.2 Popisek s ikonou (IconPropertyLabel)

Jedná se o další modifikaci popisku. Zobrazuje nalevo od textu zvolenou ikonu zarovnanou podle požadavků. Tuto komponentu je možné napojit na datový zdroj Property a jednotlivým možným hodnotám popisku přiřadit ikony, jimž budou automaticky měněny při změně hodnoty.

4.1.3 Tlačítko (Button)

Stejně jako v případě popisku je tlačítko vykreslováno elementem DIV reagujícím na JavaScript událost při jeho stisknutí. Tlačítko může zobrazovat textový popis nebo ikonu.

Rozšíření

Pro tlačítko je opět definováno několik CSS stylů přepínaných podle výčtové hodnoty. Je možné zobrazit větší nebo menší tlačítko, pouhý odkaz vypadající jako běžný A element nebo storno tlačítko pro zrušení nějaké akce.

Dále bylo nutné rozšířit i klientskou část komponenty o drobné úpravy, především změnu reakce při stisku tlačítka. U výchozího tlačítka ve Vaadin se stává, že nereaguje na stisk z důvodu složitého vyhodnocení události, například jestli bylo tlačítko stisknuté při současném pohybu myší. Z tohoto důvodu byla stávající klientská část nahrazena tlačítkem z GWT a lehce poupravena, aby přesněji reagovala na stisk.

4.1.4 Textové pole (TextField)

Textové pole prošlo pouze drobnými úpravami zlepšujícími uživatelskou přívětivost, a to například přidáním tlačítka dovnitř pole, které provede vymazání zadaného textu. Tlačítko je zobrazeno pouze, pokud je zadán nějaký text.

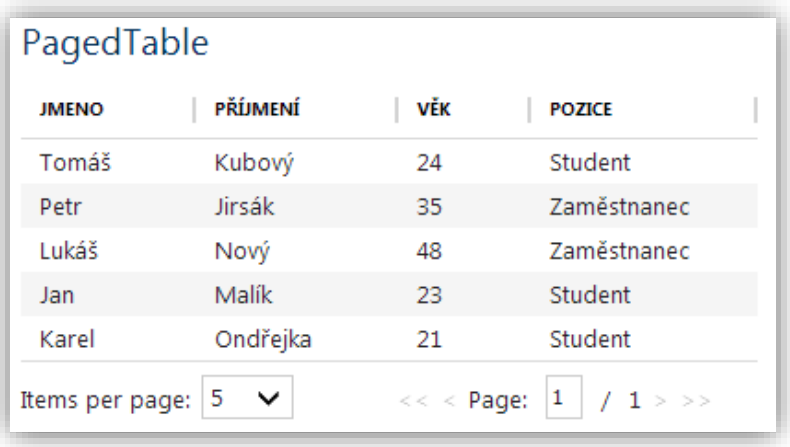
4.1.5 Tabulka (Table)

Tabulka je v základní verzi od Vaadin jako jedna z mála komponent vykreslována pomocí standardních HTML elementů pro tabulky (TABLE, TD, TR) a nikoliv složitou hierarchií elementů DIV. Komponenta slouží k zobrazování tabulkových dat. Standardně jí lze nastavit záhlaví seznamem nadpisů jednotlivých sloupců, anebo vlastní zápatí obsahující například součet hodnot ve sloupci.

Další velmi užitečnou vlastností tabulky je možnost generování vlastních sloupců. Do komponenty jsou předávána data v podobě kontejneru naplněného řádky s jednotlivými hodnotami (jak bylo zmíněno v kapitole 3.5). Pokud je například požadováno navíc u každého řádku zobrazit zaškrtávací pole, stačí vytvořit generátor tohoto sloupce, který na základě zvolených pravidel toto pole vytvoří a případně zaškrtne. Podobným způsobem je možné nastavit vizuální styl jednotlivým buňkám v tabulce. Vaadin nám dovoluje před vykreslením komponenty projít všechny její buňky a nastavit jim vlastní CSS styly, například pokud má být vykreslována různá barva sudých řádků či sloupců nebo červené podbarvení záporných hodnot v buňce. Velmi jednoduše lze také vytvořit tabulku, v níž lze buňky přepnout na editovatelné textové pole a příslušné změny zapisovat do datového objektu například po ztrátě fokusu.

Rozšíření

Pro základní tabulky bylo opět definováno několik pevně daných stylů nastavovaných pouze ve zdrojovém kódu programu. Jedná se například o světlý nebo tmavý odstín záhlaví, příslušných textů atd. Dále je umožněno nastavit zabarvení řádku při najetí kurzorem myši.



Obrázek 9: Rozšířená komponenta tabulky

Jednou z rozšiřujících komponent je `PagedTable` (viz obrázek 9) zděděná z normální tabulky. Tato stránkovací tabulka obsahuje navíc ovládací panel, v němž lze nastavit počet řádků zobrazených v tabulce a přepínat mezi dalšími stránkami.

4.1.6 Výběr prvku z množiny (SingleSelectField)

Jedná se o komponentu pro výběr z množiny hodnot, kdy jsou všechny hodnoty zároveň zobrazeny v řádku. Vybraná hodnota je uložena do zdroje `Property`, aby mohla být komponenta jednoduše použita například ve formulářích.

Seznam je možné naplnit buď postupným přidávání položek do kontejneru, elegantnější řešení je ovšem v konstruktoru třídy předat výčtový typ s definovanými popisky. To zajistí automatické vygenerování všech komponent s hodnotami pro výběr na jednom řádku zdrojového kódu. Na následujícím obrázku 10 je zobrazena ukázka komponenty pro výběr prvku z množiny hodnot.

Obrázek 10: Komponenta SingleSelectField

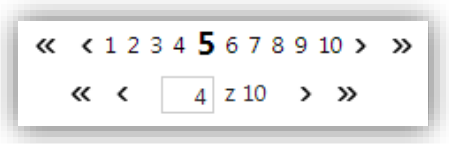
4.1.7 Stránkování (Pager)

Komponenta pro stránkování seznamů zobrazuje aktuálně vybranou stránku a šipky pro přechod na další nebo předchozí stránky. Implementace není nijak závislá na přesném typu objektu, jenž má být stránkován. Komponenta pouze poskytuje rozhraní, na které je odeslána informace s přechodem na vybranou stránku. O zobrazení správných dat se stará

objekt samotný. Díky tomu je možné stejnou komponentu na stránkování použít v celé aplikaci a sjednotit tak jejich vzhled.

Pro vytvoření několika typů komponent je jako předek použita abstraktní třída zajišťující logiku pro přepnutí stránky. Jednotlivé stránkovače musí tuto třídu rozšířit a poskytnout jí tlačítka pro přechody na další či předchozí stránky a prvky zobrazující aktuálně vybranou stránku.

V rámci práce byly implementovány dvě komponenty s rozdílným vzhledem. První z nich zobrazuje řadu čísel představujících možné stránky, v níž je právě vybraná stránka zvýrazněna. Druhá komponenta využívá pro výběr textové pole s aktuálním editovatelným číslem stránky, jak je zobrazeno na obrázku 11.



Obrázek 11: Komponenty pro stránkované seznamy

4.1.8 Kalendář (InlineDateField)

Kalendář je další základní komponentou z balíku Vaadin, která je navíc dostupná ve dvou verzích. První z nich je zobrazena v prohlížeči v podobě textového pole, jenž po stisku zobrazí vyskakovací okno s kalendářem pro výběr data. Druhá verze je bez textového pole, kdy je kalendář stále viditelný ve stránce. Komponenta je složena z elementů tabulky překreslované pomocí JavaScript při přepínaní data.

Jako většina komponent pro vstupní data lze i kalendář napojit na zdroj `Property`, do něhož bude aktualizována vybraná hodnota. Jediná událost detekovatelná na poskytovaném kalendáři je pouze změna výběru data. Další události, na něž by bylo vhodné reagovat po interakci uživatele, jako například reakce při stisknutí čísla týdne nebo popisku aktuálního vybraného měsíce, jsou dostupné až doplněním vlastního rozšíření.

Rozšíření

Kalendář kromě čísel dnů obsahuje také popisek s aktuálně zobrazovaným měsícem, rokem a číslem týdne v řádku. Pro reakci na stisknutí těchto prvků je nutné rozšířit klientskou část komponenty. K tomu je zapotřebí identifikovat jednotlivé HTML elementy tabulky a přidat na ně JavaScript událost reagující na stisk tlačítka myši.

Pro nalezení patřičných buněk, na něž má být událost přidána, je nutné je postupně procházet přes potomky od kořenového elementu (tedy elementu <TABLE>) až po požadované, jelikož jejich instance nelze v kódu získat přímo. Tento krok ulehčí znalost HTML struktury a CSS tříd jednotlivých skupin elementů.

V ukázce zdrojového kódu 2 je zobrazena ukázka kódu metody, která nalezne elementy řádků s čísly týdnů v kalendáři. Jak je vidět, postupně je procházeno přes potomky elementů až k požadovaným řádkům. Každý řádek týdne obsahuje číslo v roce a poté sedm buněk s čísly dnů v tomto týdnu. Právě na první buňku, tedy číslo týdne, je přidána událost reagující na kliknutí myši. Pomocí CSS stylů je nastaven kurzor myši na typ ukazatele pro kliknutí, aby bylo uživateli jasné, že jde o element vyvolávající nějakou akci.

```
public List<Element> getWeekNumElements() {
    Element root = calendarWidget.getElement();
    Element tab1 = root.getElementsByTagName("table").getItem(0);  //element tabulky
    Element tbody = tab1.getElementsByTagName("tbody").getItem(0); //element těla
                                                                    tabulky
    Element tr1 = tbody.getElementsByTagName("tr").getItem(1); // řádek s vnitřní
                                                               tabulkou s čísly dnů
    Node td1 = tr1.getFirstChild();   // buňky ve které je vnitřní tabulka

    Element table2 = Element.as(td1.getFirstChild()); // vnitřní tabulka s čísly dnů
    Element tbody2 = table2.getElementsByTagName("tbody").getItem(0); //tělo vnitřní
                                                                       tabulky
    NodeList allRows = tbody2.getElementsByTagName("tr"); //seznam řádků (týdnů v
                                                          měsíci)

    List<Element> weekNumRows = new ArrayList<Element>(); //seznam všech týdnů
    for (int i = 1; i < allRows.getLength(); i++) {
        weekNumRows.add(Element.as(allRows.getItem(i)));
    }

    return weekNumRows;
}
```

Zdrojový kód 2: Metoda pro získání elementů s řádky týdnů

Další metoda uvedená v ukázce zdrojového kódu 3 provede registraci obsluhy události na element předaný v parametru metody. Jak je z kódu patrné, klíčové slovo `native` překladači napoví, že se nejedná o kód jazyku Java, a tudíž že nemá být překládán do `bytecode`. Právě touto konstrukcí je možné v běžném zdrojovém souboru Javy použít syntaxi JavaScriptu, a tím přidat na element reakci na událost `onclick`. V obsluze události je vyvolána metoda `weekNumClicked(Element el)` napsaná v jazyce Java, která odesílá informaci o stisknutí čísla týdne na serverovou část komponenty, kde může být obsloužena zaregistrovaným posluchačem.

```
public native void registerWeekNumClick(Element el)
/*-{
    var self = this;
    el.onclick = $entry(function() {
        self.@cz.marbes.vaadin.components.inlinedatefield.extension.dateclickevent.
                client.DateFieldConnector::
                weekNumClicked(Lcom/google/gwt/dom/client/Element;)(el)
    });
}-*/;
```

Zdrojový kód 3: Metoda pro přidání události na element

Podobným způsobem jsou vytvořeny i obsluhy pro kliknutí na popisek měsíce a roku. Jelikož je tento popisek společný, jak je vidět na následujícím obrázku, je nutné jej navíc rozdělit na dva elementy, jeden pro měsíc a jeden pro rok, aby bylo možné reagovat na kliknutí myši na každého z nich zvlášť.

I když by se dala celá klientská část komponenty napsat v jazyce JavaScript, je vhodnější použít kód Java a do něho nezbytné JavaScript části vkládat. Výsledkem po kompilaci je sice pouze čistý JavaScript kód přeložený překladačem GWT, ovšem již rovnou v několika verzích pro různé internetové prohlížeče. Navíc je serverová část napsána také v Java a pro komunikaci pomocí sdílených stavů nebo volání vzdálených metod RPC (Remote procedure call – vzdálené volání procedur) je tento postup vývoje přehlednější a rychlejší.

Vzhled komponenty kalendáře je ilustrován na obrázku 12.

Obrázek 12: Komponenta kalendáře

4.1.9 Stažení souborů (Upload)

Komponenta Upload slouží k nahrávání souborů uživatelem z klientské stanice na server. Vaadin nám nabízí verzi, jež neumožňuje nic jiného než vybrání a nahrání souboru. Nelze filtrovat typy souborů v dialogovém okně pro výběr, nahrávat více souborů najednou ani zobrazovat aktuální stav již odeslaných dat. Pro moderní aplikace vyžadující uživatelskou přívětivost, například možnost přetažení souboru z plochy operačního systému do prohlížeče, je nutné komponentu podstatně přepracovat, aby zmíněné funkce umožňovala a zároveň byla dostatečně jednoduše použitelná pro programátory.

Rozšíření

Pro celou komponentu je nutné přepracovat téměř od počátku jak klientskou, tak serverovou část, aby fungovaly podle požadavků. Nejprve je nově vytvořena klientská část obsluhující dialogové okno pro výběr souborů k nahrání. Tento prvek obsahuje pouze tlačítko pro zobrazení dialogového okna, všechny ostatní prvky, jako jsou ukazatele stavu nahrávání nebo tlačítko pro smazání nahraného souboru, jsou vytvářeny na serverové straně.

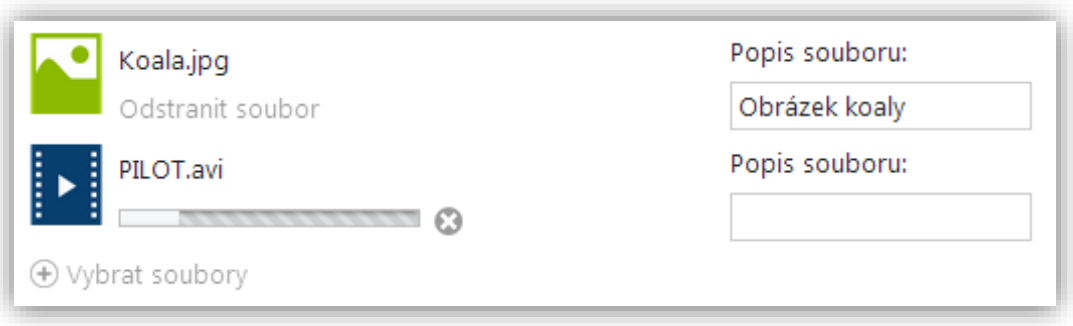
Po výběru jsou soubory k uložení odeslány v podobě proudových proměnných na serverovou část, kde dále probíhá jejich zpracování. Pro každý nahrávaný soubor je vytvořen ukazatel stavu zobrazující, kolik dat z celku je již přijato. Tento stav je pro každý soubor aktualizován během nahrávání. Po dokončení jsou stažené soubory zapsány do zdrojové proměnné a proběhne informování posluchačů o ukončení procesu. Nahrávané soubory jsou ukládány do dočasného adresáře, který je po ukončení aplikačního serveru smazán, aby zbytečně nezabíral místo. Pro uchování je nezbytné jejich přesunutí podle požadavků.

Přetažení souboru do prohlížeče

Další možností pro nahrání souboru je pomocí přetažení z prostředí operačního systému na klientském počítači do komponenty v prohlížeči, jež je určena jako přijímající. Průběh procesu je podobný jako v předchozím případě, kdy jsou pro nahrávané soubory vytvořeny komponenty s ukazatelem stažených dat a po dokončení s tlačítkem pro odebrání souboru ze seznamu.

V implementaci komponenty pro stahování je vytvořena i standardní komponenta zobrazjící informace o stahovaném souboru. Lze ovšem vytvořit i vlastní nezávislý řádek

se souborem, který se sám stará o aktualizaci stavu stahování a o změnu po dokončení stažení, pokud je tato změna požadována v aplikaci.



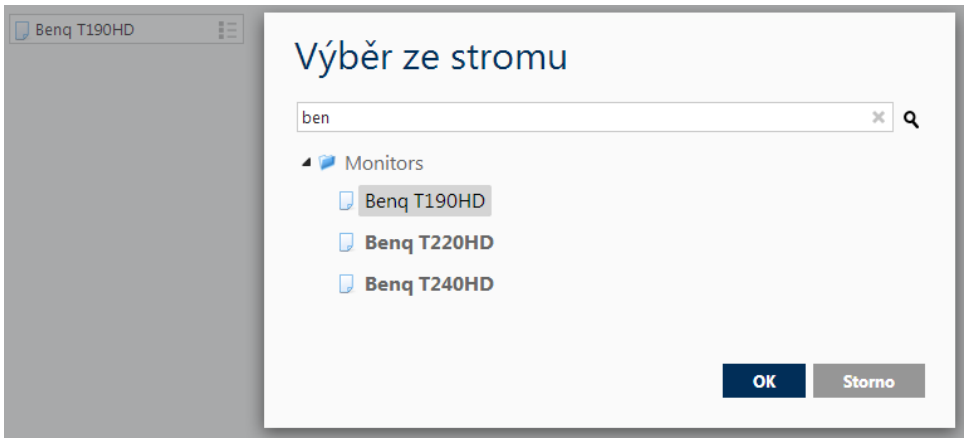
Obrázek 13: Komponenta pro nahrávání souborů na server

Na obrázku 13 je zobrazena ukázka komponenty pro nahrávání souborů. Jak je patrné, jeden soubor již byl úspěšně přijat, jelikož má uživatel možnost jej odstranit, zatímco druhý soubor je stále ještě nahráván na server. Ke každému souboru je možné přidat vlastní popisek a pracovat s ním tak, jak vývojář stránek definuje v aplikaci, jelikož se jedná o vlastní implementaci řádku, jak bylo zmíněno v předchozím odstavci. Standardní implementace obsahuje pouze část zobrazenou v levé části uvedeného obrázku.

4.1.10 Výběr položky ze stromu (TreeSelect)

Komponenta zobrazená na obrázku 14 slouží pro výběr prvků ze stromové struktury. Jedná se o rozšíření rozevíracího seznamu o možnost vložit do zdrojových dat stromovou strukturu, a dále o tlačítko zobrazující po stisknutí dialogové okno s vyhledávacím polem a celým stromem s daty.

Při psaní do textového pole je automaticky rozevřen seznam s položkami odpovídajícími napsanému textu, které je možné po stisknutí vybrat. Pokud je položek větší množství a výběr mezi nimi je méně přehledný, pomocí tlačítka je možné zobrazit dialogové okno. Toto okno obsahuje stejná data jako rozevírací seznam a položky lze opět filtrovat podle zvoleného textu. Vybraná položka v dialogovém okně je s rozevíracím seznamem propojena přes zdrojovou proměnnou, čímž odpadá starost o aktualizaci při otevírání nebo zavírání okna, jelikož je vše prováděno automaticky.



Obrázek 14: Komponenta pro výběr položek ze stromové struktury

4.1.11 Záhlaví a zápatí

Pokud spolu nějakým způsobem souvisí více samostatných aplikací (např. tvoří společně jedno řešení), je pro uživatele dobré, aby měly jednotný vzhled a funkčnost základních prvků a lišily se pouze například barvou a názvem. V tom případě je nutné vytvořit komponentu pro záhlaví a zápatí stránek. Jedná se o prvky viditelné na všech stránkách, jež mají stále stejnou pozici a funkčnost základních prvků umístěných na komponentě. Například při stisku loga v záhlaví proběhne přesměrování na úvodní stránku, nebo že se v pravém horním rohu nachází ikona pro odhlášení z aplikace.

Záhlaví

Komponenta pro záhlaví je zobrazována jako úzký pruh v horní části každé stránky. Tato verze záhlaví nemá možnost zobrazovat složité struktury s komplexními komponentami, jak tomu bývá např. v kancelářských aplikacích od firmy Microsoft. Obsahem jsou vždy pouze jednořádkové prvky, jako jsou popisky, tlačítka nebo rozevírací menu.

Řádek záhlaví je rozdělen na tři části. V levé části je zobrazeno logo aplikace fungující také jako odkaz na hlavní stránku. Vedle něj se nachází prostor pro navigační prvky, jako je například rozbalovací menu s odkazy na přechod do jiné části aplikace. Uprostřed záhlaví je umístěn kontejner pro ovládací prvky aktuálně zobrazované stránky. Každá stránka si může definovat vlastní tlačítka sloužící k jejímu základnímu ovládání nebo k vyvolání typické akce, jakou může být například tisk nebo editace uložených hodnot. Poslední, třetí část záhlaví, slouží ke konfiguraci aplikace. Jsou zde prvky jednak pro uživatelské nastavení, jako je změna dat vlastního účtu nebo tlačítko pro přihlášení

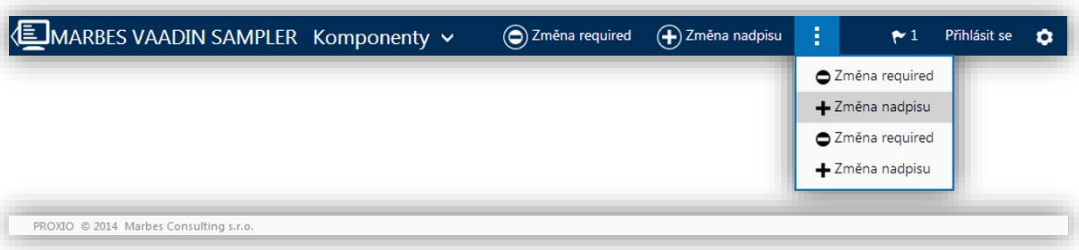
či odhlášení uživatele. Dále tato část obsahuje notifikační ikonu zobrazující uživateli informace o nových událostech (zprávách), a poté menu pro nastavení aplikace.

Jednotlivé části záhlaví jsou tvořeny základní Vaadin komponentou `MenuBar`, jež slouží jako kontejner zobrazující rozbalovací menu, pod–menu a samotné položky. Přímo k těmto vlastním instancím komponenty `MenuBar` nemá programátor přístup, aby neměl možnost, ať již úmyslně nebo omylem, měnit základní vhled záhlaví. Pro všechny potřebné akce, jako je přidání či odebrání položek z menu nebo nastavení popisku aplikace, jsou implementovány metody, pomocí nichž má vývojář možnost hodnoty menu nastavit. Záhlaví slouží pouze pro základní, neměnné funkce, které by měly být z větší části jednoduché a fixní, aby nepůsobily pro uživatele zmatečným dojmem. Pokud je nutné vytvořit složitější obsluhu konkrétní stránky, měla by být umístěna uvnitř.

Zápatí

Podobně jako záhlaví je i zápatí zobrazeno na všech stránkách aplikace jako řádek ve spodní části stránky. Je podstatně jednodušší, neobsahuje žádné ovládací prvky a slouží pouze k zobrazení informací o autorovi aplikace, jeho kontaktu a dále například data vydání a označení verze aplikace. V současnosti mají takto navržené aplikace záhlaví i zápatí stále viditelné, a pokud obsah stránky přesahuje velikost mezi nimi, je posouván pouze uvnitř vymezujícího prostoru.

Obě komponenty, jak záhlaví, tak zápatí jsou znázorněny na obrázku 15.



Obrázek 15 Záhlaví a zápatí aplikace

4.2 Formuláře

Silnou stránkou Vaadin jsou jednoznačně formuláře. Základem funkčnosti je data-binding (uveden v kapitole 3.5), dávající formulářům implementační jednoduchost a zároveň sílu pro řešení různorodého množství situací. Proměnným hodnotám z datového objektu jsou vytvořena příslušná editační pole a dojde k jejich vzájemnému provázání.

Při editaci jsou následně zadané hodnoty podrobeny nastavené validaci a popřípadě trvale uloženy do datového objektu.

Použití základní implementace mechanizmu formulářů od autorů Vaadin je pro často opakované a obdobné situace velmi zdlouhavé. Nachází se zde několik částí, které lze vhodným způsobem upravit s výsledkem zjednodušení, a tedy zrychlení vývoje stránek s formuláři.

4.2.1 Základní implementace

Základní implementaci tvoří layout formuláře, do něhož jsou vkládána editační pole. Jednotlivá pole jsou propojena pomocí třídy `FieldGroup` starající se o zapsání skupiny editovaných hodnot najednou a zadání validních dat.

Ukázka a popis základní implementace

Nejprve je nutné vytvořit datový objekt nesoucí v sobě seznam proměnných hodnot, do nichž budou zapisována data získaná z formuláře. Ukázka jednoduchého datového objektu je zobrazena v následující ukázce zdrojového kódu 4.

```
PropertysetItem item = new PropertysetItem();
item.addItemProperty("jmeno", new ObjectProperty<String>("Tomáš"));
item.addItemProperty("vek", new ObjectProperty<Integer>(24));
```

Zdrojový kód 4: Vytvoření objektu Item s Property

Poté je definován layout spolu s editovanými poli, která budou svázána s jednotlivými `Property` výše vytvořené položky `Item`. Jedná se tedy o `Property` `jmeno` a `vek`.

Třída FieldGroup

Tato třída se stará o provázání komponent (vstupních polí formuláře) s hodnotami datových objektů (`Property`). V podstatě jde o automatickou aktualizaci dat při změně hodnoty v poli. Bez této třídy by bylo nutné se pro každou aktualizaci dat dotazovat všech editačních polí, jakou mají hodnotu, aby mohla být zapsána do proměnné. Tento postup by byl ve většině případů na programátorovi.

```
// layout, do kterého jsou umístěny pole
FormLayout form = new FormLayout();

// následně jsou do formuláře pole přidána pomocí prvku binder, který provede
// vytvoření polí a jejich svázání s property
FieldGroup binder = new FieldGroup(item);
form.addComponent(binder.buildAndBind("Jméno", "jmeno"));
form.addComponent(binder.buildAndBind("Věk", "vek"));
```

Zdrojový kód 5: Jednoduchá komponenta se standardním formulářem

Jak je vidět v uvedeném příkladu zdrojového kódu 5, třída `FieldGroup` umožňuje také samotné vytváření vizuálních komponent. Pomocí metody `buildAndBind()` proběhne vytvoření instance komponenty podle datového typu zadané `Property`, tedy pro `jmeno` bude zhotoveno textové pole přijímající jakýkoliv text. Pro zdrojovou položku `vek` bude vytvořeno také textové pole, ovšem jeho vstupní hodnota může být pouze číslo. Pokud by číslo zadané nebylo, formulář se neuloží a zobrazí chybové hlášení o nepsrávném formátu dat. Metodě `buildAndBind()` je předáván jako parametr popisek textového pole zobrazený vedle komponenty a jméno `Property`, kam budou zapsána zadaná data.

Třída FieldFactory

Jde o třídu vytvářející vizuální komponentu vstupního pole podle datového typu. Tato třída je nastavována ve třídě `FieldGroup`. Pokud je zaregistrována vlastní implementace, je možné například nastavit, že ke vstupnímu poli `věk` bude vytvořena komponenta posuvníku, v němž lze vybrat pouze hodnotu od 1 do 199. Tím je zároveň zajištěno, že uživatel do položky věk nebude moci zapsat jinou než kladnou celočíselnou hodnotu.

S těmito prvky je možné vytvářet stránky s jednoduchými formuláři. Práce s nimi sice není složitá na porozumění, ale zdlouhavá u rozsáhlejších internetových aplikací, kde je potřeba velké množství formulářů majících ve většině případů stejné nebo podobné rozložení vstupních polí.

4.2.2 Rozšířená implementace

Cílem rozšíření tohoto mechanizmu je opět jednak usnadnění práce programátorům, a zároveň stanovit základ jednotného vzhledu formulářů.

Datový objekt pro formulář

Zdrojem dat pro formulář je vždy nějaký datový objekt obsahující jednoduché proměnné požadovaných datových typů. Například pro datový objekt `Osoba` budou proměnnými jméno, přímení, věk, email atd. Na tyto proměnné lze nastavit několik anotací určujících jejich vlastnosti sloužících pro generování komponent vstupních polí formuláře. Jde především o anotace popisující vzhled a funkčnost polí nebo určující formát vstupních dat pro úspěšnou validaci. Patří mezi ně například popisek vstupního pole zobrazovaný vedle komponenty, přepínač, zda je pole povinné pro úspěšnou validaci nebo nastavení maximální délky zadaného řetězce. Textovým datům lze nastavit předpis v podobě regulárního výrazu nebo anotaci `email` definující, že vstupní řetězec musí mít běžný tvar emailu. Dále, pokud se jedná například o datový typ pro datum, lze natavit povinný formát vstupu a přesnost požadovaného časového údaje (nastavení nejmenší časové jednotky – např. minuty).

Třída GridBeanFieldValues

Třída je hlavní částí rozšíření formulářů spojující všechny potřebné prvky dohromady. V první řadě se jedná o vizuální komponentu, v níž jsou pole automaticky rozmísťována do jednoho sloupce pod sebe, kdy v levé části sloupce jsou umístěny popisky vstupních polí a v pravé části samotná pole. Tuto komponentu lze poté vložit na požadované místo, ať již na stránku nebo do dialogového okna. Pro maximální využití funkcí je lepší vložit formulář do komponenty `Form`, která navíc přidává místo pro chybové hlášení vzniklé validací formuláře a lištu pro umístění ovládacích tlačítek.

Pokud je potřeba standardně vytvořené vstupní pole rozšířit, například za rozbalovací seznam přidat tlačítko vkládající do položek seznamu nový objekt, je nutné nastavit vlastní rozhraní vyrábějící vizuální komponenty. Rozhraní se jmenuje `GridBeanFieldFactory` a obsahuje pouze metodu vracející vytvořenou komponentu. V této metodě, která má jako jeden z parametrů název vstupního pole, je identifikován rozbalovací seznam a následně umístěn do horizontálního layoutu spolu s tlačítkem přidávající prvek do seznamu. Layout je poté vrácen a vykreslen do formuláře místo samotného rozevíracího seznamu.

Další funkcí formuláře je vytváření vstupního pole v závislosti na typu proměnných hodnot v datovém objektu. K tomu slouží třída `BeanFieldGroup` popsaná v následující části.

Třída BeanFieldGroup

Tato třída reprezentuje seznam polí obsažených ve formuláři, nad nimiž lze provádět standardní úkony, jako jsou uložení hodnoty nebo validace. Jedná se o potomka stejnojmenné třídy ze základu frameworku Vaadin rozšířeným navíc o možnosti validace celých datových objektů společně, zatímco základní implementace umožňuje definování pouze validátorů na jednotlivá vstupní pole. Jako ukázku si lze představit situaci, kdy je výběr jedné hodnoty závislý na jiné vybrané hodnotě. Požadavek, že musejí být zadané obě hodnoty, v tomto případě nezajistí jejich logickou souvislost.

Pro vytváření vstupních polí je nutné definovat do této třídy tovární třídu rozšiřující rozhraní `BeanFieldGroupFieldFactory`. Standardně je použita implementace vytvářející běžná vstupní pole, a to `SimpleFieldGroupFieldFactory`. Postupně jsou procházeny všechny proměnné datového objektu, pro nějž je sestavován formulář. U každé proměnné je nalezen seznam anotací zpřesňujících vlastnosti, jak je uvedeno výše, a následně je příslušné vstupní pole vytvořeno a provázáno s proměnnou.

Ukázka jednoduchého formuláře

V následující části bude uvedena ukázka vytvoření jednoduchého formuláře pro registraci osoby.

Datový objekt uvedený v ukázce zdrojového kódu 6 obsahuje čtyři proměnné, pro které budou vytvořena vstupní pole formuláře. Formát dat je uveden v anotacích umístěných nad příslušnými proměnnými hodnotami. Například hodnota data narození musí být ve tvaru určeném v anotaci `DateAttrDesc`. Datový objekt musí navíc obsahovat typu `get` a `set` pro používané proměnné, jež nejsou v příkladu z důvodu úspory místa uvedeny.

```
public class Person {
    @AttrDesc(caption = "Jméno", required = true, length = 100)
    private String firstName;

    @AttrDesc(caption = "Příjmení", required = true, length = 100)
    private String lastName;

    @AttrDesc(caption = "Datum narození", required = true)
    @DateAttrDesc(format = "dd.MM.yyyy HH:mm")
    private Date birth;

    @Email
    @AttrDesc(caption = "Email", length = 100)
    private String email;
 }
```

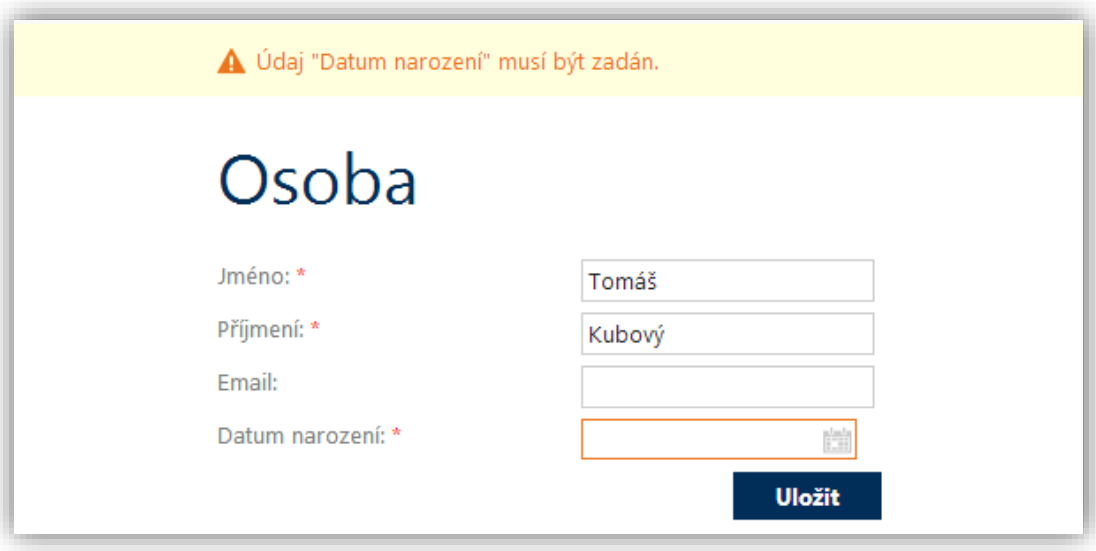
Zdrojový kód 6: Vytvoření komponenty formuláře

Následuje nastavení komponenty s formulářem v ukázce zdrojového kódu 7. Nejprve je vytvořen datový objekt osoby (může být například získaný z databáze, aby obsahoval předvyplněná data ve formuláři). Tento objekt je poté vložen do obalovací třídy `BeanItem`, která proměnným `jméno`, `příjmení` atd. vytvoří jednotlivé zdroje `Properties`. Třída `GridBeanFieldValues` provede vygenerování všech vstupních polí za pomoci zmíněné `BeanFieldGroup`. Pole budou postupně vykreslována v pořadí uvedeném v seznamu `fieldsOrder`. Nakonec je tabulka vložena do komponenty formulář, kde je k ní přiřazeno tlačítko pro uložení dat.

```
Person person = new Person();
person.setFirstName("Tomáš");
person.setLastName("Kubový");

BeanItem<Person> personItem = new BeanItem<Person>(person);

GridBeanFieldValues<Person> gridValues = new GridBeanFieldValues<->(personItem);
gridValues.setFirstColumnWidth(200, Unit.PIXELS);
List<String> fieldsOrder = Arrays.asList("firstName","lastName","email","birth");
gridValues.showAllFields(fieldsOrder);

Form<Person> form = new Form<Person>("Osoba", gridValues);
form.addConfirmButton("Uložit");
form.setSizeFull();
```

Zdrojový kód 7: Práce s datovým objektem formuláře

Na obrázku 16 je uvedena ukázka jednoduchého formuláře vytvořeného pomocí rozšířeného mechanizmu pro tvorbu formulářů. Snímek je zachycen po stisku tlačítka pro uložení dat. Vzhledem k nevyplněnému poli data narození je v horní části formuláře zobrazena chybová hláška informující o tom, že není vyplněno povinné pole.



Obrázek 16: Jednoduchý formulář pro registraci osoby

Na zmíněném příkladu je patrná jednoduchá práce s formulářem, kdy prakticky stačí vyvolat metodu pro vygenerování datových polí a výslednou komponentu umístit do stránky. Pro mnoho situací je tento postup dostačující a nepotřebuje větších zásahů do vzhledu formuláře. Pokud je změna nutná, je zde možnost nastavit vlastní pravidla pro generování polí a jejich rozmístění. V posledním případě lze obejít předdefinované rozložení polí a jednotlivé prvky umisťovat do vlastního formuláře podle vlastních požadavků.

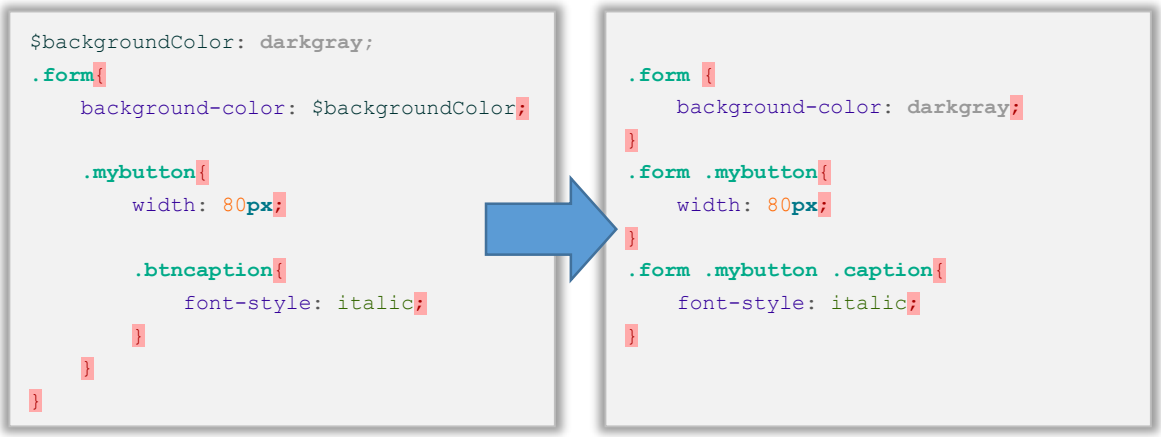
V případě datového objektu je sice nezbytné, aby byl tento kód napsán speciálně pro příslušný formulář, ovšem objekt může být bez problému použit i v dalších částech aplikace bez nutnosti změny. Anotace jsou pouze doplňující částí formuláře a neovlivňují běžnou práci s proměnnými hodnotami.

4.3 Vzhled komponent

Pro nastavení vzhledu komponent ve Vaadin je využívána technologie SASS (Syntactically Awesome Stylesheets). Jedná se o rozšíření kaskádových stylů CSS3 přidávající do šablony vnořená pravidla, definovatelné proměnné hodnoty, dědičnost

a několik dalších vlastností. Tento způsob redukuje počet nutných řádků pro nastavení vzhledu komponenty, čímž zároveň podstatně zpřehledňuje čitelnost souboru se styly.

Definované stylové třídy jsou do sebe zanořované stejným způsobem jako HTML elementy popisující tyto třídy.



```
$backgroundColor: darkgray;
.form{
    background-color: $backgroundColor;

    .mybutton{
        width: 80px;

        .btncaption{
            font-style: italic;
        }
    }
}
```

Zdrojový kód 8: Styly v SASS

```
.form {
    background-color: darkgray;
}
.form .mybutton{
    width: 80px;
}
.form .mybutton .caption{
    font-style: italic;
}
```

Zdrojový kód 9: Styly v CSS

Na předchozích ukázkách je zobrazen překlad jednoduchého souboru stylů z formátu SASS (ukázka zdrojového kódu 8) do formátu CSS (ukázka zdrojového kódu 9). Je vhodné si například na vrcholu hierarchie SASS souborů se styly definovat základní společné konstanty, jakými mohou být různé barvy určující vzhled aplikace nebo standardní šířky tlačítek a podobně. Díky tomu je jednoduché v rozsáhlé aplikaci tyto konstanty změnit pouze na jednom místě, pokud je potřeba vyřešit požadavek na globální změnu vzhledu. Každá komponenta může obsahovat prvky podléhající například barvě aplikace. Jelikož jsou komponenty vytvářeny abstraktně, aby jejich použití nebylo závislé pouze na jedné implementaci, je vhodné definovat takové konstanty, podle nichž se komponenta přizpůsobí vzhledu aplikaci. Pokud je na začátku aplikace nastaveno, že základní barva je modrá, všem ovlivněným komponentám je tato změna přepsána a není nutné nikde jinde barvu měnit.

Překlad do CSS stylů, které jsou odesílány na klientský prohlížeč, je prováděn při kompilaci Vaadin aplikace. Ve vývojovém módu lze překládat styly za provozu, tedy při každém požadavku klienta o sadu stylů, aby nebylo nutné při každé změně znovu překládat aplikaci.

4.4 Implementace komponent

Vaadin nabízí pro vývoj nových komponent tři různé způsoby. Nejjednodušším způsobem je rozšíření pouze serverové části komponenty, ať již přidáním funkčnosti nebo spojením několika menších prvků. Další možností je rozšířit stávající soustavu serverové a klientské části komponenty o vlastní funkce, jež budou přeloženy do jazyka JavaScript a budou použity spolu s klientskou částí komponenty. Nejobtížnějším způsobem vývoje je vytvoření celé vlastní komponenty, počínaje serverovou částí, komunikačním kanálem až po klientskou část obsahující jak funkční prvky, tak HTML elementy.

4.4.1 Rozšíření serverové části komponenty

Způsob vytvoření nové komponenty pouze na serverové části je z uvedených možností nejjednodušší. Není nutné překládat žádný kód do jazyka JavaScript, což urychluje vývoj, a navíc umožňuje jednoduchý způsob ladění napsaného kódu přímo ve vývojovém prostředí, na rozdíl od ostatních postupů.

Některým aplikacím postačuje funkčnost klientských částí komponent a není nutné je rozšiřovat. Ve většině případů se pak nová komponenta skládá z několika potomků vhodně propojených a zastřešených vlastní logikou. Pokud se jedná pouze o jednoduché úpravy nad stávající komponentou, stačí vytvořit potomka od základní třídy tohoto prvku a přidat vlastní funkce. Takovým příkladem může být rozšíření popisku (Label viz oddíl 4.1.1) o seznam výchozích stylů, které lze jednoduše programově nastavit metodou. Pro rozsáhlejší změny je vhodnější dědit od třídy `CustomComponent`.

CustomComponent

Tato třída reprezentuje předka rozšířených komponent. Obsahuje všechny potřebné metody k tomu, aby ji bylo možné zobrazit v prohlížeči. Lze jí nastavit šířku, výšku a název třídy stylu, jenž bude použit na vrcholový HTML element komponenty. Obsah se přiřadí metodou `setCompositionRoot`() mající jako parametr komponentu představující kořen celku (většinou layout naplněný jednotlivými prvky).

Pokud má být vytvořená komponenta použitelná ve formulářích, musí obsahovat datový zdroj sloužící pro uchovávání zadané hodnoty. V tomto případě je předek rozšíření abstraktní třída `CustomField`. Ta se stará o logiku spojenou s komunikací s datovým zdrojem. Nastavení obsahu je podobné jako v případě `CustomComponent` s tím

rozdílem, že je nutné implementovat abstraktní metodu `initContent()` vracející kořenovou komponentu.

```
    private void init() {
        Image icon = new Image(null, ProxioThemeResources.ADD_ITEM_BLACK_16);
        Label caption = new Label("Přidat osobu");

        HorizontalLayout rootLayout = new HorizontalLayout();
        rootLayout.addComponent(icon);
        rootLayout.addComponent(caption);
        rootLayout.setExpandRatio(caption, 1.0f);

        setCompositionRoot(rootLayout);
    }
```

Zdrojový kód 10: Ukázka jednoduché komponenty IconLabel

V ukázce zdrojového kódu 10 je zobrazeno, jak je řešeno vytvoření komponenty popisku s ikonou, kdy je v levé části zobrazena ikona pro přidání osoby a v pravé části popisek. Tyto prvky jsou umístěny do společného horizontálního layoutu tvořící kořenovou část komponenty.

4.4.2 Rozšíření klientské části komponent

Jde o použití nezměněné komponenty na stránce. Pro tuto komponentu je navíc vytvořeno rozšíření klientské části, které na ni může i nemusí být aplikováno. Samotná komponenta musí pracovat i bez rozšíření. Jde tedy pouze o doplnění funkčnosti na klientské straně překládané do jazyka JavaScript.

Rozšíření se skládá především ze dvou částí, a to ze třídy `AbstractExtension` rozšiřující serverovou část a konektoru `AbstractExtensionConnector` v části klientské.

AbstractExtension

Třída zděděná z třídy od `AbstractExtension` obsahuje metodu `extend()`, jež jako parametr přijímá komponentu, na níž bude rozšíření aplikováno, a dále umožňuje komunikaci se sdíleným stavem (viz 4.4.4).

AbstractExtensionConnector

Konektor slouží ke spojení třídy `AbstractExtension` s klientskou částí komponenty. Spojení se zajistí pomocí anotace na třídě konektoru, ve které je odkazováno

na potomka třídy rozšiřující `AbstractExtension`. Samotné propojení s instancí rozšiřované komponenty je řešeno přes metodu `extend`() zmíněnou výše. Po propojení je možné v konektoru získat přístup ke klientskému widgetu[4], díku čemuž lze editovat HTML elementy komponenty, vytvářet nové elementy nebo přidávat posluchače v jazyce JavaScript obsluhující události vyvolané uživatelem.

4.4.3 Vytvoření vlastní komponenty

Vytvoření vlastní komponenty je podobné oběma předešlým možnostem s několika rozdíly. Serverovou část komponenty tvoří potomek třídy `AbstractComponent` (rodič třídy `CustomComponent` zmíněné v oddílu 4.4.1), s níž se nadále pracuje jako s každou jinou komponentou, například umístěním do layoutu.

Dále je nutné vytvořit vlastní widget, jenž bude vykreslen do internetového prohlížeče. Vaadin nám dává možnost vytvořit jakoukoliv vlastní HTML strukturu. Tento widget je propojen se serverovou částí opět pomocí konektoru. V případě vlastní komponenty se jedná o potomka konektoru `AbstractComponentConnector`, ve kterém je nutné překrýt metody pro vytvoření a získání widgetu tak, aby vracel vlastní implementaci. Samotný widget je vytvořen jádrovými funkcemi GWT nástroje, jak je uvedeno v ukázce zdrojového kódu 11.

```
@Override
protected Widget createWidget() {
    return GWT.create(MyNewWidget.class);
}
```

Zdrojový kód 11: Ukázka vytvoření instance vlastního widgetu

Posledním krokem je propojení se serverovou komponentou anotací `@Connect` nad třídou konektoru, jež je na rozdíl od pouhého rozšíření komponenty (viz oddíl 4.4.2) napojena přímo na serverovou část, a ne na třídu rozšíření. Komunikace je zajištěna způsobem uvedeným v následující kapitole.

4.4.4 Komunikace serverové a klientské části

Komunikace mezi serverovou a klientskou částí probíhá výměnou zpráv v JSON formátu. Vaadin pro komunikaci nabízí především dva mechanizmy. Jeden z nich

[4] Widget – Klientská část komponenty překládaná z Java do JavaScript, která obsahuje HTML elementy vykreslené do stránky prohlížečem.

využívá sdíleného stavu, druhý pak vzdálené volání procedur (RPC – Remote Procedure Call).

Sdílený stav – SharedState

Jde o třídu obsahující společné proměnné hodnoty sloužící pro komunikaci především ze serverové strany na klientskou. Při změně hodnoty proměnné dojde k odeslání JSON zprávy na klientský konektor, jenž zachytí událost změny stavu a provede příslušnou akci. Vaadin definuje několik potomků třídy SharedState obsahující základní proměnné pro komponenty, jako jsou například šířka, výška atd., aby bylo možné používat základní funkčnost i u vlastních komponent bez nutnosti implementace. Vývojáři stačí rozšířit danou třídu a přidat vlastní proměnné hodnoty přenášející data do klientské části.

Tento typ komunikace je použit především u zpráv uchovávajících stav komponenty, jako jsou šířka, výška nebo například aktuální popisek na tlačítku. Klientská část může jednak okamžitě reagovat na změnu stavu, zároveň umožní dané hodnoty opakovaně číst bez vyvolání změny a bez zbytečné komunikace mezi stranami.

Vzdálené volání procedur – RPC

Vzdálené volání procedur je používáno pro předávání informací o vzniku událostí bez stavu, jakými může být například stisk tlačítka nebo jiná uživatelská interakce, na rozdíl od sdíleného stavu.

Pro implementaci této komunikace je zapotřebí definovat rozhraní možných volaných metod spolu s návratovými hodnotami a parametry. Nové rozhraní rozšiřuje třídu `ClientRpc` nebo `ServerRpc` podle toho, jestli bude probíhat komunikace ze serveru na klienta nebo opačně. Odesílající poté získá instanci rozhraní metodou `getRpcProxy()` dostupnou v běžné vizuální komponentě, jíž se jako parametr předá typ požadovaného komunikačního rozhraní. Poté lze vyvolat nad získanou instancí definovanou metodu. Na druhé straně musí příjemce metodou `registerRpc()` s parametrem typu komunikačního rozhraní zajistit reakci na přijatou zprávu od odesílatele.

V obou případech komunikace, tedy jak u sdíleného stavu, tak u RPC, jsou data transformována do JSON formátu automaticky. Přenášet lze všechny primitivní datové

typy, kolekce, map (ovšem klíčem musí být řetězec) nebo jednoduché serializovatelné objekty.

4.5 Knihovna komponent

Všechny vyvíjené komponenty, které jsou společné pro více aplikací, jsou umístěny v jedné samostatné knihovně. Tuto knihovnu mohou aplikace použít přidáním závislosti, čímž získají přístup jak ke všem komponentám, tak i třídám nezbytným pro zobrazení obsahu ve Vaadin.

Z důvodu překladu a vydávání různých verzí je knihovna vedena jako Maven projekt. Vaadin podporuje vývoj pomocí Maven. Všechny nezbytné knihovny pro vývoj aplikací, jako jsou klientské, serverové a jádrové části, jsou jednoduše dostupné přidáním závislosti na knihovnu do souboru popisující projekt (pom.xml soubor).

Obsah knihovny

Tato knihovna obsahuje všechny zdrojové kódy pro běh Vaadin aplikace. Jde především o komponenty tvořící vzhled aplikace. Dále je zde zahrnuta třída VaadinServlet pro obsluhu HTTP požadavků týkajících se frameworku Vaadin. Pro vzhled jsou zde umístěny všechny CSS a SASS soubory obsahující styly vyvinutých komponent. Poslední částí knihovny jsou soubory pro přeložené texty použité v komponentách, díky nimž lze jednoduše přepínat jazyk uživatelského rozhraní aplikace.

Překlad projektu

O překlad zdrojových textů se stará Framework Maven spolu se standardními pluginy od Vaadin. Jsou překládány klientské části komponent do jazyka JavaScript a dále zdrojové kódy serverových částí komponent do bytecode. Posledním prvkem překladu jsou soubory s vizuálními styly, které je nutné převést z formátu SASS do CSS. Po dokončení překladu jsou všechny vygenerované soubory uloženy do balíku Java (soubor typu .JAR), jež lze používat jako knihovnu v aplikacích.

5 Aplikace pro správu znalostí

V rámci vývoje komponent byla vytvořena i jednoduchá aplikace pro správu znalostí uživatelů. Aplikace je postavena na jádře vyvíjeném ve společnosti Marbes Consulting, zajišťující základní společnou funkčnost všech aplikací. Důvodem vzniku aplikace je ověření funkčnosti komponent a dále vytvoření systému pro správu znalostí a životopisů zaměstnanců firmy.

Doposud byly životopisy zaměstnanců uchovávány v nestrukturované formě (v dokumentech MS Word). Tento způsob s sebou přináší několik omezení, ať již nemožnost vyhledávání podle kritérií, rozdílnou strukturu dokumentů, anebo zdlouhavý způsob aktualizace znalostí zaměstnanců.

5.1 Základní informace

Modul pro správu životopisů není samostatná spustitelná aplikace. Jedná se o rozšíření komplexního systému pro správu lidských zdrojů, projektů a obchodních procesů a tvoří pouze malou součást rozsáhlé funkčnosti. V aplikaci jsou vedeni zaměstnanci firmy spolu se všemi potřebnými informacemi. Chybí ovšem informace o jejich znalostech a zkušenostech s různými technologiemi, jež by sloužily jako data pro životopisy. Právě k tomu především je vytvořen tento rozšiřující modul umožňující evidovat všechny znalosti na jednom místě. Pokud zaměstnanci svoje zkušenosti zadají do tohoto systému, bude možné generovat takové životopisy obsahující pouze informace potřebné pro získání daného projektu.

5.2 Struktura znalostí

Znalosti jsou rozděleny do několika úrovní stromové struktury, na jejímž vrcholu jsou oblasti znalostí. Pod jednou oblastí si lze představit například informační technologie. Další úrovní spadající pod oblast jsou kategorie představující jemnější rozdělení společných znalostí. V případě technologií může jít o kategorie programovací jazyky, správa databází, operační systémy atd. Poslední částí struktury jsou konkrétní znalosti. Ke každé znalosti je definován výčet možných úrovní, kterých může zaměstnanec dosáhnout. Není možné mít společné úrovně pro všechny kategorie, jelikož jsou definované slovně a mají různou váhu vyjadřující zkušenost s danou znalostí. Například pokud programovací jazyky mají úrovně jako junior, senior nebo profesionál, v případě

účetnictví se může jednat o počet let praxe, a tedy úrovně bez praxe, 2 roky praxe, 5 let praxe atd.

5.3 Funkce aplikace

V systému pro správu znalostí a životopisů jsou definovány dvě role uživatelů, a to administrátor a běžný zaměstnanec. Administrátor zajišťuje naplnění a průběžnou aktualizaci všech číselníků, tj. jednotlivých oblastí, kategorií a znalosti s úrovněmi. Má také oprávnění na editaci dosažených znalostí zaměstnanců anebo generovat jejich životopisy.

Běžný uživatel aplikace může pracovat pouze s přehledem vlastních dosažených znalostí a měnit dosažené úrovně. Nemá možnost přidávat další údaje, ať již znalosti nebo kategorie do databáze. Mezi další jeho oprávnění patří generování vlastního životopisu nebo filtrování zaměstnanců podle znalostí.

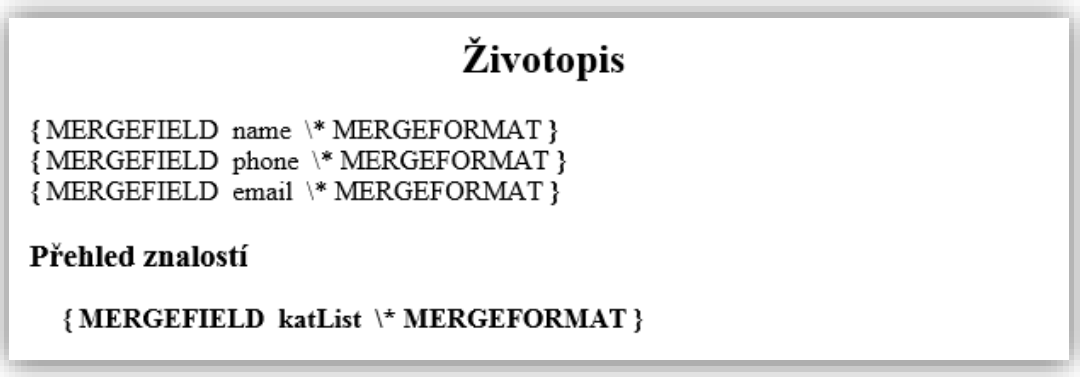
Filtrování zaměstnanců

V aplikaci je vytvořena stránka pro výpis zaměstnanců splňujících určité podmínky. Každý uživatel má možnost filtrovat zaměstnance podle znalosti a dosažené úrovně. Uživatel vybere požadovanou znalost s minimální dosaženou úrovní a přidá filtr. Těchto filtrů může být aplikováno několik, přičemž je nalezen průnik vyhledaných záznamů pro jednotlivé filtry. Administrátor si může navíc nechat vygenerovat životopisy pro jednotlivé nalezené zaměstnance.

5.4 Generování životopisů

Jak bylo již zmíněno výše, hlavním úkolem systému je automatické generování životopisů ze zadaných znalostí zaměstnanců. Generovat lze jak jeden životopis, jenž je přes internetový prohlížeč stažen do počítače jako dokument, tak i celý ZIP archiv všech vybraných zaměstnanců.

Generování výsledných dokumentů zajišťuje třída `RTFGeneratorController`, která podle id zaměstnance získá jeho znalosti z databáze a vytvoří z nich dokument ve formátu RTF. Samotné generování je v principu skládání šablon vytvořených taktéž v RTF dokumentu. V šabloně jsou definována pole (jde o slučovací pole MS Word), do nichž lze vkládat data. Každá část životopisu představuje jednu šablonu. Pokud je datový zdroj šablony tvořen více záznamy (kolekce), pak se pro každý záznam vloží odpovídající kopie šablony.



Obrázek 17: Ukázka šablony životopisu

Na obrázku 17 je ukázka hlavní šablony, do níž jsou vkládána data s vygenerovanými kategoriemi. Šablona je zde pouze jako ukázková ve formátu RTF. Může být editovaná v jakémkoliv editoru pracujícím s tímto formátem. Ve výsledném životopisu bude samozřejmě vypadat jinak, důležité pouze je, aby osahovala všechna definovaná pole. K vygenerování životopisu slouží metoda `merge()` z generátoru `RTFTransformer` vytvořeném ve společnosti Marbes, jež jako parametr přijímá mapu, kde klíčem je identifikátor pole v šabloně (na obrázku například „*name*“) a hodnotou text vypsaný do šablony. V případě pole „*katList*“ je hodnotou v mapě seznam předem vygenerovaných kategorií.

5.5 Přehled znalostí zaměstnanců

Aplikace obsahuje stránku, na níž je vidět přehled všech oblastí, kategorií, znalostí a úrovní definovaných v databázi spolu se znalostmi zaměstnanců. Jedná se o tabulku, kde sloupce představují dosaženou úroveň, řádky jsou děleny na oblasti, kategorie až po jednotlivé znalosti a v buňkách tabulky je uveden seznam zaměstnanců, kteří mají v životopisu danou znalost a úroveň.

Pro přiřazení zaměstnance do dané úrovně znalostí je připraveno dialogové okno, v němž lze vyhledávat osoby. Mezi buňkami lze také zaměstnance přetahovat myší, kdy při stisknuté klávese CTRL dochází ke kopírování osoby do dalších buněk.

5.6 Implementace Drag&Drop

Uchycení objektu myší, jeho přetažení na jiný objekt a následné upuštění je součástí Vaadin frameworku a dovoluje nastavit přesouvání téměř jakýchkoliv komponent splňujících určitá kritéria. Jsou zde implementovány metody pro přesouvání řádků v rámci tabulek, uzlů ve stromech nebo vlastních komponent ze zdrojového umístění

do cílového, kde je při upuštění vyvolána událost. V případě aplikace pro správu životopisů je použit třetí způsob, a to přesouvání vlastních komponent.

5.6.1 Přesouvaný objekt

Každá komponenta, která může být zachycena myší a přetažena, musí být potomkem třídy `DragAndDropWrapper`. Tato třída přijímá v parametru konstruktoru odkaz na přesunovatelnou komponentu. Jde pouze o obalovou třídu, jež není viditelná a je vkládána do stránky místo naší komponenty.

5.6.2 Příjem přesouvaného objektu

Přijímacím objektem může být opět třída, jež je potomkem zmíněné `DragAndDropWrapper`. Třída implementuje jak rozhraní pro možnost přesunutí, tak rozhraní pro příjem přesouvaných objektů. V aplikaci pro správu životopisů jsou těmito potomky buňky tabulky v přehledu všech znalostí obsahující seznam uživatelů, kteří mají odkazovanou znalost dané úrovně. Metodou `setDropHandler()` je nastavena obsluha události upuštění objektu implementující metodu `drop()`, v níž se odehrává všechna logika s přesunutím. Na přetahovaný objekt je uvnitř metody odkazováno jako na třídu `WrapperTransferable`, přes jejíž vlastnosti se lze dostat až k samotnému přesouvanému objektu, v našem případě na objekt uživatele.

Při upuštění objektu nad buňkou proběhne kontrola, jestli může být uživatel do tohoto pole zařazen. Například ve stavu, kdy má uživatel splněnou znalost úrovně junior, nemůže mít zároveň stejnou znalost dosaženou s úrovní profesionál. V tomto případě probíhá kontrola pouze při upuštění objektu, během tažení uživatel nepozná, jestli může být do daného místa objekt přesunut.

5.6.3 Podmínky pro příjem objektu

Pro průběh přesouvání lze definovat podmínky, za nichž je možné objekt nad komponentou upustit. Tato kontrola je nastavitelná jak na klientské části, kdy se například přijímací komponentě určí, jaké typy přesouvaných objektů mohou být upuštěny, tak na části serverové, kde lze akceptovat objekt podle komplexnějších kritérií. Bohužel ani jeden způsob není moc vhodný pro vytvořenou aplikaci správy životopisů. V prvním případě je přesouván vždy pouze jeden typ komponenty, kterým je uživatel, a tudíž je toto omezení vždy splněno. V případě kontroly na serverové části probíhá při každém přesunu nad přijímací komponentu komunikace na server, kde se vypočte akceptace

přetahovaného objektu a odešle odpověď na klientskou část. Jelikož může pohyb do cílové buňky přejít přes několik dalších, pro něž jsou vypočítávána kritéria přijetí, je zbytečně zatěžován komunikační kanál a navíc je reakce výrazně pomalejší.

6 Závěr

V první fázi vypracování této diplomové práce bylo nejdůležitější detailní seznámení s frameworkem Vaadin 7 a všemi nezbytnými podpůrnými technologiemi, bez nichž by nebyl vývoj aplikací téměř možný. Jednalo se především o samotný frameworkVaadin, jenž byl pro mě do té doby naprosto neznámý, a také většina částí tvořící jádro aplikací, jimiž jsou především Java EE spolu s frameworky Java Spring a Hibernate. Další zkušenosti bylo nutné získat s nástroji jako jsou Apache Maven a Apache Subversion. Vzhledem k tomu, že se jedná o specializované nástroje především na programovací jazyk Java, je jejich výuce na univerzitě věnován velmi malý prostor.

V této práci se mi podařilo úspěšně dosáhnout všech stanovených cílů. V první části jsou zmíněny základní používané technologie pro vývoj aplikací ve společnosti Marbes Consulting s.r.o.. Dále následuje základní popis frameworku Vaadin 7 s jeho nejvýznamnějšími funkcemi a porovnáním se současnými největšími konkurenty v oblasti komponentně orientovaných frameworků v jazyce Java. V druhé části bylo dosaženo hlavního cíle práce, jímž bylo vyvinutí řady vizuálních komponent nezbytných pro tvorbu webových Vaadin aplikací. Tyto komponenty byly vytvářeny především za účelem stanovení jednotného stylu vzhledu a jednoduché použitelnosti ve zdrojovém kódu vedoucí k podstatnému urychlení vývoje aplikací. V poslední části práce je popsán modul pro aplikaci společnosti sloužící ke správě znalostí zaměstnanců a k automatickému generování jejich životopisů do dokumentu formátu RTF.

Tato práce pro mě byla velkým zkušenostním přínosem, především ve vývoji a správě velkých aplikací vybudovaných na platformě Java EE. Detailně jsem se seznámil s frameworkem Vaadin 7 a vytvořil několik základních vizuálních komponent pro prezentační vrstvu budoucích aplikací. Podle mého názoru framework Vaadin dosáhl svých požadovaných cílů, kdy umožňuje jednoduchou tvorbu webových aplikací, které se svým vzhledem a funkčností přibližují desktopovým. Při srovnání se zmíněnými konkurenčními frameworky lze framework Vaadin doporučit pro vývoj všem budoucím zájemcům, kteří se rozhodují nad velkým množstvím technologií, jež jsou v dnešní době dostupné.

Přehled zkratek

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface (Aplikační rozhraní)
CSS	Cascading Style Sheets (kaskádové styly)
CVS	Concurrent Version System
EIS	Enterprise Information System
EJB	Enterprise Java Beans
GWT	Google Web Toolkit
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JAR	Java ARchive
Java EE	Java Enterprise Edition
Java SE	Java Standard Edition
JSON	JavaScript Object Notation
JSP	JavaServer Pages
MVC	Model-view-controller
ORM	Objektově relační mapování
POM	Project Object Model
REST	Representational State Transfer
RIA	Rich Internet Application
RPC	Remote Procedure Call
RTF	Rich Text Format
SASS	Syntactically Awesome Stylesheets
SOAP	Simple Object Access Protocol
SVG	Scalable Vector Graphics
SVN	Subversion
UI	User interface
URL	Uniform Resource Locator
XML	Extensible Markup Language

7 Literatura

1. BEN, C.S. B. FITZPATRICK a M. PILATO. *Version Control with Subversion: For Subversion 1.7* [online]. PDF edition. 2011, 446 s. [cit. 2014-03-10]. Dostupné z: http://svnbook.red-bean.com/en/1.7/svn-book.pdf

2. BERNARD, B. Úvod do architektury MVC. In: *Zdroják.cz* [online]. 07. 05. 2009 [cit. 2014-03-11]. Dostupné z: http://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/

3. DRESLER, R. Vícevrstvé architektury aplikací. In: *Robert Dresler* [online]. 26. 04. 2011 [cit. 2014-03-02]. Dostupné z: http://www.robertdresler.cz/2011/04/vicevrstve-architektury-aplikaci.html#posouzeni

4. FOUNDATION, T. A. S. Introduction. *Apache Maven Project* [online]. 2002-2014 [cit. 2014-02-20]. Dostupné z: http://maven.apache.org/what-is-maven.html

5. FRANKEL, N. *Learning Vaadin*. 1st edition. Birmingham (UK): Packt Publishing Ltd, 2011, 412 s.. ISBN 9781849515238.

6. GONCALVES, A. *Beginning Java EE 7*. ilustrované vydání. New York City: Apress, 2013, 608 s.. ISBN 9781430246268.

7. KADLEC, T. Úvod do Java EE. In: *edux.feld.cvut.cz* [online]. 07. 09. 2009 [cit. 2014-03-15]. Dostupné z: https://edux.feld.cvut.cz/howto/slides/example

8. MAPLE, S. The Curious Coder's Java Web Frameworks Comparison. In: *ZeroTurnaround* [online]. 30. 07. 2013 [cit. 2014-04-05]. Dostupné z: http://zeroturnaround.com/rebellabs/the-curious-coders-java-web-frameworks-comparison-spring-mvc-grails-vaadin-gwt-wicket-play-struts-and-jsf/#!/

9. MARKO GRÖNROOS. *Book of Vaadin: Vaadin 7 Edition* [online]. 2nd revision. Finland: Vaadin Ltd, 2014, 618 s. [cit. 2014-03-15]. Dostupné z: https://vaadin.com/download/book-of-vaadin/vaadin-7/pdf/book-of-vaadin.pdf

10. NOVOTNÝ, J. K čemu je nám užitečný komponentový web framework? In: *blog.novoj.net* [online]. 23. 03. 2013 [cit. 2014-04-02]. Dostupné z: http://blog.novoj.net/2013/03/23/k-cemu-je-nam-uzitecny-komponentovy-web-framework/

11. ROONEY, G. a D. BERLIN. *Practical Subversion*. 2. vydání. Apress, 2006, 304 s.. ISBN 9781430203568.

12. SRIRANGAN, I. *Apache Maven 3 Cookbook*. Birmingham (UK): Packt Publishing Ltd, 2011, 224 s.. ISBN 9781849512459.

13. TACY, A. et al. *GWT in Action*. 2nd ed. Manning Publications Company, 2013, 680 s.. ISBN 9781935182849.

Seznam obrázků

Přílohy

Příloha A. Ukázka vygenerovaného životopisu

Životopis

Tomáš Kubový
605123456
greenbery@centrum.cz

Přehled znalostí

Program jazyky

Java	*Znalý*
Delphi	*Junior*

Správa DB serveru

MySQL	*Znalý*
Oracle	*Junior*
MSSQL	*Junior*

Správní řízení

Správní řád	*Junior*

Účetnictví

Hmotný majetek	*Bez praxe*
Nehmotný majetek	*Bez praxe*