Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Studijní program : Aplikovaná informatika

Obor: Informační systémy a technologie

JSF framework pro komplexní vizualizaci dat

DIPLOMOVÁ PRÁCE

Student : Bc. Martin Linha

Vedoucí : Mgr. Zbyněk Šlajchrt

Oponent : Mgr. Jiří Vitinger

2015

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

V Praze dne 27. dubna 2015 . . . . . .

Bc. Martin Linha

Poděkování

Rád bych touto cestou poděkoval Mgr. Zbyňku Šlajchrtovi za odborné rady, připomínky a metodické vedení práce a Mgr. Jiřímu Stuchlíkovi za cenné implementační poznatky.

Abstrakt

Diplomová práce je zaměřena na vývoj JSF frameworku poskytující komponenty pro komplexní datové vizualizace. Její cíl se skládá z realizace API pro tvorbu JSF komponent vykreslujících komplexní grafy založené na JavaScript knihovně C3.js a z následné implementace sady komponent grafů za použití tohoto API. Přínosem této práce je tedy knihovna poskytující nástroj pro tvorbu nových JSF komponent založených na C3.js společně se sadou okamžitě použitelných komponent. Na počátku je provedena rešerše relevantních JSF knihoven, pokračováno je analýzou a návrhem API. Na tomto základě je implementováno API, ve kterém je následně realizována komponentová sada. Součástí práce je uživatelská příručka, referenční příručka API a prezentační web implementovaných komponent.

Klíčová slova

JavaServer Faces, C3.js, JSF knihovna komponent, vývoj JSF komponent, vizualizace dat

Abstract

The thesis is focused on the development of JSF framework providing components for complex data visualizations. Its objective consists of the implementation of the API for creating JSF components rendering complex charts based on the JavaScript library C3.js and subsequent implementation of a set of chart components using this API. The contribution of this thesis is a library providing a tool for creating new JSF components based on C3.js together with a set of ready to use components. It begins with research of relevant JSF libraries, following with API analysis and design. Based on that is API implemented, in which a component set is then created. As a part of this work is a user guide, API reference guide and presentation web of implemented components.

Keywords

JavaServer Faces, C3.js, JSF component library, JSF component development, data visualization

Obsah

1 Úvod 1

1.1 Cíl práce ... 2

1.2 Přínos a cílová skupina ... 2

2 Rešerše ... 3

2.1 Práce zaměřené na JSF a vizualizace dat ... 3

2.2 JSF knihovny pro pokročilé zobrazování dat ... 4

2.2.1 Grafy jako součást komplexní knihovny ... 4

2.2.2 Specializované JSF knihovny pro vizualizaci dat ... 5

2.3 Shrnutí a smysl vývoje vlastního řešení ... 6

3 C3.js a JavaServer Faces ... 7

3.1 C3.js ... 7

3.2 JavaServer Faces ... 7

3.2.1 Historie ... 8

3.2.2 Architektura ... 9

3.2.3 Základní vlastnosti ... 9

3.2.4 Komponentový model ... 10

3.2.5 Životní cyklus zpracování požadavku ... 11

3.2.6 Event model ... 12

3.2.7 Podpora technologie AJAX ... 12

3.3 Shrnutí ... 12

4 Analýza a návrh ... 13

4.1 Metodika vývoje ... 13

4.2 Analýza C3.js ... 13

4.2.1 Analýza syntaxe a struktury kódu deklarace grafu ... 13

4.2.2 Analýza syntaxe a struktury kódu API ... 16

4.3 Specifikace požadavků ... 18

4.4 Analýza požadavků ... 19

4.4.1 Způsob realizace požadavků v JSF ... 20

4.5 Návrh architektury ... 21

4.5.1 Deklarace grafu ... 22

4.5.2 Selekce datových sad a události o změnách ... 23

4.5.3 Modifikace vizuálních vlastností ... 23

4.6 Návrh API ... 24

4.6.1 Skript pro konstrukci grafu (A01) a modifikace grafu (A03) ... 24

4.6.2 Vybrána datová sada (A02) ... 26

4.7 Implementace API ... 27

4.7.1 Implementace API pro tvorbu komponent ... 27
4.7.2 Implementace API pro tvorbu vizuálních vlastností a jejich modifikací ... 30
4.7.3 Programátorské testy implementace API ... 33

5 Použití API pro tvorbu vizuálních komponent ... 35
5.1.1 Implementace sady komponent vykreslující grafy ... 35
5.1.2 Implementace základní sady vizuálních vlastností ... 36
5.1.3 Implementace tříd pro dynamické modifikace ... 37
5.1.4 Zhodnocení použitelnosti API ... 37

6 Uživatelská příručka komponentové sady ... 38
6.1 Základní nastavení ... 38
6.2 Deklarace grafu ... 38
6.2.1 Datové sady grafu ... 41
6.3 Deklarace vizuálních vlastností grafů ... 41
6.3.1 Tag barProperties ... 41
6.3.2 Tag donutProperties ... 42
6.3.3 Tag gaugeProperties ... 42
6.3.4 Tagy grids, gridX a gridY ... 43
6.3.5 Tag legend ... 43
6.3.6 Tag padding ... 44
6.3.7 Tag pieProperties ... 45
6.3.8 Tag point ... 45
6.3.9 Tagy regions a region ... 46
6.3.10 Tag size ... 46
6.3.11 Tag subchart ... 47
6.3.12 Tag tooltip ... 47
6.3.13 Tag zoom ... 47
6.4 Práce s vybranou datovou sadou na grafu ... 48
6.5 Aktivace AJAX ... 48
6.6 Posluchače změn ... 49
6.7 Dynamické modifikace vizuálních vlastností grafu ... 49
6.7.1 Programatické získání a změna vizuální vlastnosti grafu ... 50
6.7.2 Modifikovatelné vizuální vlastnosti ... 51
6.7.3 Programatické přidání dynamické vlastnosti ... 51

7 Dokumentace API ... 53
7.1 Přehled ... 53
7.2 Příkladné implementace pomocí API ... 53
7.2.1 Implementace komponenty pro graf ... 53
7.2.2 Implementace vlastnosti grafu ... 55
7.2.3 Implementace komponenty vizuální vlastnosti grafu ... 56

7.2.4 Implementace modifikátoru ... 57
7.3 Referenční příručka ... 59
7.3.1 Balíček com.martinlinha.c3faces.component ... 59
7.3.2 Balíček com.martinlinha.c3faces.jsf ... 60
7.3.3 Balíček com.martinlinha.c3faces.component.property ... 61
7.3.4 Balíček com.martinlinha.c3faces.listener ... 62
7.3.5 Balíček com.martinlinha.c3faces.listener.change ... 63
7.3.6 Balíček com.martinlinha.c3faces.script ... 64
7.3.7 Balíček com.martinlinha.c3faces.util ... 67
7.3.8 JavaScript API ... 69

8 Závěr 71

Terminologický slovník ... 73

Seznam literatury ... 75

Seznam obrázků a tabulek ... 78
Seznam obrázků ... 78
Seznam tabulek ... 78

Příloha A: Systémové požadavky ... 80

Příloha B: Integrace s frameworkem PrimeFaces ... 81

Příloha C: Zdrojové kódy ... 83

1 Úvod

V dnešní době existuje velmi rozmanitý výběr rámců určených k tvorbě webových aplikací v jazyce Java. Počet a používanost jednotlivých frameworků se dynamicky vyvíjí. Jen nepatrný počet z nich si dokázalo vybudovat stabilní a dlouhodobou podporu jak ze strany jejich tvůrců, uživatelů, tak komunity. To, že JavaServer Faces (JSF) je jedním z nich, jistě umocňuje skutečnost, že je již řadu let součástí Java EE specifikace. Síla JSF spočívá především v jeho základní filozofii. Jedná se totiž o komponentově založený framework. Programátoři tak mohou vytvářet vlastní, přepoužitelné sady komponent podporující například specifickou oblast webových aplikací. Není tedy zvláštností, že vývojářské společnosti využívající tuto technologii tvoří vlastní sady komponent, které pak přepoužívají v různých projektech. Ač může vývoj vlastní knihovny komponent s sebou nést vyšší náklady, v budoucnu lze s jejich znovupoužitím dosáhnout výrazné úspory. S rostoucí popularitou JSF se přirozeně začaly objevovat i řady knihoven, které byly uvolněny k volnému použití.

Pracuji ve vývojářské firmě, která tento framework využívá na řadě projektů. Sám se s ním tedy setkávám každý pracovní den. V nedávné době jsem se ocitl v situaci, kdy mi bylo zadáno implementovat obrazovku obsahující různá zobrazení dat prostřednictvím grafů. Součástí zadání byl požadavek na nadprůměrně komplexní animace, přechody a chování těchto grafů. Za mou programátorskou praxi jsem se nesetkal s JSF knihovnou, která by obsahovala komponenty splňující takovéto požadavky. Řešení úkolu nakonec spočívalo ve využití JavaScriptové knihovny C3.js, která takové možnosti poskytuje. Ve výsledku se jednalo o psaní JavaScript kódu, čehož abych byl schopen, jsem musel tuto knihovnu nejdříve nastudovat. Nebylo tedy využito komponentového přístupu, který rámec JSF nabízí.

Z těchto důvodů vznikla motivace stojící za touto prací. Vyplývá z potřeby takové sady komponent, která by obsahovala komplexní grafy pro framework JavaServer Faces. Vývojář by při tvorbě podnikových aplikací nemusel vytvářet nezbytný JavaScript kód pro vygenerování C3.js vizualizací a od této činnosti by byl kompletně odstíněn. Stačilo by mu přidat do svého Java projektu zmíněnou knihovnu a pak již jen velmi jednoduše na libovolném místě definovat komponentu. Její vlastnosti by pak mohl určovat na základě atributů, které by komponenty poskytovaly prostřednictvím svého rozhraní.

Na počátku této diplomové práce prostřednictvím rešerše ověřuji, zda hledané řešení již nebylo realizováno. Po krátké teoretické části pokračuji analýzou. Zde je provedena analýza kódu C3.js a jeho struktury. Následně je navržena architektura programu společně s API frameworku. Na základě návrhu implementuji API, v kterém je posléze implementována knihovna JSF komponent generující C3.js grafy. V závěru práce je pak k

nalezení uživatelská příručka pro vytvořenou komponentovou sadu společně s dokumentací API pro tvorbu komponent nových.

1.1 Cíl práce

Cíl této práce byl rozdělen na dvě části.

1. První část spočívá v naleznutí abstrakce nad grafy knihovny C3.js, jenž by byla použitelná v prostředí JavaServer Faces, a její následné implementaci. Vznikl by tak framework pro tvorbu komplexních grafů nad JSF. Pokud by bylo žádoucí vytvářet nové JSF komponenty vykreslující C3.js grafy, framework by prostřednictvím API podporoval podstatnou část celého procesu.
2. Druhá část tkví v realizaci knihovny JSF komponent generující C3.js grafy s definovatelnými vizuálními vlastnostmi. Vývojář by tak měl k dispozici sadu komponent připravenou k okamžitému použití bez nutnosti znát detaily C3.js nebo API vytvořeného frameworku. Pouze by deklaroval tagy komponent ve Facelet stránce tak, jak je zvyklý. Knihovna komponent bude vyvinuta ve vzniklém frameworku a její součástí bude prezentační web.

1.2 Přínos a cílová skupina

Přínosy této diplomové práce jsou tvořeny JSF frameworkem pro tvorbu grafů z JavaScriptové knihovny C3.js a implementovanou knihovnou vizuálních komponent k okamžitému použití. Vývojář v JavaServer Faces tak deklarativním způsobem bude moci přidávat na své stránky komplexní grafy a to bez psaní JavaScript kódu. Bude-li v budoucnu knihovna C3.js rozšířena o nové typy grafů nebo jejich vizuálních atributů, vytvořený framework značně ulehčí jejich přenos do prostředí JSF.

Cílová skupina je tvořena především vývojáři webových aplikací ve frameworku JavaServer Faces. Zároveň je však tvořena i uživateli jiných webových rámců postavených nejen na jazyce Java, jelikož práce demonstruje výhody komponentového přístupu jako takového.

2 Rešerše

Aby bylo potvrzeno, že neexistuje adekvátní knihovna obsahující JSF komponenty pro vykreslování komplexních grafů, byla provedena následující rešerše. Kapitola se věnuje zmapování akademických prací týkajících se této oblasti jak na národní, tak mezinárodní úrovni a tržní situace v oblastech komplexních JSF knihoven a knihoven specializovaných na vizualizace dat prostřednictvím grafů.

Následují výsledky řešerše včetně okomentovaných závěrů.

2.1 Práce zaměřené na JSF a vizualizace dat

Prvním krokem rešerše bylo zkoumání prací týkajících se vývoje v JavaServer Faces. Zdrojem byla především databáze akademických prací na *Theses.cz*, dále pak různé databáze knihoven vysokých škol a fulltextové vyhledávání na *Google.com*.

Bakalářských a diplomových prací týkajících se oblasti JavaServer Faces byla sepsána celá řada. Častým zaměřením je hlubší analýza některé z existujících komponentových knihoven. Příkladem jsou práce týkající se frameworku *RichFaces* a *Primefaces*. Práce Kořistkové se zaměřuje na jednotlivé aspekty této knihovny a v praktické části implementací referenční aplikace [2]. Podobnou strukturou se vyznačuje i bakalářská práce Mačouna, týkající se vývoje v JSF za použití knihovny PrimeFaces [3]. V obou případech se však jedná o používání již vytvořených komponent a nikoliv o vývoj komponent nových.

Práce týkající se vlastního vývoje komponent je znatelně méně. Na národní úrovni diplomová práce byla identifikována pouze jediná. Jedná se o práci s názvem *Framework pro návrh responzivního uživatelského rozhraní nové generace*. Rychtecký k dosažení definovaného cíle mimo jiné navrhne a následně implementuje celkem 6 komponent, které dodržují pravidla responzivního webu [6]. Na úrovni zahraniční byly v rámci rešerše identifikovány diplomové práce v počtu jednotek avšak žádná z nich se netýkala vývoje komponent pro podporu vizualizace dat a nebudou tedy hlouběji rozebírány.

Lze tedy shrnout, že nebyla nalezena žádná bakalářská či diplomová práce zaměřená na vývoj komponent v oblasti vizualizace dat prostřednictvím grafů.

2.2 JSF knihovny pro pokročilé zobrazování dat

2.2.1 Grafy jako součást komplexní knihovny

Komplexní JSF knihovny poskytující komponenty pokrývající vývoj podnikových aplikací často obsahují jako součást svého balíčku i komponenty pro vykreslování grafů. K této části rešerše byl využit srovnávač *JSFMatrix.com*, který nabízí maticové srovnání nejpopulárnějších JSF komponentových knihoven. Z výsledků vyplývá, že frameworky *PrimeFaces*, *RichFaces*, *ICEfaces* a *MyFaces Trinidad* nabízejí v rámci celé komponentové sady i komponenty určené pro vizualizaci dat [25]. Budou tedy zkoumány blíže.

Knihovna *PrimeFaces* nabízí celkem 17 komponent pro vykreslování grafů, které jsou součástí základní distribuce tohoto rámce[1] [24]. Generování grafů je zde založeno na JavaScriptové knihovně *jqPlot*, která je uvolněna jako Open Source plugin pro JavaScriptový framework *jQuery* [27].

JSF knihovna komponent od společnosti *JBoss*, nesoucí název *RichFaces*, od verze 4.5 také obsahuje komponentu pro vytváření grafů. *RichFaces* využívají kapacit JavaScriptové knihovny třetí strany s názvem *Flot* [19]. Výhodou *Flot* knihovny je její kompatibilita se staršími prohlížeči, včetně Internet Explorer verze 6.0 [20].

ICEsoft, tvůrci knihovny *ICEfaces*, obdobně jako *PrimeFaces* využili možností poskytovaných knihovnou *jqPlot* [29].

Apache MyFaces Trinidad využívá vlastní knihovnu [30].

Výsledky rešerše komplexních komponentových sad v závislosti na použité JavaScript knihovně shrnuje Tabulka 1 níže.

Tabulka 1: *Seznam komplexních JSF knihoven, poskytujících komponenty pro vykreslování grafů, a názvy JavaScript knihoven vykreslujících grafy (Zdroj: autor)*

JSF knihovna	JavaScript knihovna pro datové vizualizace
PrimeFaces	jqPlot
RichFaces	Flot
ICEfaces	jqPlot

[1] PrimeFaces kromě základní distribuce nabízí další komponenty v knihovně nazvané PrimeFaces Extensions, zde však nejsou zahrnuty komponenty pro vizualizace dat

JSF knihovna	JavaScript knihovna pro datové vizualizace
MyFaces Trinidad	*vlastní*

Je možné konstatovat, že komplexní knihovny možnosti vykreslování grafů poskytují. Až na jeden případ využívají JavaScript knihovny třetích stran. Jejich výhodou je kompatibilita se staršími prohlížeči prostřednictvím technologie VML. Vzhledem však k povaze použitých JavaScript knihoven se nejedná o natolik komplexní grafy, jež jsou cílem této práce.

2.2.2 Specializované JSF knihovny pro vizualizaci dat

K naleznutí jsou i knihovny, které se na problematiku grafů specializují. Ve veřejném katalogu knihoven na oficiálních komunitních stránkách *JSFCentral.com* byly nalezeny celkem 3 knihovny, jež se přímo specializují na datové vizualizace prostřednictvím JSF [27]. Výsledky této části rešerše opět oproti použité JavaScript knihovně shrnuje Tabulka 2 obsahující pouze 2 ze zmíněných knihoven [32] [33] . Třetí, Exadel Fiji, je postavena na technologii Adobe Flash a proto nebude dále zkoumána [31].

Tabulka 2: *Seznam JSF knihoven zaměřených na vizualizaci dat a podpůrných JavaScript knihoven (Zdroj: autor)*

JSF knihovna	JavaScript knihovna pro datové vizualizace
JSFlot	Flot
Flot4Jsf2	Flot

Z rešerše JSF knihoven specializovaných na grafy vyplývá, že obdobně jako některé knihovny z kapitoly 2.2.3 využívají výše zkoumané rámce JavaScript knihovnu *Flot*. Lze tedy zopakovat závěr z předchozí kapitoly. Nejedná se o natolik komplexní grafy, které by uspokojovaly předmět této práce.

V době realizace této práce byla dne 25. ledna přidána na stránky *JSFCentral.com* knihovna *HighFaces*. Tato knihovna poskytuje JSF komponenty pro vykreslování grafů založené na JavaScriptové knihovně *HighCharts* [27]. Na základě skutečnosti, že v době uvedení této knihovny byla rešerše již realizována a vývoj vlastního řešení již zahájen, nebyla knihovna *HighFaces* brána v této rešerši v úvahu.

2.3 Shrnutí a smysl vývoje vlastního řešení

C3.js je moderní knihovna poskytující široké možnosti tvorby rozsáhlé palety datových vizualizací založených na grafech. JSF oproti tomu je komponentově založený framework, který umožňuje přepoužívat jednou vytvořené objekty jako komponenty a tak rapidně urychlit vývoj. Využití JSF komponent navíc odstiňuje vývojáře od potřeby znát specifika technologie ležící pod danou komponentou.

Jak lze konstatovat z provedené rešerše, komponenty pro tvorbu grafů prostřednictvím JavaScriptových knihoven třetích stran úspěšně využívají dnešní dominantní frameworky postavené nad JSF. Z rešerše je však především zřejmé, že neexistuje JSF knihovna, která by C3.js grafy poskytovala a zároveň neexistuje JSF knihovna poskytující komplexní grafy jako jsou ty, které produkuje C3.js.

Chce-li JSF vývojář integrovat do své aplikace tyto komplexní grafy, nemůže využít kapacit JSF frameworku a musí se uchýlit k implementaci prostřednictvím JavaScript kódu. Vývoj JSF rámce pro JavaScript knihovnu C3.js tento problém řeší a je tedy zřejmým důvodem vývoje tohoto řešení.

3 C3.js a JavaServer Faces

Cílem této kapitoly je položit teoretické základy o technologiích C3.js a JavaServer Faces. Soustředěno bude především na aspekty, které souvisí s předmětem této práce.

3.1 C3.js

C3.js je knihovna postavená nad JavaScriptovým frameworkem D3.js, který slouží pro tvorbu složitých datových vizualizací. D3.js ve své zkratce skrývá název Data-Driven Documents, tedy ve volném překladu "dokumenty řízené daty" a jejím tvůrcem je Mike Bostock. Je založena na technologiích SVG, CSS a HTML a vyznačuje především tím, že

- načítá data do paměti prohlížeče,
- navazuje načtená data na elementy dokumentu,
- dynamicky vytváří a odstraňuje elementy podle aktuálně načtených dat,
- poskytuje sofistikované přechodové animace mezi stavy objektů (odstavec viz [14]).

Framework D3.js neposkytuje žádné přepoužitelné vizualizace, jedná se pouze o nástroj podporující jejich tvorbu.

Knihovna C3.js obsahuje sadu přepoužitelných grafů implementovaných v D3.js. Jedná se o grafy s komplexními vizuálními vlastnostmi. C3.js není knihovnou pouze pro vykreslování grafů, ale poskytuje možnosti pro jejich dynamické modifikace prostřednictvím definovaného API. O tomto se zmiňuje Masayuki Tanaka, tvůrce C3.js: *"When I was developing a web application, I wanted to control the state of the chart after it is drawn and fire events on the chart interactively. However, the libraries I found were not designed like that (basically they're designed to just draw a chart)."* [13]. Z této citace mimo jiné vyplývá, že grafy umožňují zachycovat různé nastalé události.

Knihovna obsahuje celkem 11 typů grafů s desítkami nastavitelných vizuálních vlastností [11].

3.2 JavaServer Faces

Tato podkapitola je zaměřena na rámcový popis technologie JavaServer Faces. Vzhledem ke skutečnosti, že možnosti a vlastnosti tohoto frameworku jsou značně rozsáhlé, pozornost bude věnována především těm aspektům, které souvisejí s vývojem komponent a tedy s cílem této práce.

Společně se stručným popisem základních stavebních kamenů bude zmíněna krátká historie.

3.2.1 Historie

Prvním důležitým milníkem v oblasti webového vývoje v jazyce Java se datuje k roku 1998, kdy společnost *Sun Microsystems* představila Java Servlet API [34]. Před představením této specifikace byl vývoj server-side[2] webových aplikacích v jazyce Java velmi problematický. Kritickým problémem pro efektivní vývoj webů bylo velmi problematické generování HTML kódu, který musel být součastí Java tříd.

Značný posun znamenalo představení JavaServer Pages (JSP). Technologie postavená nad Java Servlet API umožnila oddělit prezentační vrstvu od vrstvy aplikační logiky. JSP stránka totiž obsahuje klasické HTML značky a server-side kód je vkládán prostřednictvím speciálních tagů. JSP stránky fungují na základě Servletu nazývaném JSP kontejner. Ten je umístěn na aplikačním serveru a jeho podstatou je překlad jednotlivých JSP stránek do Servletu, který je následně zkompilován a proveden. Dynamický vývoj internetového prostředí působil na zvyšování komplexity webových řešení. Čím víc byly stránky komplexnější, tím více obsahovaly Java kódu a začínaly být méně přehlednými a především pak náchylné k chybám (odstavec viz [34]).

To zapříčinilo vznik prvního MVC frameworku pro Javu s názvem *Apache Struts*. *Struts* právě díky využití návrhového vzoru MVC získalo odpovídající popularitu. Postupem času po vzniku *Spring frameworku* vzniká další MVC framework[3], nazvaný *Spring MVC*. V době, kdy *Spring MVC* spatřil světlo, se již pracovalo na specifikaci pro JSF 1.0 (odstavec viz [5]).

JavaServer Faces byl vytvořen na základě JCP skupinou předních technologických společností. Jmenovitě se jednalo o organizace *Sun Microsystems*, *Oracle*, *Borland*, *BEA* a *IBM*. Na vývoji se však podíleli i další přední experti v oblasti webového vývoje. První požadavek na vytvoření specifikace (JSR 127) se objevil v průběhu roku 2001 a první verze byla oficiálně vydána společně s J2EE verze 1.4 (odstavec viz [5]).

Je nutno podotknout, že JSF framework se od vydání verze 2.0 v roce 2009 (JSR 314) značně změnil. Byla přidána podpora pro tvorbu kompozitních komponent, AJAX a mnohé další.

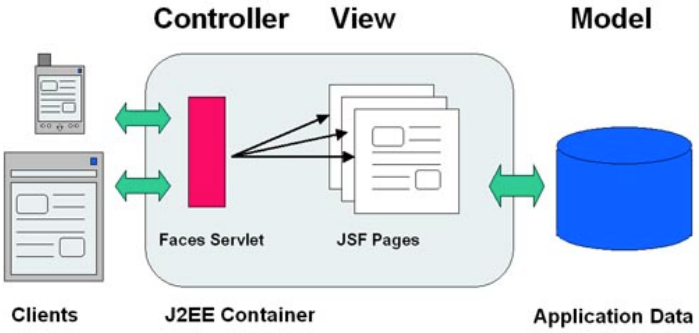
---

[2] V oblasti client-side aplikací Java umožňovala vytvářet tzv. Applets, které běžely ve webovém prohlížeči

[3] Spring Framework představuje alternativu pro tvorbu podnikových aplikací k Java EE

3.2.2 Architektura

Návrh JSF vznikal postupně jako kombinace úspěšných praktik tehdejších webových přístupů a frameworků. Kořeny byly postaveny na úspěšném webovém návrhovém vzoru MVC, stejně jako v té době konkurenční rámce [5]. Třída reprezentující základní Controller obhospodařující příchozí HTTP požadavky a zároveň odpovědi ve formě HTML značek je `javax.faces.webapp.FacesServlet`. Od verze Java EE 6 a JSF 2.0 se pod View vrstvou nalézají především šablonovací stránky Facelets obsahující deklarované komponenty a pod Model vrstvou třídy nazvané Backing beans [7]. Toto rozdělení souhrně znázorňuje Obrázek 1.



Obrázek 1 : *Architektura JSF (Zdroj: [6])*

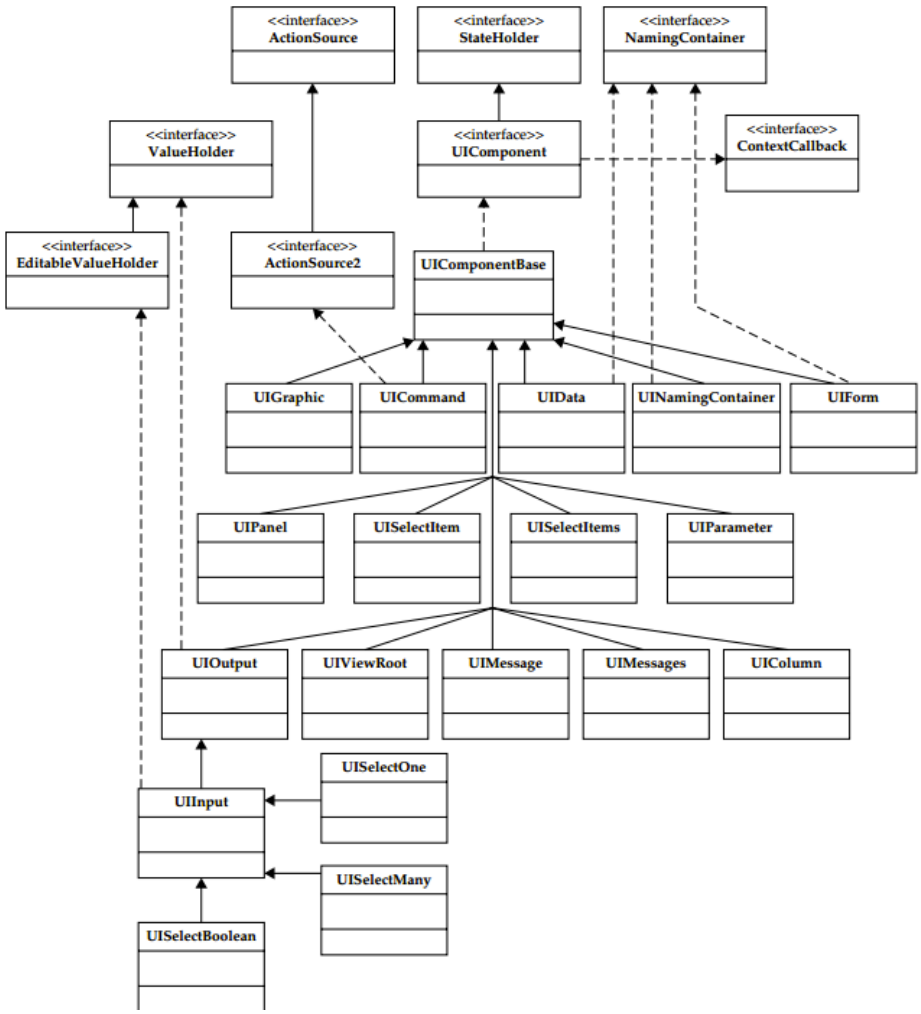
3.2.3 Základní vlastnosti

Současná verze (2.2) se vyznačuje především

- odstíněním uživatele od nutnosti získávání parametrů z HTTP požadavků prostřednictvím životního cyklu zpracování požadavku,
- automatickou správou stavů objektů mezi jednotlivými HTTP požadavky,
- již rozebíraným komponentovým přístupem pro tvorbu uživatelského rozhraní,
- navigačním modelem pro definici možných přechodů mezi stránkami,
- šablonovacím systémem pro maximalizaci možností přepoužitelnosti kódu prostřednictím tzv. Facelets,
- výrazovým jazykem umožňujícím komunikaci mezi prezentační vrstvou a aplikační logikou (Java objekty),
- konzistentní stromovou reprezentací všech použitých komponent na klientské stránce v paměti serveru [7].

3.2.4 Komponentový model

JSF specifikace obsahuje dvě základní knihovny komponent, HTML tag library a Core tag library. První zmíněná slouží pro tvorbu základních HTML tagů za pomoci JSF, Core tag library pak pro možnost definice cyklů, konverterů, validátorů a dalších prvků přímo ve Facelet stránce. Nadtypem všech komponent je rozhraní `javax.faces.component.UIComponent`, jehož základní implementaci poskytuje abstraktní třída `javax.faces.component.UIComponentBase`. Obrázek 2 znázorňuje komponentový model definovaný specifikací. Na obrázku jsou také znázorněna klíčová rozhraní, s kterými JSF pracuje (odstavec viz [5]).



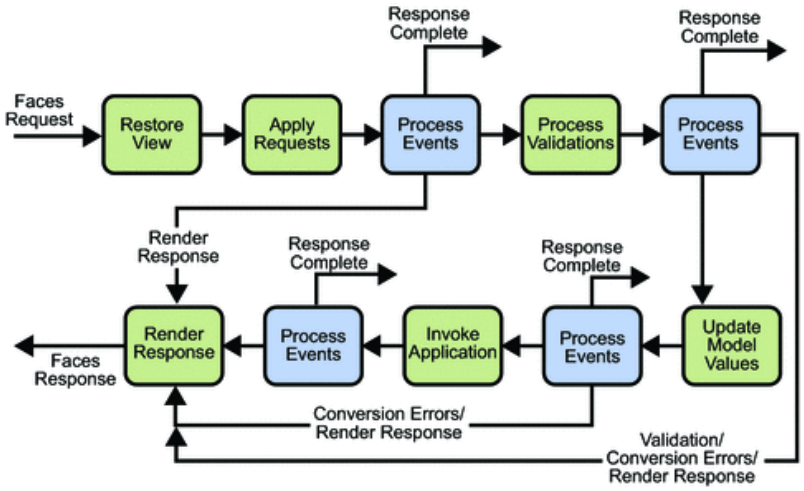
Obrázek 2 : *Komponentový model JavaServer Faces (Zdroj: [5])*

3.2.5 Životní cyklus zpracování požadavku

Životní cyklus zpracování požadavku pokrývá řadu aktivit, které jsou třeba realizovat při získávání a zpracovávání parametrů získaných z HTTP požadavku. Při každém přijatém požadavku je nutné

- validovat přijatá data oproti definovaným validačním pravidlům,
- provést konverzi na odpovídající datové typy,
- aktualizovat stavy objektů na serveru aktualními daty,
- provést případné operace relevantní pro daný požadavek,
- uložit stavy změněných objektů,
- vytvořit odpověď pro požadavek a odeslat ji.

Tyto úlohy plně pokrývají jednotlivé fáze životního cyklu zpracování požadavku v JavaServer Faces. Cílem je vývojáře odstínit od práce s HTTP protokolem. Obrázek 3 znázorňuje všechny fáze zmíněného cyklu (odstavec viz [5]).



Obrázek 3 : *Životní cyklus zpracování požadavku v JSF (Zdroj: [7])*

Každá komponenta z komponentového stromu v daném požadavku prochází všemi fázemi. Komponentový vývojář má tedy z komponenty přístup do každé z fází životního cyklu.

3.2.6 Event model

Netradičním specifikem JavaServer Faces ve světě webových frameworků je programový model založený na událostech[4] [4]. Tento přístup vychází z Java knihoven pro tvorbu grafických uživatelských rozhraní Swing a AWT. Motivací stojící za použití tohoto typu architektury byl předpoklad použití JSF při vývoji komplexních aplikací. Události jsou členěny na dvě základní kategorie – životního cyklu a aplikační. JSF nabízí celkem 16 událostí životního cyklu, které vznikají v jednotlivých fázích životního cyklu zpracování požadavku a 2 události aplikační [5]. Jednotlivé události lze zachycovat prostřednictvím tříd typu `javax.faces.event.FacesListener`.

3.2.7 Podpora technologie AJAX

Specifikace JavaServer Faces ve své první verzi nepodporovala technologii AJAX a tak vznikaly různé knihovny třetích stran, které AJAX v JSF zprostředkovaně umožňovaly. Příkladem je *Ajax4jsf* z roku 2005 z knihovny *RichFaces* [20]. Až v JSR 314 byla podpora této moderní technologie přidána. Komponenty od JavaServer Faces verze 2.0 mohou na základě různých událostí generovat částečné odpovědi pro klienta. Jejich součástí může být i JavaScript, jenž je po obdržení na klientu vyhodnocen [4].

3.3 Shrnutí

Na základě JSF API a komponentového modelu je možné vytvářet vlastní JSF komponenty. Od verze 2.0 JSF podporuje technologii AJAX, jejímž prostřednictvím lze realizovat JavaScript funkce v AJAX odpovědi. Komponenty tedy mohou generovat do odpovědi kód specifický pro C3.js. Současně obě technologie podporují práci s událostmi. Nebyl tedy identifikován žádný teoretický problém, jenž by bránil cílům této práce.

[4] Tzv. Event-based programming model

4 Analýza a návrh

Tato kapitola je orientována na analýzu a návrh frameworku. Na jejím začátku je krátce popsána metodika vývoje, následně provedena analýza C3.js a specifikovány požadavky. Část týkající se návrhu obsahuje návrh architektury a návrh API.

4.1 Metodika vývoje

Základním přístupem při realizaci cílů této práce je iterativní vývoj založený na prototypování. V průběhu vývoje tedy bude v jednotlivých iteracích docházet k vyvinutí neúplných částí software nazývaných prototypy. Jednotlivé iterace navazují na další iterace do doby, kdy produkt splňuje požadavky konkretizované v této kapitole.

4.2 Analýza C3.js

Analýza knihovny C3.js je založena na analýze syntaxe a struktury kódu používaného k vykreslení grafu a následně kódu pro modifikace částí grafu prostřednictvím C3.js API.

4.2.1 Analýza syntaxe a struktury kódu deklarace grafu

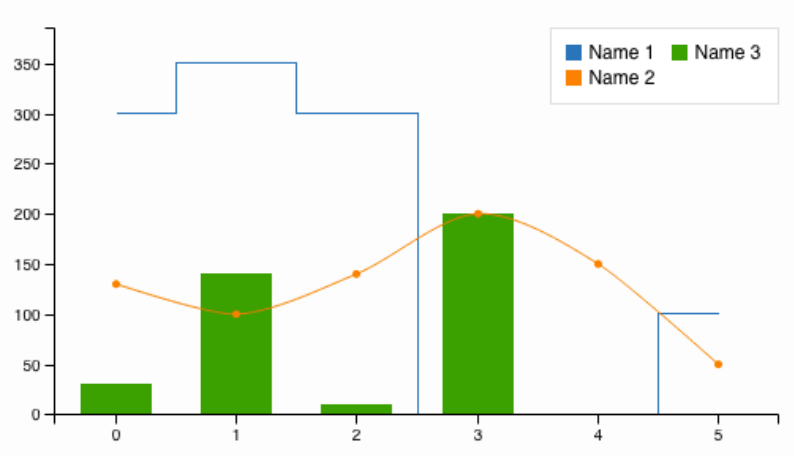
Jelikož komponenty budou pracovat nad C3.js skripty vykreslující grafy s odpovídající strukturou a syntaxí, je nutné začít s analýzou nad tímto základním stavebním kamenem. Výsledným produktem vizuálních komponent bude skript se strukturou odpovídající Výpisu 1 uvedenému níže.

Výpis 1: C3.js skript vytvářející graf se základními vlastnostmi (Zdroj: autor)

```
var chart = c3.generate({
    data: {
        columns: [
            ['data1', 300, 350, 300, 0, 0, 100],
            ['data2', 130, 100, 140, 200, 150, 50],
            ['data3', 30, 140, 10, 200]
        ],
        types: {
            data1: 'step',
            data2: 'spline',
            data3: 'bar'
        },
```

```
        names: {
            data1: 'Name 1',
            data2: 'Name 2',
            data3: 'Name 3'
        }
    },
    size: {
        height: 280,
        width: 500
    },
    legend: {
        position: 'inset',
        inset: {
            anchor: 'top-right',
            step: 2
        },
        item: {
            onclick: function (id) {
                console.log(id);
            }
        }
    }
});
```

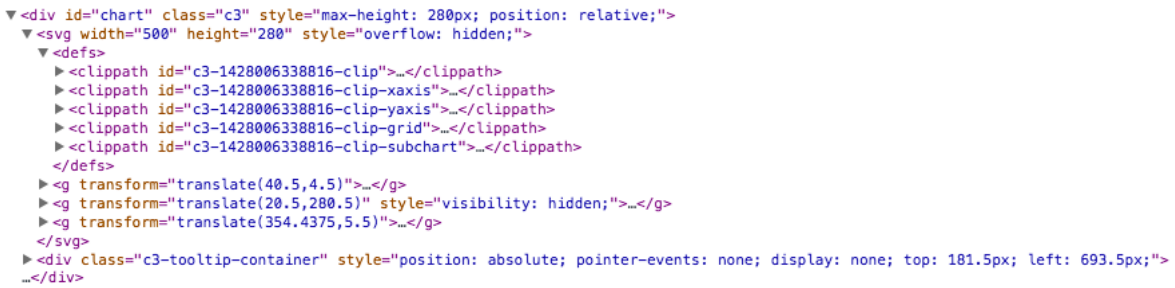
Skript z Výpisu 1 má za následek vykreslení grafu ve webovém prohlížeči odpovídající ilustraci na Obrázku 4 níže.



Obrázek 4 : *Vygenerovaný graf za pomoci kódu z Výpisu 1 (Zdroj: autor)*

Graf je v prohlížeči zobrazen díky úpravě objektového modelu dokumentu (DOM). Konkrétně byly vypsány elementy a podelementy ze specifikace SVG společně s hlavním

`div` elementem, což je možné ověřit v aktualizovaném zdrojovém kódu stránky v prohlížeči. Tato skutečnost je ilustrována Obrázkem 5.



Obrázek 5 : *Upravený zdrojový kód stránky odpovídající grafu na Obrázku 4 (Zdroj: autor)*

Základem této úpravy je fundamentální metoda knihovny C3.js, a sice metoda `generate()`. Tato metoda vrací odkaz na vygenerovaný graf, který lze uložit do libovolné proměnné. V uvedeném příkladu se jedná o proměnnou `chart`.

Z Výpisu 1 je dále patrné, že metoda `generate()` obsahuje celkem 3 bloky pro specifikaci atributů grafu, jmenovitě `data`, `size` a `legend`. Každý blok reprezentuje definici určité vlastnosti grafu. Těch nabízí knihovna celou řadu, zde je zahrnut pouze příklad. Všechny zmíněné bloky odpovídají zápisu dat typu JSON. Blok tedy může mít podobu objektu, jak zachycuje Výpis 2.

Výpis 2: Objektový typ bloku (Zdroj: autor)

```
legend: {
…
}
```

Může také obsahovat pouze pár název a hodnota, jak je patrné z Výpisu 3.

Výpis 3: Typ bloku název-hodnota (Zdroj: autor)

```
height: 280
```

Na příkladném Výpisu 1 si lze dále povšimnout JavaScript metody, jež se skýtá pod polem `item`. Dalším typem je tedy typ definující metodu, znázorněný Výpisem 4.

Výpis 4: Typ bloku definující metodu (Zdroj: autor)

```
onclick: function (id) {
                console.log(id);
            }
```

Posledním typem je typ definovaný jako pole hodnot, znázorněný Výpisem 5.

Výpis 5: Typ bloku pole (Zdroj: autor)

```
columns: [
            ['data1', 300, 350, 300, 0, 0, 100],
            ['data2', 130, 100, 140, 200, 150, 50],
            ['data3', 30, 140, 10, 200]
        ]
```

Ze struktury bloků `data` a `legend` lze odvodit, že některé bloky se skládají z bloků dalších (konkrétně `columns`, `names`, `types`, respektive `inset`). Je možné tedy říci, že bloky mohou být kompozitními. Blok `data` je však důležitý ještě z jiného důvodů. Obsahuje sadu dat, jež se uloží do paměti prohlížeče a následně vykreslí v grafu. Pokud datové sady nejsou poskytnuty, JavaScript skončí s chybou a požadavkem o její nápravu. Zároveň i `data` sama o sobě musí být deklarována, jinak program opět skončí chybou. Samotná struktura tohoto bloku se vyznačuje několika důležitými prvky, a sice že

- mohou být předány ve formátu sloupců, řádků, objektů typu JSON nebo prostřednictvím URL,
- jsou podporovány pouze číselné datové typy,
- lze předat v podstatě libovolný počet datových sad,
- hodnoty datových sad jsou vykresleny v deklarované posloupnosti,
- každá datová sada může mít libovolnou délku,
- každá datová sada má unikátní identifikátor.

Právě přes unikátní identifikátor se na datové sady uložené v paměti prohlížeče lze odkazovat a měnit tak jejich atributy. Vlastnosti datových sad se deklarují v rámci těla vlastnosti `data` a jedná se o zmiňované kompozitní bloky, jmenovitě `types` a `names`.

4.2.2 Analýza syntaxe a struktury kódu API

C3.js nabízí API, jehož prostřednictvím lze upravovat vlastnosti deklarované při prvním vygenerování grafu včetně načtených dat. Je-li graf upravován prostřednictvím tohoto API, jsou změny doprovázeny přechodovými animacemi. Skutečností však je, že nelze upravovat celou množinu vlastností nabízených při deklaraci grafu, nýbrž jejich podmnožinu. Skript znázorňující příklad volání tohoto API ilustruje Výpis 6 a bude dále rozebrán a okomentován.

Výpis 6: Volání API pro dynamickou změnu vlastností (Zdroj: autor)

```
chart.data.colors({
  data1: '#ff0000',
  data2: '#000000',
  data3: '#EFEFEF'
```

```
});
setTimeout(function () {
    chart.unload();
}, 500);
setTimeout(function () {
    chart.load({
        columns:[
            ['data4', 30, 20, 50, 40, 60, 50]
        ]
    });
}, 1000);
setTimeout(function () {
    chart.transform('spline', 'data4');
}, 1500);
```

Výpis 6 obsahuje volání sady metod nad C3.js API. Konkrétně se jedná o volání celkem 4 metod nad proměnnou s názvem `chart`, jenž byla definována při vytvoření grafu ve Výpisu 1. Po názvu proměnné následuje název metody a její tělo, které syntakticky odpovídá JSON notaci a tedy některému z typů bloků popsaných v kapitole 4.2.1.

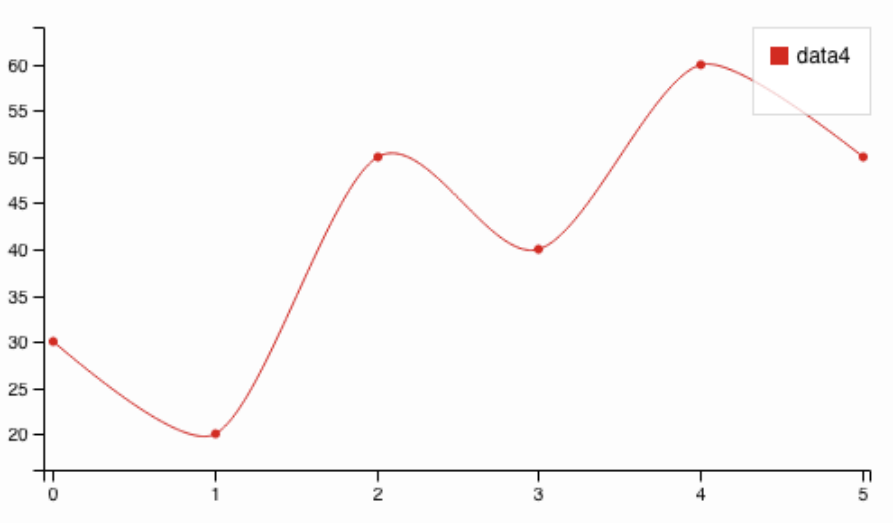
Poslední 3 metody jsou obaleny metodou `setTimeout(function, duration)`. Jelikož při volání API jsou změny nad SVG elementy prováděny postupně tak, aby byla docílena animace mezi přechody, musí být v případě dávkového volání metod nastaveny časové rozestupy mezi nimi. V opačném případě může jednu animaci přerušit animace jiná. Proměnná `duration` je v tomto případě nastavena s rozestupy po 500 ms z toho důvodu, že výchozí nastavení délky animací v C3.js je 500 ms. Metoda `chart.data.colors()` není obalena, jelikož neprovádí žádnou animaci a ve výsledku by byla viditelná pouze pauza mezi přechody, což není uživatelsky přívětivé.

Paměť prohlížeče má k dispozici unikátní identifikátory datových sad, kterým jsou měněny jejich vizuální vlastnosti. Vzhledem k této skutečnosti Výpis 7 obsahuje legitimní změnu atributů nad datovými sadami `data1`, `data2` a `data3`.

Výpis 7: Změna vizuálních vlastností již načtených dat pomocí identifikátorů (Zdroj: autor)

```
chart.data.colors({
  data1: '#ff0000',
  data2: '#000000',
  data3: '#EFEFEF'
});
```

Nakonec Obrázek 6 znázorňuje vizuální výsledek po dávkovém volání modifikačních metod nad grafem z Obrázku 4.



Obrázek 6 : *Upravený graf z Obrázku 4 za pomoci C3.js API (Zdroj: autor)*

4.3 Specifikace požadavků

Následují konkrétní požadavky, jejichž splnění povede k naplnění obou cílů stanovených touto prací. Jedná se tedy o požadavky na to, co má budoucí framework poskytovat a o požadavky na komponentovou sadu, jež bude součástí distribuce. Jelikož bude sada implementována prostřednictvím vzniklého API, požadavky na sebe navazují.

Seznam požadavků na vyvinutý framework vyplývající z předešlých kapitol shrnuje Tabulka 3.

Tabulka 3: *Seznam požadavků na framework (Zdroj: autor)*

ID požadavku	Obsah požadavku
R01	vlastnosti grafu bude možné definovat ve Faceletu
R02	komponenty budou podporovat AJAX požadavky
R03	změny komponent na serverové straně při AJAX požadavku budou vykresleny na klientu prostřednictvím C3.js API
R04	bude existovat podpora pro přidání nové vlastnosti grafu, a to bez nutnosti zásahu do kódu existujících komponent
R05	bude existovat podpora pro přidání nového typu grafu
R06	bude-li přidán objekt reprezentující vizuální vlastnost, bude podporována implementace komponenty pro jeho deklaraci ve

ID požadavku	Obsah požadavku
	Faceletu
R07	podpora implementace serverové reprezentace nových C3.js API vlastností
R08	selekce na grafu bude mít odraz i na serverové straně (včetně podpory AJAX)
R09	změní-li se objekt reprezentující vizualizovaná data, graf se bezchybně překreslí
R10	podpora programatického přístupu k vlastnostem grafu, čímž budou moci být měněny na straně serveru
R11	rámec bude poskytovat dostatečnou úroveň zapouzdření skriptů tak, aby Java programátor s JavaScriptem nemusel přijít do styku
R12	API bude odpovědné za správu externích zdrojů nutných pro C3.js (CSS a JavaScript soubory)

Souhrn požadavků na komponentovou sadu je obsažena v Tabulce 4.

Tabulka 4: *Seznam požadavků na komponentovou sadu (Zdroj: autor)*

ID požadavku	Obsah požadavku
R13	komponentová sada bude implementována ve vzniklém frameworku
R14	knihovna bude obsahovat komponentovou sadu pokrývající všechny typy C3.js grafů současné verze
R15	knihovna bude obsahovat implementaci základní sady komponent a objektů pro definici vlastností grafů
R16	knihovna bude obsahovat implementaci základní sady objektů pro dynamické modifikace grafů

4.4 Analýza požadavků

Definované požadavky jsou popsány detailněji v této podkapitole.

Aby bylo maximálně využito deklarativního přístupu, ve Facelet stránce kromě deklarace grafů bude možné deklarovat jejich vizuální vlastnosti (R01). Komponenty musí

umožňovat aktivaci AJAX typickou pro JSF a tedy zpracovávat požadavky tohoto typu (R02). Jelikož mají být vizualizace vytvořené touto prací na pokročilé úrovni, framework bude využívat C3.js API ke komplexním přechodovým animacím (R03). C3.js je stále v aktivním vývoji, framework by měl podporovat proces přidání nových typů grafů (R05). Stejně tak by měl umožnit jednoduše přidávat nové vlastnosti a to i již existujícím grafům bez toho, aby bylo nutné zasahovat do jejich stávajícího kódu (R04). Aby takto vzniklá vlastnost byla použitelná ve Facelet stránce, framework bude podporovat jednoduché vytvoření odpovídající komponenty (R06).

Jak dále vyplývá z analýzy kódu v předešlé kapitole, pouze podmnožina vlastností grafů má svůj odraz v C3.js API. Aby bylo umožněno budoucí přidání funkcionality API, bude framework podporovat tvorbu objektů reprezentujících tyto funkčnosti založené na serverových objektech (R07). Bude-li na grafu vybrána datová sada, bude mít tento výběr odezvu i na serveru. Bude-li žádoucí bezprostřední reakce provedené selekce, bude možné aktivovat AJAX pro promptní odeslání požadavku a dat na server (R08). Změní-li se kompletní datový zdroj prostřednictvím AJAX požadavku, graf se bezchybně překreslí (R09). Jelikož při definici vlastností ve Facelet stránce nelze specifikovat dynamické chování grafu, bude umožněn programatický přístup k přiřazeným vlastnostem. Jeho prostřednictvím se pak na straně serveru budou moci upravit atributy vlastností, či přidávat vlastnosti kompletně nové. Tato modifikace bude následně promítnuta v přechodových animacích v prohlížeči po obdržení odpovědi ze serveru (R03).

Autor stránek Facelet bude odstíněn od psaní skriptů v jazyce JavaScript, jež budou zapouzdřeny v jednotlivých Facelet elementech a v případě programatického přístupu v Java objektech (R11). Zároveň komponentový vývojář používající toto API nebude muset přidávat závislosti na soubory typu JavaScript a CSS nutné pro funkci C3.js (R12).

Komponentová sada bude implementována ve vzniklém frameworku, čímž mimo jiné bude ozkoušena jeho použitelnost (R13). C3.js ve verzi použité pro účely této práce nabízí celkem 11 typů grafů[5]. Požadavkem je implementace všech typů grafů do vzniklé knihovny (R14). Zároveň bude obsahovat základní sady komponent a objektů pro deklaraci vizuálních vlastností grafů a také objektů pro jejich dynamické modifikace (R15, R16). Jelikož se nebude jednat o kompletní implementaci všech vlastností a modifikací poskytovaných C3.js, součástí této práce je příručka zahrnující návod pro implementaci dalších typů těchto objektů a komponent.

4.4.1 Způsob realizace požadavků v JSF

Komponentový model definovaný ve specifikaci JavaServer Faces obsahuje výchozí sadu implementovaných komponent. Lze tedy zdědit vlastnosti a chování relevantní

[5] Kompletní seznam systémových požadavků a verzí obsahuje Příloha A

komponenty a následně překrýt požadované odlišnosti. Tímto způsobem lze efektivně vytvářet nové komponenty a bude tak postupováno při realizaci požadavků R01 a R05.

Přechodové animace z požadavku R03 budou realizovány prostřednictvím volání JavaScriptových funkcí nad vytvořenými grafy. Tohoto bude dosaženo prostřednictvím AJAX podpory ze strany JSF, konkrétně pomocí metod poskytovaných třídou `javax.faces.context.PartialResponseWriter`. V AJAX odpovědi tak budou zabaleny potřebné skripty pro C3.js, které se na klientu po přijetí vyhodnotí.

Pokud vývojář definuje v těle komponenty tag pro aktivaci podpory AJAX, bude po selekci datové sady odeslán požadavek na server, jenž povede k realizaci životního cyklu zpracování požadavku. Součástí tohoto požadavku bude i hodnota zvolené datové sady, která bude následně zkonvertována na odpovídající datový typ. Tyto možnosti zpřístupňuje rozhraní `javax.faces.component.EditableValueHolder`, jenž je využito k naplnění požadavku R08.

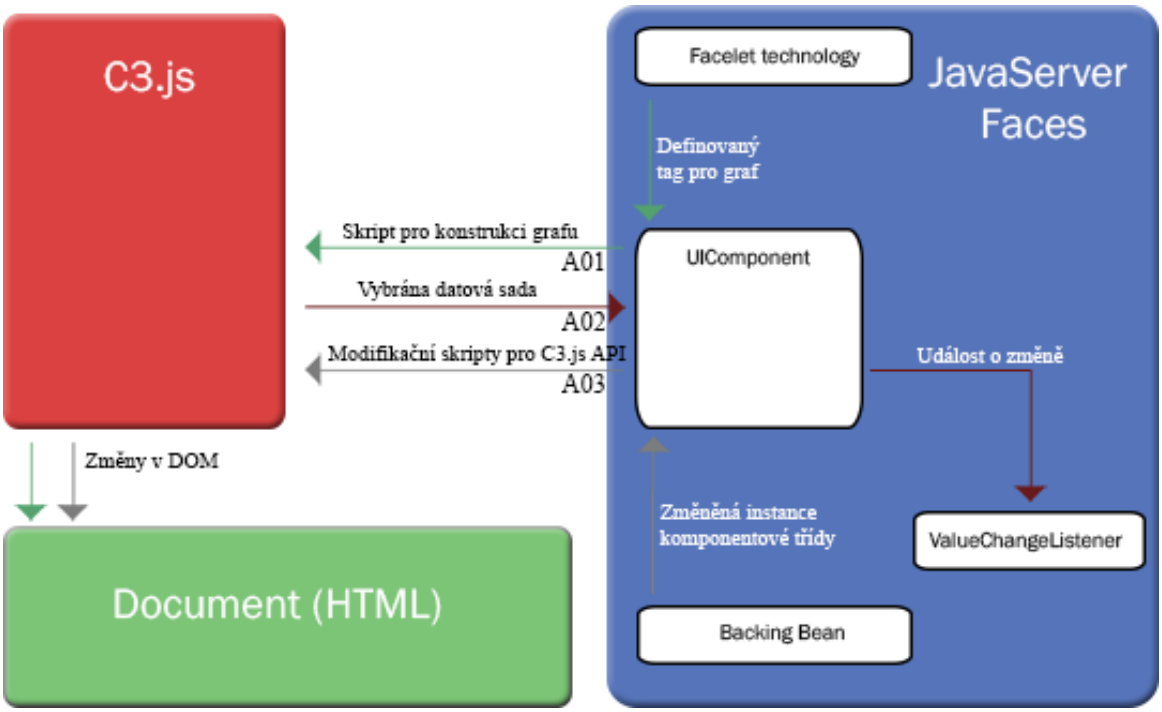
Instance komponentových tříd mají rozsah platnosti pouze jednoho HTTP požadavku. Jelikož bude nutné znát změny datového zdroje pro požadavek R09 déle, než je životnost instance komponentové třídy, bude využito řízení rozsahu platností objektů poskytované frameworkem JSF. Konkrétně bude použit objekt typu `javax.faces.component.StateHelper`, jenž slouží k udržení stavu objektů napříč AJAX požadavky.

Objekty tříd reprezentující vizuální vlastnosti grafů budou vázány k jednotlivým instancím komponentových tříd. Z toho vyplývá, že programátor bude mít přístup k těmto instancím, jinak by nebylo možné naplnit požadavek R10 o programatickém přístupu k vlastnostem grafu. Získání reference na objekty typu `javax.faces.component.UIComponent` z kontextu JSF je realizovatelné různými způsoby. Některé z nich budou použity v kapitole obsahující uživatelskou příručku.

4.5 Návrh architektury

Následující kapitola se zabývá návrhem architektury vyvíjeného programu.

Pro účely naplnění definovaných požadavků byl sestaven návrh zachycující JavaServer Faces, C3.js a dokument. Obrázek 7 znázorňuje jednotlivé komunikační toky, jež jsou zároveň tématy následujících podkapitol. Ty, jež na sebe navazují, jsou zobrazeny stejnou barvou. Záměrně byla vynechána HTTP komunikace mezi C3.js a JSF reprezentovaná šipkami A01, A02 a A03, které budou detailněji rozebrány v kapitole 5.6.



Obrázek 7 : *Architektura a komunikace mezi komponentami vyvíjeného programu (Zdroj: autor)*

4.5.1 Deklarace grafu

Vztah znázorněný na Obrázku 7 zelenou barvou představuje proces deklarace grafu. Začíná formulací příslušných tagů ve Facelet stránce. Společně se základními atributy grafu bude mít vývojář k dispozici paletu vizuálních vlastností reprezentovanými samostatnými komponentami. Bude-li tedy žádoucí konfigurovat určitou vizuální vlastnost grafu, přidá vývojář do těla tagu deklarující daný graf odpovídající komponentu. Příkladný kód je obsažen ve Výpisu 8.

Výpis 8: Příklad deklarace grafu se skrytou legendou (Zdroj: autor)

```
<c3:pie data="#{backingBean.data}">
    <c3:legend show="false" />
</c3:pie>
```

Toto řešení je výhodné, jelikož

- komponenty deklarující vizuální vlastnosti jsou nezávislé na typu grafu,
- objektu reprezentující graf lze dynamicky přidávat a odebírat tyto vlastnosti,
- je možné implementovat nové komponenty vizuálních vlastností a přidávat je stávajícím typům grafu bez nutnosti do zásahu existujícího kódu grafů,

- prostřednictvím kompozitních komponent a shlukováním komponent vizuálních vlastností lze vytvářet přepoužitelné grafy se specifickými vizuálními vlastnostmi bez nutnosti psaní Java kódu.

Následně kompetentní komponenta znázorněna jako *UIComponent* požádá příslušné objekty o vygenerování korespondujícího kódu a odešle jej klientovi. C3.js provede adekvátní JavaScript operace, jež mají za následek změnu DOM daného dokumentu. Výsledkem je vykreslený graf.

4.5.2 Selekce datových sad a události o změnách

Tok znázorněn vínovou barvou charakterizuje výběr provedený na grafu uživatelem. Jedná se o výběr jedné z komponentových sad, jenž jsou na grafu vykresleny. Tato událost bude mít svou reakci na serverové straně. Jelikož jsou datové sady reprezentovány na serveru Java objekty, třída typu `UIComponent` bude odpovědná za konverzi selekce na objekt náležitého typu.

JavaServer Faces na základě *Event Modelu* poskytuje kapacity pro definici posluchačů změn hodnoty provedených na komponentě. Pokud tedy dojde k selekci na grafu, dojde následně i k notifikaci zaregistrovaných posluchačů o změně hodnoty komponenty. Posluchači zároveň budou schopni získat jak hodnotu novou, tak hodnotu původní. Tato skutečnost je vyznačena na Obrázku 7 pokračováním toku k objektu nazvaným `ValueChangeListener`.

4.5.3 Modifikace vizuálních vlastností

Jelikož vytvořené komponenty budou podporovat AJAX požadavky, je nutné počítat i s částečnými modifikacemi vizuálních vlastností grafu. Jedná se o stav, kdy nemá být překreslen celý graf, ale pouze jeho část odpovídající provedené serverové změně. Dojde-li tedy na serveru k úpravě vizuálních vlastností nebo dat již vykresleného grafu, je nezbytné na základě informací o provedených změnách vytvořit modifikační skripty odpovídající specifikaci C3.js API. O vygenerování skriptů požádá třída typu `UIComponent`, jenž jej zároveň ve vhodný okamžik odešle klientovi zabalený v AJAX odpovědi.

Typicky dochází ke změně vizuální komponenty v objektech tříd typu Backing Bean, jenž slouží jako Controller pro stránku (*view*) obsahující danou komponentu. Je zároveň nutno počítat s tím, že za dobu tvorby odpovědi na HTTP požadavek může dojít na serveru k celé řadě těchto modifikací. Skript tedy může být reprezentován dávkovým voláním JavaScript funkcí.

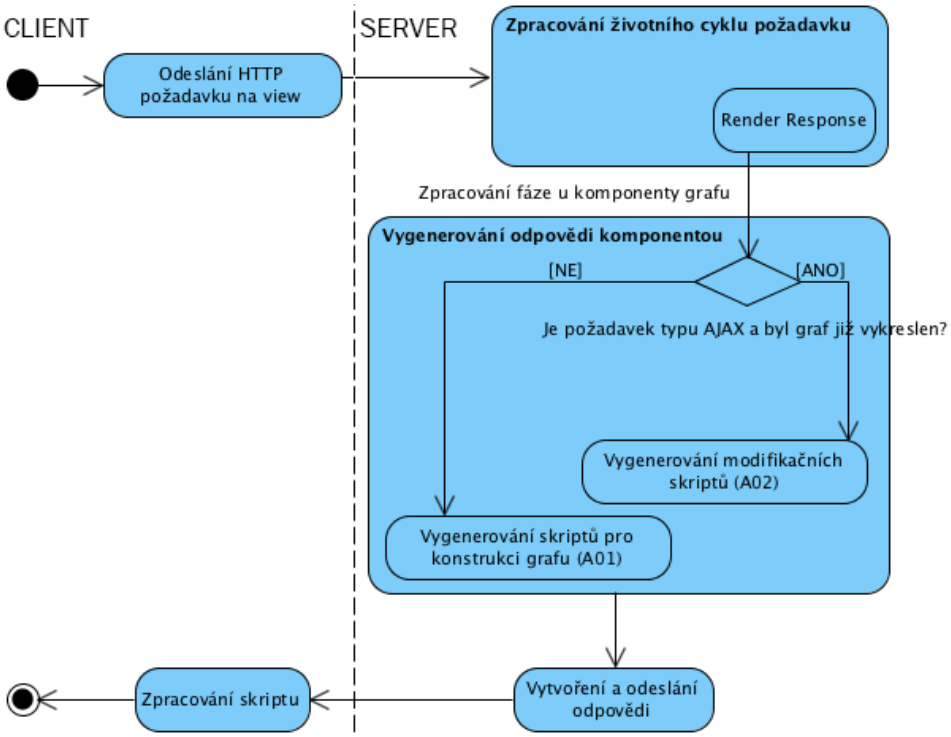
4.6 Návrh API

Tato kapitola je zaměřena na návrh API. Jedná se o popis rozhraní mezi C3.js a JSF, respektive o popis kontraktu komunikace mezi nimi. Je soustředěno opět na vztah identifikovaný v kapitole 4.5 a znázorněný na Obrázku 7. Tentokrát je však podrobněji popsán po technologické stránce.

4.6.1 Skript pro konstrukci grafu (A01) a modifikace grafu (A03)

Z provedené analýzy vyplývá, že generovaný skript je dvou typů. Skript, jenž slouží pro první vykreslení grafu a skript, který slouží pro dynamické modifikace již vykresleného grafu. Za poskytování obou typů kódu bude odpovědná implementace třídy `UIComponent`. Je však potřebné definovat, kdy a jak k těmto operacím bude docházet.

V momentě, kdy je přijat HTTP požadavek, JSF postupně realizuje jednotlivé fáze životního cyklu zpracování požadavku nad všemi komponentami daného *view*. V momentě, kdy dojde k fázi *Render Response* je komponenta prostřednictvím metod `encodeAll()`, `encodeBegin()`, `encodeChildren()` a `encodeEnd()` požádána o poskytnutí kódu. Jelikož bude v případě AJAX požadavků podporováno dynamických modifikací, při kterých nebude docházet ke kompletnímu překreslení grafu, ale pouze jeho části, je v tomto kroku nutné zjistit, zda se jedná o typ požadavku AJAX a zda již byl graf vykreslen. Tento odstavec znázorňuje diagram aktivit na Obrázku 8 níže.



Obrázek 8 : *Diagram aktivit pro generování skriptu konstrukce a modifikací grafu (Zdroj: autor)*

Skript pro konstrukci grafu (A01)

Pokud doposud graf nebyl vykreslen, dochází k procesu A01. Implementace třídy `UIComponent`, jenž je odpovědná za vykreslení grafu, požádá svou kolekci objektů vizuálních vlastností o vygenerování skriptu. Následně prostřednictvím odpovídajících metod třídy `javax.faces.context.ResponseWriter` takto nahromaděné skripty zapíše do konstruované odpovědi pro klienta.

V případě, kdy se jedná o první vykreslení grafu, je skript vypsán přímo do zdrojového kódu stránky. Odpovídající část zdrojového kódu webové stránky tak bude obsahovat skript odpovídající struktuře kódu ve Výpisu 9.

Výpis 9: Příklad vypsaného skriptu pro konstrukci grafu ve zdrojovém kódu stránky (Zdroj: autor)

```
<script>
var chart = c3.generate({
    data: {
        columns: [
            ['data1', 30, 200, 100, 400, 150, 250],
            ['data2', 20, 180, 240, 100, 190]
        ]
    }
});
</script>
```

C3.js následně provede korespondující operace a vykreslí graf na stránce.

Modifikační skripty pro C3.js API (A03)

Je-li vyhodnoceno, že požadavek je typu AJAX a graf již byl vykreslen, dochází k procesu A02, tedy konstrukce modifikačních skriptů. Pro změnu jednoho vizuálního atributu grafu odpovídá jedno JavaScript volání. V momentě generování odpovědi ve fázi *Render Response* je tedy nutné požádat odlišné objekty o vygenerování skriptů než tomu bylo u procesu A01. Jedná se o objekty, jež poslouchají změny provedené na serveru u objektů vizuálních vlastností od doby přijetí požadavku až po dobu generování onoho skriptu. Na základě sady odposlechnutých změn tyto objekty vygenerují kód pro modifikace na grafu.

Jak již bylo zmíněno, je pravděpodobné, že nastalých změn bude více. Odpověď tedy může bude obsahovat dávku JavaScript funkcí. Skript bude realizován v AJAX odpovědi v části *eval*, kam zápis umožňují metody `startEval()`, `endEval()` a `write()` třídy `javax.faces.context.PartialResponseWriter`. Příklad odpovědi tohoto typu je obsažen ve Výpisu 10.

Výpis 10: Příklad odeslané AJAX odpovědi obsahující modifikační skripty pro C3.js API (Zdroj: autor)

```
<?xml version='1.0' encoding='UTF-8'?>
<partial-response id="j_id1">
   <changes>
      ...
      <eval><![CDATA[
                setTimeout(function () {
                   j_idt46$pie.load({columns:
                      [['ihTOyhYXYVTnwqpW', 150, 250, 350, 50, 50],
                      ['jassJqWeyVKKkDRV', 50, 50, 50, 50, 50]]})}, 0);
                j_idt46$pie.data.names({jassJqWeyVKKkDRV: 'NEW NAME',
                   ihTOyhYXYVTnwqpW: 'NEW NAME 2'});
                j_idt46$pie.data.colors({
                   ihTOyhYXYVTnwqpW: '#16525E',
                   jassJqWeyVKKkDRV: '#16525E'});
                setTimeout(function () {
                   j_idt46$pie.transform('step')}, 500);
                ]]>
      </eval>
      ...
   </changes>
</partial-response>
```

4.6.2 Vybrána datová sada (A02)

Bude-li graf v momentě odeslání požadavku evidovat selekci, je nezbytné, aby tato informace byla součástí požadavku. Odesílaná formulářová data tedy musí obsahovat hodnotu reprezentující vybranou datovou sadu. Ta následně bude přiřazena komponentě a konvertována na odpovídající datový typ.

Z analýzy vyplývá, že C3.js identifikuje jednotlivé datové sady prostřednictvím unikátních řetězců. Komponenta na serverové straně tedy bude v požadavku očekávat právě tento řetězec. Následně požádá objekt vlastnící reference na instance datových sad o tu, která odpovídá danému identifikátoru. Tím proběhne konverze na příslušný datový typ.

Aby byl řetězec obsahující identifikátor datové sady součástí odesílaných formulářových dat, bude pro každý graf automaticky generováno skryté pole. Při provedené selekci na grafu se zavolá JavaScript funkce, jenž nastaví tomuto poli hodnotu atributu `value` na identifikační řetězec zvolené datové sady. Aby byl název pole unikátní, bude použit identifikátor komponenty s příponou. Kompetentní pro tvorbu pole včetně jeho názvu tedy bude komponentová instance.

Komponenty vlastnící editovatelné hodnoty musí dle JSF specifikace implementovat rozhraní `javax.faces.component.EditableValueHolder` požadující poměrně rozsáhlý počet

metod. Na základě toho bude děděno ze třídy `javax.faces.component.UIInput`, jenž zmíněné rozhraní implementuje. Následně pak bude překryto jen potřebné chování.

Ve fázi *Apply Request Values* bude hodnota ze skrytého pole přiřazena komponentě prostřednictvím metody `setSubmittedValue()` z rozhraní `javax.faces.component.EditableValueHolder`. Toho bude docíleno prostřednictvím mapy HTTP parametrů, které poskytuje `javax.faces.context.ExternalContext` a jeho metoda `getRequestParameterMap()`. Komponenta tedy ve formulářových datech POST požadavku bude očekávat hodnotu podobnou Výpisu 11 níže. V metodě `getConvertedValue()` z téhož rozhraní bude hodnota přeložena na datový typ reprezentující datovou sadu.

Výpis 11: Hodnota skrytého pole obsahující unikátní identifikátor datové sady (Zdroj: autor)

```
j_idt46:chartIndex:dtYgUHduHdMWrzUj
```

JSF se následně postará o vytvoření události o změně prostřednictvím třídy `javax.faces.event.ValueChangeListener` a zároveň poskytne v tomto rozhraní objekt původní (jenž byl součástí posledního požadavku) a nově zkonvertovaný.

Celý proces A02 tedy může být pouze jednosměrný, respektive nemusí generovat žádnou odpověď pro klienta.

4.7 Implementace API

Implementace API vychází z návrhu obsaženého v předchozí kapitole. Realizovaný framework s názvem C3Faces je dostupný na adrese *http://www.github.com/martin-linha/c3faces*.

4.7.1 Implementace API pro tvorbu komponent

Základními třídami API pro tvorbu komponent jsou abstraktní třídy `C3Chart` a `C3Property`. Následuje jejich podrobný popis.

Třída C3Chart

Podpora tvorby nových komponent pro nové typy grafů je předmětem požadavku R05.

Základem pro komponenty vykreslující C3.js grafy je abstraktní třída nazvaná `C3Chart`. Jak je zřejmé z návrhu v kapitole 4.6.2, třída bude dědit z `javax.faces.component.UIInput`. Poněvadž je dále dle požadavku R02 žádoucí, aby komponenty umožňovaly aktivaci podpory AJAX, je dle JSF specifikace nezbytná implementace rozhraní `javax.faces.component.behavior.ClientBehaviorHolder`.

Každý graf je popsán sadou vizuálních vlastností. `C3Chart` proto vlastní kolekci objektů reprezentující tyto vlastnosti. Jedná se o objekty typu `Property`, jež generují odpovídající C3.js skript a podrobněji budou popsány v následující kapitole. Poněvadž se jedná o dynamickou kolekci vlastností a nikoliv o statický výčet, je možné je bez nutnosti zásahu do existujícího kódu přidávat a odebírat. Dle analýzy kódu C3.js vyplývá, že definice grafu musí vždy obsahovat blok nazvaný *data*. Objekty typu `C3Chart` a odpovídající tag ve Facelet stránce tedy budou vyžadovat objekt představující tento blok.

Hlavní službou rozebírané třídy je implementace životního cyklu zpracování požadavku. Jedná se o metody, při kterých jsou zpracovány parametry z HTTP požadavku a metody volané při generování odpovědi. Konkrétně se jedná o

- `decode()`,
- `encodeBegin()`,
- `encodeEnd()`.

Metoda `decode()` zjistí, zda komponenta má ve Facelet stránce přiřazen tag <f:ajax /> a případně provede příslušné operace.

V metodě `encodeEnd()` jsou požádány odpovídající objekty o vygenerování skriptu, jenž je následně zapsán do konstruované odpovědi.

K definici typu vykresleného grafu slouží metoda `getChartType()` s modifikátorem přístupu `protected`.

Třída `ComponentProperties` vlastní kolekci objektů vizuálních vlastností dané komponentové třídy grafu. Aby předané vizuální vlastnosti byly k dispozici nad rámec jednoho AJAX požadavku a měly tedy odpovídající rozsah platnosti, `C3Chart` se pro aktuální instanci dotazuje objektu typu `javax.faces.component.StateHelper`, jenž má rozsah platnosti napříč AJAX požadavky.

`C3Chart` dále vyžaduje implementaci abstraktní třídy `getDefaultProperties()` vracející kolekci objektů typu `Property`. Všechny její záznamy budou přidány do kolekce zapouzdřené v objektu `ComponentProperties`. Implementace této abstraktní třídy tedy může prostřednictvím této metody přiřadit vizuální vlastnosti, jež bude mít graf ve svém výchozím nastavení. Tato skutečnost je vázána k požadavku R05.

Dle návrhu třída překrývá metodu `getConvertedValue(FacesContext context, Object newSubmittedValue)` vracející konvertovaný objekt. Konverze je prováděna na základě získaného unikátního identifikátoru ze skrytého pole.

Aby bylo možné naplnit požadavek R12, musí mít tato třída na starosti přidání všech externích souborů na CSS styly a JavaScript soubory. Toho bylo docíleno prostřednictvím

anotací `@ResourceDependencies` a `@ResourceDependency` z balíčku `javax.faces.application`. Konkrétně zde byly specifikovány cesty pro *d3.js*, *c3.js*, *c3.css* a *c3faces.js*[6].

Třída C3Property

Předmětem požadavku R06 je podpora implementace komponent pro specifikaci vizuálních vlastností grafu ze strany API. Komponenty bude možné vkládat do těla komponent vykreslující grafy ve Facelet stránce, čímž bude splněn požadavek R01.

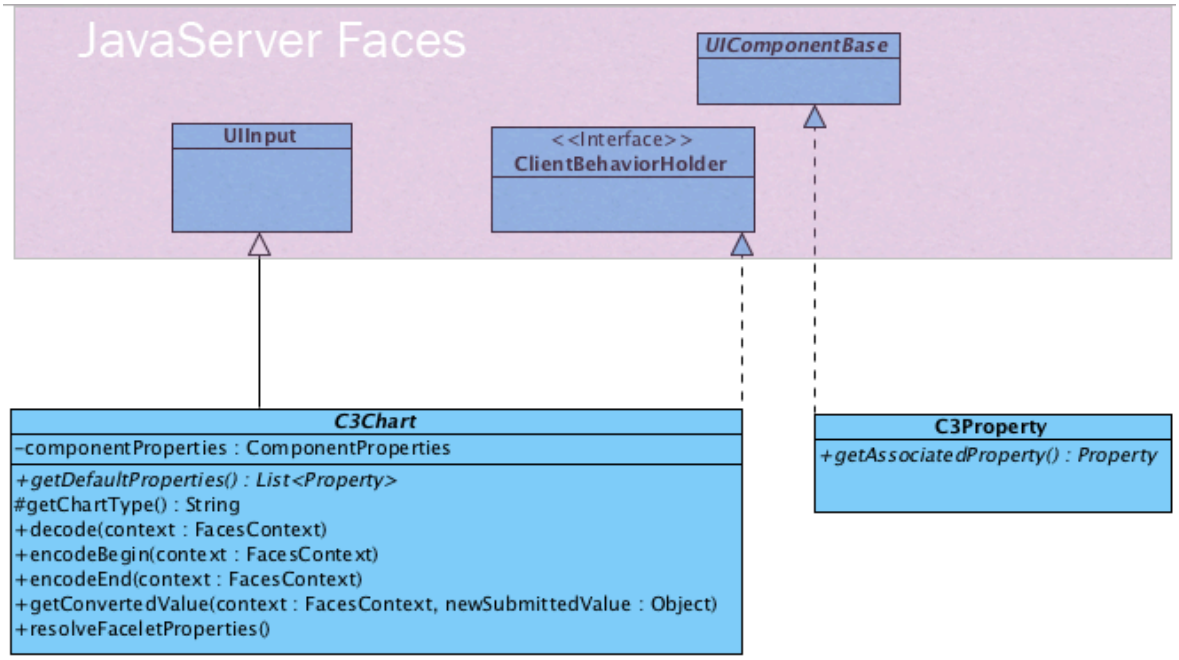
Pro tyto účely byla navržena třída `C3Property`. Jedná se o jednoduchou abstraktní třídu s jednou abstraktní metodou nazvanou `getAssociatedProperty()`, která nepotřebuje dědit žádné speciální chování JSF komponent z komponentového modelu. Vystačí si tak se základní implementací `javax.faces.component.UIComponent` poskytované abstraktní třídou `javax.faces.component.UIComponentBase`.

Jelikož má tag definovaný ve Facelet stránce odpovídat jedné vizuální vlastnosti grafu, vrací metoda `getAssociatedProperty()` právě jeden objekt typu `Property`. Vývojář využívající API tento objekt naplní hodnotami získanými z Facelet stránky prostřednictvím mapy atributů dostupné ze zděděné metody `getAttributes()`.

Instance třídy `C3Chart` pak jen projde komponenty, jež má obsaženy v těle svého tagu. Bude-li se jednat o instanci třídy `C3Property`, převezme prostřednictvím její abstraktní metody objekt typu `Property` a přidá jej do své kolekce vlastností v objektu `ComponentProperties`. Tuto operaci má na starost metoda `resolveFaceletProperties()`.

Návrh této části API s výše identifikovanými metodami a atributy znázorňuje Obrázek 9, jehož součástí je zároveň i zachycená závislost na JSF.

[6] Použité verze lze nalézt v Příloze A



Obrázek 9 : *API pro tvorbu JSF komponent (Zdroj: autor)*

4.7.2 Implementace API pro tvorbu vizuálních vlastností a jejich modifikací

Vizuální vlastnosti grafů jsou reprezentovány objekty, jejichž hlavní operací je sestavování skriptů respektující syntaxi C3.js a tedy typy bloků z kapitoly 4.2.1. Jelikož je stejným způsobem přistoupeno i k objektům generující modifikační skript, je tímto docíleno zapouzdření JavaScript kódu z požadavku R11.

API tvorby objektů vizuálních vlastností

V analýze kódu C3.js bylo zjištěno, že kód je strukturován do specifických bloků, jež jsou složené z názvu a těla. Každý blok navíc může ve svém těle obsahovat další bloky. Na základě této skutečnosti lze považovat každý blok za kompozitní objekt. Pro řešení tohoto typu návrhového problému se zabývá vzor *Composite*, jehož principů bude při návrhu využito. Poněvadž každý blok reprezentuje určitou vlastnost grafu, je primární třída pro tyto objekty pojmenována `Property`. Jedná se o abstraktní třídu, jež bude implementovat základní algoritmus pro generování JavaScript kódu pro C3.js.

Současně bylo v analýze rozpoznáno, že jednotlivé bloky se v jistých odchylkách liší. Byla tedy navržena skupina podtypů třídy `Property`, jenž mají za cíl zachytit zmíněné odchylky. Lze říci, že se jedná o šablony pro jednotlivé typy skriptů zachycující JSON notaci. Hlavní odchylkou je znak na začátku a na konci těla vlastnosti. Třída `Property` tedy vyžaduje

implementaci metod `getPrefix()` a `getSuffix()`, které budou sloužit potomkům pro určení obalujících znaků.

Mezi názvem vlastnosti a tělem vlastnosti se nalézá další znak, jenž může měnit svou podobu. Jestliže bude chtít potomek změnit tento separátor, může překrýt metodu `getSeparator()`. Jelikož se spíše jedná o výjimečnou situaci, nebude vyžadována od abstraktní třídy implementace a jako výchozí hodnotu je nastaven znak dvojtečky.

Názvy tříd, vzniklé jako šablony bloků generovaných skriptů odpovídající JSON notaci společně s ukázkou vygenerovaného kódu zachycuje Tabulka 5 níže.

Tabulka 5: *Názvy tříd pro šablony generovaných skriptů a příklady užití (Zdroj: autor)*

Název třídy	**Příklad generovaného skriptu**
ObjectBlock	name: { body }
MethodBlock	name: function (){ body }
ArrayBlock	name: [body]
ValueBlock	name: body
CommaValueBlock	name, body

K implementaci zmíněného návrhového vzoru *Composite* byla přidána třídě `Property` kolekce objektů typu `Property`. Následně například metoda `getScript()` iteruje přes všechny záznamy kolekce, na kterých zavolá stejnou metodu. Tím jsou navštíveny všechny uzle a tedy vygenerován skript i kompozitních objektů.

Objekty modifikací vizuálních vlastností

Jak vyplývá z analýzy, C3.js poskytuje JavaScript API pro částečnou úpravu již vytvořených grafů. Podpora tohoto API ze strany serveru a implementace příslušných objektů je předmětem požadavku R03. Požadavek R07 žádá, aby framework podporoval tvorbu nových objektů tohoto typu.

Jelikož se jedná o modifikace grafu, realizovaná třída má název `Modifier`. Jedná se o podobnou třídu třídě `Property`. Generuje skript prostřednictvím metody `getScript()`, avšak se strukturou dle specifikace C3.js API. Stejně tak je tato třída kompozitní.

Jak dále vyplývá z analýzy, animace modifikační změny může trvat specifickou dobu. Tuto skutečnost vystihuje atribut `duration` a metoda `totalDuration()`. Ta slouží k vrácení součtu trvání animace všech kompozitních objektů, jež objekt obsahuje. Jelikož existují i takové objekty, jejichž modifikace neobsahují žádné animace a tedy nepotřebují zaznamenat dobu trvání, může každý objekt typu `Modifier` patřičně nastavit atribut `timeoutable`.

Modifikační skript má obsahovat změny, jež mají být vykresleny na grafu. Jeho obsah má reflektovat změny událé na serverové straně na objektech vizuálních vlastností, tedy na objektech typu `Property`, a na datových sadách vykreslených v grafu. Jak z analýzy vyplývá, nejedná se o změny na všech vizuálních vlastnostech, pouze na jejich podmnožině. Řešení tohoto problému je založeno na návrhovém vzoru *Observer*. Základ tvoří vydavatelé změn a pozorovatelé, jenž jsou při nastalé změně vydavatelem notifikováni.

Konkrétně tedy bylo vytvořeno rozhraní nazvané `ChangeListener` vlastnící jedinou abstraktní metodu předávající objekt typu `Change`. Její podoba odpovídá `onChange(Change<?> change)`. Objekty typu `Change` reprezentují název vzniklé změny a hodnotu provedené změny. Aby bylo možné sledovat historii provedených změn nad jedním objektem tohoto typu, vlastní kolekci dříve provedených změn. Do této kolekce jsou záznamy přidávány pouze v případě, je-li objekt třídy `Change` označen jako kumulativní. To vystihuje atribut `cumulatible`. Současně vznikla třída `CumulatibleChange` dědící chování z `Change`, která prostřednictvím překrytí metody zděděné `isCumulatible()` nastavuje své instance jako kumulativní. Aby bylo možné odlišit změny provedené nikoliv nad objekty typu `Property`, ale nad datovými sadami, vznikly obdobným způsobem třídy `ViewDataSetChange` a `ViewDataSetCumulatibleChange`.

Základní implementaci rozhraní `ChangeListener` poskytuje abstraktní třída `Changes`. Jejím hlavním cílem je správa mapy všech provedených změn. Pokud tedy vydavatel upozorní třídu `Changes` na změnu prostřednictvím příslušné metody, třída uloží do zmíněné mapy název provedené změny jako klíč a celý objekt změny jako hodnotu.

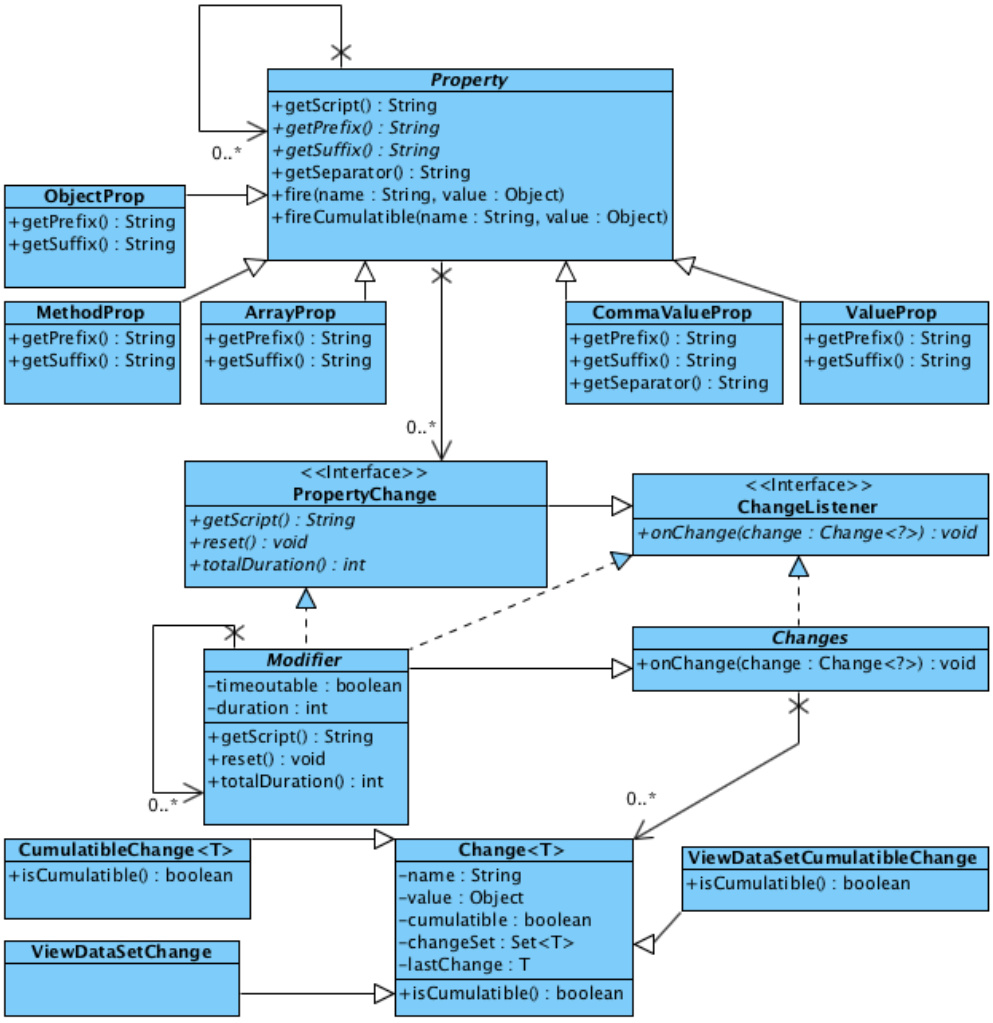
Abstraktní třídu `Changes` pak dědí všechny instance typu `Modifier`. V momentě, kdy budou tyto instance požádány o vygenerování skriptu, požádají zděděnou mapu ze třídy `Changes` o objekt změny pod daným názvem. Pokud existuje, vygenerují skript s hodnotou této změny dle požadavků C3.js API. Pokud je tomu naopak, nevygenerují skript žádný.

Nad třídu `Modifier` bylo postaveno rozhraní s názvem `PropertyModifier`, jenž definuje požadované metody od této třídy. Vyžaduje implementaci metod `getScript()`, `totalDuration()` a `reset()`. Zároveň však toto rozhraní dědí z rozhraní `ChangeListener` a není tedy možné, aby existoval objekt typu `Modifier` bez implementované metody `onChange(Change<?> change)`. Poslední vysvětlení je dlužné metodě `reset()`, jenž se stará o vynulování, případně vymazání všech instancí vzniklých v jednom časovém rozmezí mezi přijetím HTTP požadavku a vygenerování odpovědi. Tím je zamezeno, aby při dalším požadavku existovaly změny z požadavku předchozího. Logicky je tato metoda volána komponentou v momentě, kdy jsou vygenerovány všechny potřebné skripty.

Aby objekty vizuálních vlastností mohly upozorňovat pozorovatele na změny, bylo nutné do třídy `Property` přidat kolekci objektů typu `ChangeListener`. Následně při změně, u které je relevantní notifikovat pozorovatele, je volána metoda `fire(String name, Object value)`, případně `fireCumulatible(String name, Object value)`. Tyto metody projdou celou

kolekci pozorovatelů a u všech zavolají metodu `onChange(Change<?> change)`, kde předávaný objekt obsahuje název a hodnotu změny z předaných parametrů zmíněných metod.

Všechny výše jmenované metody a atributy jsou znázorněny v diagramu tříd na Obrázku 10.



Obrázek 10 : *API pro tvorbu objektů vizuálních vlastností a modifikací (Zdroj: autor)*

4.7.3 Programátorské testy implementace API

Programátorské testy implementace API byly provedeny prostřednictvím jednotkových testů za pomoci frameworku *JUnit*. Celkem bylo vytvořeno 30 testovacích tříd pro metody generující C3.js API skript. Princip spočívá v základní konfiguraci vizuálních vlastností grafu a následném porovnání vygenerovaného skriptu oproti očekávanému. Třídy lze

nalézt v testovacích balíčcích `com.martinlinha.c3faces.script.property` a `com.martinlinha.c3faces.script.modifier`.

Autorem této práce bylo zvažováno automatické webové testování prostřednictvím aplikace Selenium. Vzhledem k poměru náročnosti tvorby těchto testů a potenciálním přínosům byla tato možnost zavržena.

Současně byla implementace API detailně ověřena při tvorbě prezentačního webu a také v následující kapitole, kde je na prostřednictvím API vytvořena knihovna komponent.

5 Použití API pro tvorbu vizuálních komponent

Tato kapitola je orientována na implementaci vizuálních komponent, základní sady tříd pro definici vizuálních vlastností a tříd pro dynamické modifikace vizuálních vlastností. Poněvadž implementace je realizována ve vzniklém API, závěr této kapitoly obsahuje krátké zhodnocení jeho použitelnosti.

5.1.1 Implementace sady komponent vykreslující grafy

Na základě požadavku R14 byly za pomoci vzniklého frameworku (R13) implementovány všechny typy grafů knihovny C3.js. Je tedy možné deklarovat jakýkoliv typ grafu ve Facelet stránce. Jejich výčet dle jmen odpovídající terminologii C3.js společně s Facelet tagy poskytuje Tabulka 6. Jednotlivý tag má jediný povinný atribut nazvaný `data` s očekávaným typem `com.martinlinha.c3faces.property.Data`.

Tabulka 6: *Názvy implementovaných grafů a odpovídající tagy pro Facelet stránky (Zdroj: autor)*

Název grafu	Tag bez atributů
line	<c3:line />
spline	<c3:spline />
step	<c3:step />
area	<c3:area />
area-spline	<c3:area-spline />
area-step	<c3:area-step />
bar	<c3:bar />
scatter	<c3:scatter />
pie	<c3:pie />
donut	<c3:donut />
gauge	<c3:gauge />

5.1.2 Implementace základní sady vizuálních vlastností

K naplnění požadavku R15 je vytvořena série tříd dědících z třídy `Property` a jejích podtypů. Tyto třídy reprezentují jednu či více vizuálních vlastností, které mají být přiřazeny k danému grafu. Souhrn implementovaných tříd spolu s tagy sloužící k deklaraci vlastností ve Facelet stránce obsahuje Tabulka 7.

Tabulka 7: *Názvy implementovaných vizuálních vlastností a odpovídajících tagů (Zdroj: autor)*

Název třídy	**Tag(y) bez atributů**
Area	-
Axis	-
BarProperties	<c3:barProperties />
Bindto	-
Data	-
DonutProperties	<c3:donutProperties />
GaugeProperties	<c3:gaugeProperties />
GridProperties	<c3:grids > <c3:grid /> </c3grids>
Interaction	-
Legend	<c3:legend />
Padding	<c3:padding />
Pie	<c3:pieProperties />
Point	<c3:point />
Regions	<c3:regions> <c3:region /> </c3:regions>
Size	<c3:size />
Subchart	<c3:subchart />
Tooltip	<c3:tooltip />
Transition	-

Název třídy	Tag(y) bez atributů
Zoom	<c3:zoom />

5.1.3 Implementace tříd pro dynamické modifikace

Implementací tříd dědících z typu `Modifier` bylo docíleno splnění požadavku R16 týkající se realizace základní sady tříd pro dynamické modifikace grafu.

Konkrétně byly implementovány třídy

- `Colors`,
- `LegendHide`,
- `LegendShow`,
- `Load`,
- `Names`,
- `Resize`,
- `Transform`,
- `TransformTypes`,
- `XGridAdd`,
- `XGridRemove`,
- `YGridAdd`,
- `YGridRemove`.

5.1.4 Zhodnocení použitelnosti API

Implementace výše uvedených tříd prostřednictvím vytvořeného API proběhla bez významnějších problémů. Je možné konstatovat, že se jednalo i o způsob ověření implementace API. Objevilo se totiž několik drobností, které se týkaly implementačních chyb API a byly tedy opraveny.

K demonstrativním účelům implementovaných komponent byl vytvořen prezentační web, jenž lze nalézt na adrese *http://c3faces.martinlinha.com/*. Jeho obsahem je kromě všech typů grafů řada příkladných implementací včetně zdrojových kódů.

6 Uživatelská příručka komponentové sady

Tato podkapitola je věnována příručce komponentové sady. Postupně se soustředí na základní nastavení, deklaraci grafu ve Facelet stránce, deklaraci vizuálních vlastností grafu a jejich dynamické úpravy. Průběžně také odkazuje na příklady dostupné na prezentačním webu frameworku.

6.1 Základní nastavení

Aby bylo možné využít kapacit knihovny, je nezbytné ji zahrnout mezi knihovny webového projektu. Způsob přidání nové závislosti na knihovně je mimo rozsah této práce. Poté, co je knihovna přítomna, je zpřístupněn nový jmenný prostor pro všechny Facelet stránky obsažené ve webovém projektu. Jeho identifikátorem je *http://c3faces.martinlinha.com/ns*. Všechny příklady v příručce používají prefix tagů *c3*, s kterým ukázka kompletní definice jmenného prostoru vypadá dle Výpisu 12.

Výpis 12: Příklad definice jmenného prostoru knihovny C3Faces s prefixem c3 (Zdroj: autor)

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:c3="http://c3faces.martinlinha.com/ns"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
</html>
```

6.2 Deklarace grafu

Deklarace grafu probíhá prostřednictvím odpovídajících tagů ve Facelet stránce. Pro každý typ grafu, jenž lze vykreslit prostřednictvím C3.js knihovny, framework poskytuje právě jeden tag. Pokud je tedy například žádoucí vykreslit graf typu *Pie*, je použit tag z Výpisu 13. Jelikož všechny komponenty implementují rozhraní `javax.faces.component.EditableValueHolder`, musí být vždy součástí formuláře, tedy obaleny tagem `<form />`.

Výpis 13: Deklarace grafu typu Pie (Zdroj: autor)

```
<h:form>
    <c3:pie />
</h:form>
```

Využívá-li vývojář sofistikovanější IDE, je mu oznámeno, že chybí povinný atribut `data`. Jedná se o atribut, který musí být přítomen u každého grafu a je tedy logicky vyžadován i u tagu. Jeho obsahem je reference na objekt typu `com.martinlinha.c3faces.script.property.Data`. Tento objekt vlastní informace o typu grafu a datových sadách, jež mají být vykresleny. Jelikož typ grafu odpovídá názvu tagu, je frameworkem automaticky doplněn. Pokud jsou však atributy objektu `com.martinlinha.c3faces.script.property.Data` změněny programaticky, lze docílit částečných transformací grafu. Deklarovaný typ grafu ve Facelet stránce tedy není konečný a lze ho v čase měnit.

Poněvadž je nutné doplnit zmíněný povinný atribut, je vytvořena ukázková třída s názvem `ChartBean`, jenž se stará o konstrukci daného objektu. Zároveň tato třída naplní v metodě `init()` seznam ukázkových datových sad společně s ukázkovými atributy. Jedná se o seznam objektů typu `com.martinlinha.c3faces.model.C3ViewDataSet`, kterým jsou při jejich konstrukci předány názvy, datové sady a barvy. Kód třídy `ChartBean` je obsahem Výpisu 14. Na výpisu jsou patrné i anotace z balíčku `javax.faces.bean`, nicméně vysvětlení jejich podstaty není v rozsahu této práce.

Výpis 14: Třída ChartBean obstarávající konstrukci objektu Data (Zdroj: autor)

```
@ManagedBean(name = "chartBean")
@RequestScoped
public class ChartBean implements Serializable {

    private Data data = new Data();

    @PostConstruct
    public void init() {
        data.getDataSets().add(new C3ViewDataSet("Data sample 1",
                new C3DataSet(Arrays.asList(22, 91, 158, 93, 201, 11)),
"#1060E0"));
        data.getDataSets().add(new C3ViewDataSet("Data sample 2",
                new C3DataSet(Arrays.asList(1, 72, 23, 33, 21)), "#16525E"));
        data.getDataSets().add(new C3ViewDataSet("Data sample 3",
                new C3DataSet(Arrays.asList(45, 231, 2, 151, 341, 254)),
"#0792AD"));
        data.getDataSets().add(new C3ViewDataSet("Data sample 4",
                new C3DataSet(Arrays.asList(88, 13, 258, 211, 151)),
"#8FE2F2"));

    }

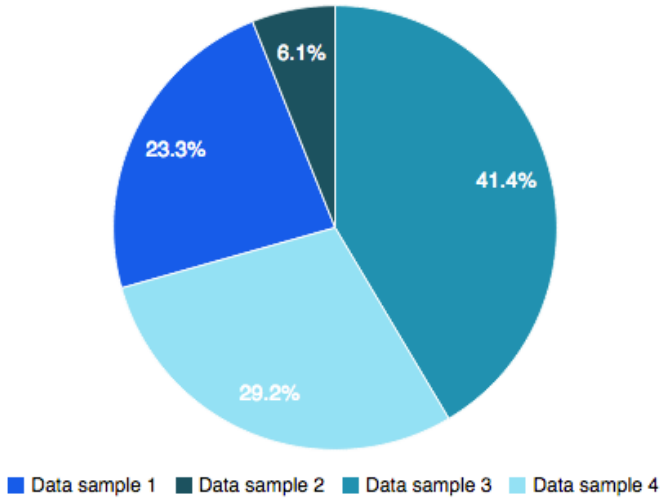
    public Data getData() {
        return data;
    }
}
```

Na základě skutečnosti, že třída `ChartBean` poskytuje vyžadovaný atribut prostřednictvím metody `getData()`, je nyní možné úspěšně dokončit deklaraci tagu ve Facelet stránce, jak uvádí Výpis 15.

Výpis 15: Deklarace grafu typu Pie (Zdroj: autor)

```
<h:form>
    <c3:pie data="#{chartBean.data}" />
</h:form>
```

Tento krátký kus kódu společně s rozebíranou referencí má za následek vygenerování grafu, jenž je znázorněn na Obrázku 11 níže.



Obrázek 11 : *Vygenerovaný graf za pomoci kódu z Výpisu 15 (Zdroj: autor)*

Kompletní výčet typů grafů deklarovatelných pomocí tagů (typ grafu odpovídá názvu grafu) společně s odkazem na příklad z prezentačního webu zahrnuje

- `<c3:area ... />` (*http://c3faces.martinlinha.com/faces/area.xhtml*),
- `<c3:area-spline ... />` (*http://c3faces.martinlinha.com/faces/area-spline.xhtml*),
- `<c3:area-step ... />` (*http://c3faces.martinlinha.com/faces/area-step.xhtml*),
- `<c3:bar ... />` (*http://c3faces.martinlinha.com/faces/bar.xhtml*),
- `<c3:donut ... />` (*http://c3faces.martinlinha.com/faces/donut.xhtml*),
- `<c3:gauge ... />` (*http://c3faces.martinlinha.com/faces/gauge.xhtml*),
- `<c3:pie ... />` (*http://c3faces.martinlinha.com/faces/pie.xhtml*),
- `<c3:line ... />` (*http://c3faces.martinlinha.com/faces/line.xhtml*),
- `<c3:scatter ... />` (*http://c3faces.martinlinha.com/faces/scatter.xhtml*),

- `<c3:spline ... />` (*http://c3faces.martinlinha.com/faces/spline.xhtml*),
- `<c3:step ... />` (*http://c3faces.martinlinha.com/faces/step.xhtml*).

6.2.1 Datové sady grafu

V uvedeném Výpisu 14 je provedeno ukázkové naplnění dat. Níže jsou popsány jednotlivé třídy nezbytné k těmto účelům.

Třída com.martinlinha.c3faces.model.C3DataSet

Objekty této třídy vlastní seznam čísel, tedy hodnot vykreslených v grafu.

Třída com.martinlinha.c3faces.model.C3ViewDataSet

Třída vlastní reference na datové sady typu `com.martinlinha.c3faces.model.C3DataSet` společně s jejich vizuálními vlastnostmi. Zároveň je vydavatelem změn, tedy instancí typů `com.martinlinha.c3faces.listener.change.C3ViewDataSetChange` a `com.martinlinha.c3faces.listener.change.C3ViewDataSetCumulatibleChange`. Objektům této třídy jsou při inicalizaci automaticky přiřazeny unikátní identifikátory vyžadované ze strany C3.js. Mezi nastavitelné vizuální vlastnosti patří

- barva (`color`),
- popisek (`name`),
- vizuální typ datové sady (`type`).

6.3 Deklarace vizuálních vlastností grafů

Tato podkapitola na příkladu popisuje způsob, jakým lze deklarovat vizuální vlastnost grafu ve Facelet stránce. Dále pak poskytuje kompletní seznam všech nabízených tagů.

Příklad rozsáhlejší deklarace vizuálních atributů grafu je společně se zdrojovým kódem k nahlédnutí na *http://c3faces.martinlinha.com/faces/chart_properties_facelet.xhtml*.

6.3.1 Tag barProperties

Aplikace vizuální vlastnosti probíhá prostřednictvím deklarace v těle tagu definující graf. Tag `<c3:barProperties />` je používán, je-li typ generovaného grafu *bar* a slouží k deklaraci vizuálních vlastností tohoto typu grafu. Příklad deklarace uvádí Výpis 16.

Výpis 16: Deklarace vlastnosti barProperties pro graf typu bar (Zdroj: autor)

```
<c3:bar data="#{chartBean.data}" >
    <c3:barProperties width="50" zerobased="true" />
</c3:bar>
```

Vykreslený graf bude sloupcový, se sloupci o šířce 50 px a minimální hodnota pro osy X a Y bude hodnota 0. Atributy tagu s kompletními údaji popisuje následující Tabulka 8.

Tabulka 8: *Popis atributů pro tag barProperties (Zdroj: autor)*

Název atributu	Datový typ	Povinný	Popis	Výchozí hodnota
width	java.lang.Integer	NE	šířka sloupce v pixe-lech	*automaticky počítané*
zerobased	java.lang.Boolean	NE	minimální hodnota na osách bude 0	true

6.3.2 Tag donutProperties

Tag popisující vlastnosti pouze je-li použit typ grafu *donut*. Tabulka 9 popisuje jednotlivé atributy.

Tabulka 9: *Popis atributů pro tag donutProperties (Zdroj: autor)*

Název atributu	Datový typ	Povinný	Popis	Výchozí hodnota
expand	java.lang.Boolean	NE	jednotlivé kusy koláče budou rozbalitelné	true
showLabels	java.lang.Boolean	NE	mají se zobrazit v grafu hodnoty v procentech	true
title	java.lang.String	NE	titulek zobrazený upro-střed grafu	*prázdné*
width	java.lang.Integer	NE	šířka jednotlivých kusů koláče v pixelech	*automaticky počítané*

6.3.3 Tag gaugeProperties

Tag popisující vlastnosti grafu typu *gauge*. Atributy obsahuje Tabulka 10.

Tabulka 10: *Popis atributů pro tag gaugeProperties (Zdroj: autor)*

Název atributu	Datový typ	Povinný	Popis	Výchozí hodnota
expand	java.lang.Boolean	NE	manometr bude rozbalitelný	true
showLabels	java.lang.Boolean	NE	mají se zobrazit v grafu hodnoty v procentech	true
max	java.lang.Integer	NE	maximální zobrazitelná hodnota	100
min	java.lang.Integer	NE	minimální zobrazitelná hodnota	0
units	java.lang.String	NE	jednotky k zobrazení pod grafem	*prázdné*
width	java.lang.Integer	NE	šířka manometru	*automatický počítané*

6.3.4 Tagy grids, gridX a gridY

Tagy `<c3:gridX />` a `<c3:gridY />` slouží k definici libovolného počtu mřížek, jež mají být vykresleny na grafu. První zmíněný vykresluje mřížku kolmou na osu X, druhý kolmou na osu Y. Oba typy jsou vkládány do těla tagu `<c3:grids />`, který slouží jako držitel seznamu mřížek s žádnými atributy. Atributy jak pro `<c3:gridX />`, tak `<c3:gridY />`, uvádí Tabulka 11.

Tabulka 11: *Popis atributů pro tagy gridX a gridY (Zdroj: autor)*

Název atributu	Datový typ	Povinný	Popis	Výchozí hodnota
text	java.lang.String	ANO	text zobrazený vedle mřížky	*prázdné*
value	java.lang.Double	ANO	hodnota na vybrané ose, odkud má být mřížka vykreslena	*žádná*

6.3.5 Tag legend

Tento tag slouží úpravě vizuálních atributů legendy. Jednotlivé atributy s popisky znázorňuje Tabulka 12.

Tabulka 12: *Popis atributů pro tag legend (Zdroj: autor)*

Název atributu	Datový typ	Povinný	Popis	Výchozí hodnota
hide	java.lang.Boolean	NE	má být legenda scho-vána	false
insetAnchor	com.martinlinha.c3faces.script.property.Legend.InsetAnchor	NE	je-li pozice INSET, specifikuje konkrétní umístění, povoleno TOPLEFT, TOPRIGHT, BOTTOMLEFT, BOTTOMRIGHT	TOPLEFT
insetStep	java.lang.Integer	NE	je-li pozice INSET, definuje počet řádků legendy	*maximum*
insetX	java.lang.Integer	NE	je-li pozice INSET, definuje vzdálenost na ose X	*automaticky počítané*
insetY	java.lang.Integer	NE	je-li pozice INSET, definuje vzdálenost na ose Y	*automaticky počítané*
onclick	java.lang.String	NE	JavaScript volaný po kliknutí na legendu	*prázdný*
onmouseout	java.lang.String	NE	JavaScript volaný po opuštění kurzoru myši nad pozicí legendy	*prázdný*
onmouseover	java.lang.String	NE	JavaScript volaný po přejetí myši nad pozicí legendy	*prázdný*
position	com.martinlinha.c3faces.script.property.Legend.Position	NE	definuje pozici, kde má být legenda vykreslena. Povoleno BOTTOM, RIGHT, INSET	BOTTOM
show	java.lang.Boolean	NE	má být legenda zobra-zena	true

6.3.6 Tag padding

Tag `<c3:padding />` slouží ke specifikaci prázdné výplně od určité strany grafu v pixelech. Atributy jsou obsaženy v Tabulce 13.

Tabulka 13: *Popis atributů pro tag padding (Zdroj: autor)*

Název atributu	Datový typ	Povinný	Popis	Výchozí hodnota
bottom	java.lang.Integer	NE	výplň zdola v pixelech	*žádná*
left	java.lang.Integer	NE	výplň zleva v pixelech	*žádná*
right	java.lang.Integer	NE	výplň zprava v pixelech	*žádná*
top	java.lang.Integer	NE	výplň shora v pixelech	*žádná*

6.3.7 Tag pieProperties

Tento tag je určen ke specifikaci vlastností koláčového typu grafu. Dva volitelné atributy jsou obsaženy v Tabulce 14.

Tabulka 14: *Popis atributů pro tag pieProperties (Zdroj: autor)*

Název atributu	Datový typ	Povinný	Popis	Výchozí hodnota
expand	java.lang.Boolean	NE	jednotlivé kusy koláčového grafu budou rozbalitelné	true
showLabels	java.lang.Boolean	NE	mají se zobrazit v grafu hodnoty v procentech	true

6.3.8 Tag point

`<c3:point />` slouží ke specifikaci vizuálních atributů bodů nalézajících se na grafu. Jejich výčet zobrazuje Tabulka 15.

Tabulka 15: *Popis atributů pro tag point (Zdroj: autor)*

Název atributu	Datový typ	Povinný	Popis	Výchozí hodnota
expand	java.lang.Boolean	NE	jednotlivé kusy koláčového grafu budou rozbalitelné	true
expandR	java.lang.Double	NE	poloměr bodu po rozbalení	r * 1.75

Název atributu	Datový typ	Povinný	Popis	Výchozí hodnota
r	java.lang.Double	NE	poloměr každého bodu	2.5
selectR	java.lang.Double	NE	poloměr bodu, je-li vybrán	r * 4
show	java.lang.Boolean	NE	mají se body na grafu zobrazovat	true

6.3.9 Tagy regions a region

V těle tagu `<c3:regions />` lze definovat libovolný počet tagů `<c3:region />`, které slouží k zvýraznění zvolených oblastí grafu podbarvením. `<c3:regions />` nezahrnuje žádné vlastní atributy a slouží pouze jako vlastník seznamu oblastí, jejichž atributy obsahuje Tabulka 16.

Tabulka 16: *Popis atributů pro tag region (Zdroj: autor)*

Název atributu	Datový typ	Povinný	Popis	Výchozí hodnota
axis	java.lang.String	ANO	jednotlivé kusy koláčového grafu budou rozbalitelné	*žádná*
cssClass	java.lang.String	NE	CSS třída pro daný region	*žádná*
start	java.lang.Double	NE	bod, kde má region začínat	*žádná*
end	java.lang.Double	NE	bod, kde má region končit	*žádná*

6.3.10 Tag size

Tento tag slouží k deklaraci šířky a výšky grafu. Podrobnosti zachycuje Tabulka 17.

Tabulka 17: *Popis atributů pro tag size (Zdroj: autor)*

Název atributu	Datový typ	Povinný	Popis	Výchozí hodnota
height	java.lang.Integer	NE	výška vykresleného grafu	*žádná*
width	java.lang.Integer	NE	šířka vykresleného	*žádná*

Název atributu	Datový typ	Povinný	Popis	Výchozí hodnota
			grafu	

6.3.11 Tag subchart

Prostřednictvím tohoto tagu lze zobrazit dílčí graf, jenž bude zobrazen pod grafem hlavním. Jednotlivé atributy dílčího grafu jsou zachyceny v Tabulce 18.

Tabulka 18: *Popis atributů pro tag subchart (Zdroj: autor)*

Název atributu	Datový typ	Povinný	Popis	Výchozí hodnota
height	java.lang.Integer	NE	výška dílčího grafu	*žádná*
onbrush	java.lang.String	NE	JavaScript provedený při pohybu na dílčím grafu	*žádná*
show	java.lang.Boolean	NE	má být dílčí graf vykreslen	false

6.3.12 Tag tooltip

`<c3:tooltip />` tag umožňuje specifikovat podrobnosti popisků zobrazených po najetí myši na datovou sadu. Podrobnosti atributů jsou obsaženy v Tabulce 19.

Tabulka 19: *Popis atributů pro tag tooltip (Zdroj: autor)*

Název atributu	Datový typ	Povinný	Popis	Výchozí hodnota
grouped	java.lang.Boolean	NE	popisky mají být seskupeny	true
show	java.lang.Boolean	NE	mají být popisky zobrazeny	true

6.3.13 Tag zoom

Zobrazená data na grafu lze přibližovat a oddalovat. Nastavení této vlastnosti je možné prostředictvím tagu `<c3:zoom />`, jehož podrobnosti zobrazuje Tabulka 20.

Tabulka 20: *Popis atributů pro tag zoom (Zdroj: autor)*

Název atributu	Datový typ	Povinný	Popis	Výchozí hodnota
enabled	java.lang.Boolean	NE	přibližování má být umožněno	false
extentFrom	java.lang.Integer	NE	minimální přiblížení	1
extentTo	java.lang.Integer	NE	maximální přiblížení	10
onzoom	java.lang.String	NE	JavaScript provedený při přiblížení grafu	*prázdné*
rescale	java.lang.Boolean	NE	umožní změnit měřítko po přiblížení	false

6.4 Práce s vybranou datovou sadou na grafu

Nad jednotlivými grafy lze poklikáním vybrat datovou sadu na grafu, jenž je reprezentována body na křivce, částí koláčového grafu a podobně. Tyto sady jsou na serverové straně zachyceny objekty typu `com.martinlinha.c3faces.model.C3ViewDataSet`. V momentě, kdy je odeslán formulář s vnořeným grafem, jenž zároveň eviduje vybranou datovou sadu, je selekce přeložena na objekt tohoto typu. Každý tag z knihovny C3Faces představující typ grafu nabízí atribut `value`, který poskytuje referenci na takto přeložený objekt.

Deklarace grafu poskytující referenci na instanci objektu vybrané datové sady popisuje příklad na Výpisu 17.

Výpis 17: Deklarace grafu s referencí na aktuálně vybranou datovou sadu (Zdroj: autor)

```
<c3:bar data="#{chartBean.data}" value="#{chartBean.selectedC3ViewDataSet}" />
```

6.5 Aktivace AJAX

Všechny komponenty z knihovny C3Faces vykreslující grafy podporují technologii AJAX. Její aktivace probíhá obdobně jako u jiných JSF komponent, tedy prostřednictvím tagu `<f:ajax />` z jmenného prostoru `xmlns:f=http://xmlns.jcp.org/jsf/core`. Příklad uvádí Výpis 18.

Výpis 18: Aktivace AJAX pro komponentu grafu (Zdroj: autor)

```
<c3:bar data="#{chartBean.data}" >
    <f:ajax render="@this" execute="@this" event="click" />
</c3:bar>
```

Atribut `event`, specifikující DOM událost, může nabývat pouze hodnoty `click`. Ta je zároveň hodnotou výchozí. Není tedy nutné tento atribut uvádět, framework ji doplní sám.

AJAX požadavky pro graf mohou být samozřejmě vyvolány i z událostí jiných komponent. Příklad z prezentačního webu na adrese *http://c3faces.martinlinha.com/faces/change_chart_type.xhtml* obsahuje komponentu rozbalovacího menu, jenž ve svém těle deklaruje AJAX akci odkazující na graf. Po zvolení položky z menu se graf adekvátně překreslí.

6.6 Posluchače změn

Knihovna také podporuje přidávání posluchačů změn na vybrané datové sady v grafu. Obdobně jako při přidávání AJAX podpory z kapitoly 7.4, stačí ze stejného jmenného prostoru přidat do těla komponenty tag `<f:valueChangeListener />` s hodnotou atributu `type` odpovídající názvu třídy, jenž má být posluchačem těchto změn. Jak je definováno ve specifikaci JSF, třída poslouchající změny hodnot musí implementovat rozhraní `javax.faces.event.ValueChangeListener`.

Příklad včetně zdrojových kódu je k naleznutí na adrese *http://c3faces.martinlinha.com/faces/chart_value_listener.xhtml*.

6.7 Dynamické modifikace vizuálních vlastností grafu

Část objektů vlastností specifikovaných v kapitole 6.3 má po modifikaci na serverové straně svůj odraz na klientu. Změní-li se vlastnost až poté, co již byla vykreslena, graf se prostřednictvím přechodových animací částečně překreslí. Poněvadž je toto realizováno prostřednictvím JavaScript volání, jsou podmínkou požadavky typu AJAX.

Částečné překreslování grafu je pozorovatelné u většiny příkladů z prezentačního webu, nejpatrněji pak na adrese *http://c3faces.martinlinha.com/faces/chart_values.xhtml*.

C3.js API nepodporuje dynamické modifikace pro všechny vlastnosti, což je odrazem stejného omezení i v této JSF knihovně. Výčet modifikovatelných vlastností této knihovny je součástí kapitoly. Zároveň je součástí ukázka, jak lze modifikace realizovat.

6.7.1 Programatické získání a změna vizuální vlastnosti grafu

Aby bylo možné upravit určitou vlastnost grafu na serveru, je nutné získat referenci příslušného objektu. Poté nad ním lze požadovanou změnu realizovat. Jelikož jsou tyto vlastnosti vztaženy ke konkrétní instanci grafu, je nejdříve nezbytné získat referenci na ní.

Existuje více způsobů, jak získat odkaz na komponentu v JSF, avšak jejich popis není v rozsahu této práce. Bude zde využit způsob založený na atributu `binding`, který všechny komponenty z knihovny C3Faces ve svém tagu nabízejí. Deklarovaná komponenta ve Facelet stránce pak vypadá například dle Výpisu 19.

Výpis 19: Binding instance komponenty (Zdroj: autor)

```
<c3:bar data="#{chartBean.data}" binding="#{chartBean.chart}" />
```

Zde je pomocí EL výrazu odkazováno na objekt, jenž je součástí třídy `ChartBean`. Jelikož je navazovaná instance grafu z knihovny C3Faces, je nutné pro něj v této třídě poskytnout objekt typu `com.martinlinha.c3faces.component.C3chart` s odpovídajícími *getter* a *setter* metodami.

Každá komponenta má své vlastnosti uložené v objektu třídy `com.martinlinha.c3faces.component.ComponentProperties`, jenž je dostupný z metody `getComponentProperties()`. Ten vlastní mapu objektů vizuálních vlastností, kde klíčem je název dané vlastnosti. Grafu tedy lze přidávat dynamicky vizuální vlastnosti bez nutnosti zásahu do existujícího kódu, jak žádá požadavek R04. Nyní, je-li získán objekt komponentové třídy pod proměnnou `chart`, lze tuto mapu prohledat dle názvu požadované vlastnosti. Výpis 20 znázorňuje získání objektu `com.martinlinha.c3faces.script.property.Size` vlastnící informace o velikosti grafu.

Výpis 20: Získání objektu reprezentující velikost grafu (Zdroj: autor)

```
Size size = (Size) chart.getComponentProperties().getProperty(Size.NAME);
```

Nyní je možné prostřednictvím příslušných metod změnit stav získaného objektu. Pro účely tohoto příkladu byla změněna výška grafu pomocí metody `setHeight(100)`. Takto vyvolaná změna zapříčiní notifikaci zaregistrovaných objektů implementujících rozhraní `com.martinlinha.c3faces.listener.ChangeListener`. Konkrétně tuto změnu zachytí a využívá třída `com.martinlinha.c3faces.listener.Resize`, jenž je ve vhodný okamžik požádána o vygenerování modifikačního skriptu obsahující provedené změny.

Příklady této funkcionality komplexně pokrývají příklady na adresách *http://c3faces.martinlinha.com/faces/bulk_changes.xhtml* a *http://c3faces.martinlinha.com/faces/bulk_changes_editable.xhtml*.

6.7.2 Modifikovatelné vizuální vlastnosti

Objekty vizuálních vlastností, jako například velikost grafu zmíněná v kapitole 6.7.1, mohou mít své pozorovatele. Ti na základě změn generují modifikační C3.js skripty. Jedná se o třídy dědící abstraktní třídu `com.martinlinha.c3faces.listener.Modifier`, v překladu o modifikátory. Jelikož vlastnost a modifikátor na sobě nejsou závislé, je možné přiřazovat modifikátory k vlastnostem libovolně. Knihovna C3Faces obsahuje implementovanou základní sadu tříd vlastností a modifikátorů. Seznam těchto vlastností společně s doporučeným modifikátorem je obsahem Tabulky 21.

Tabulka 21: *Seznam tříd vlastností a modifikátorů v C3Faces (Zdroj: autor)*

Vlastnost	Modifikátor
com.martinlinha.c3faces.script.property.Data	com.martinlinha.c3faces.listener.Colors com.martinlinha.c3faces.listener.Load com.martinlinha.c3faces.listener.Names com.martinlinha.c3faces.listener.Transform com.martinlinha.c3faces.listener. TransformTypes
com.martinlinha.c3faces.script.property. GridProperties	com.martinlinha.c3faces.listener.XGridAdd com.martinlinha.c3faces.listener.XGridRemove com.martinlinha.c3faces.listener.YGridAdd com.martinlinha.c3faces.listener.YGridRemove
com.martinlinha.c3faces.script.property. Legend	com.martinlinha.c3faces.listener.LegendHide com.martinlinha.c3faces.listener.LegendShow
com.martinlinha.c3faces.script.property.Size	com.martinlinha.c3faces.listener.Resize

6.7.3 Programatické přidání dynamické vlastnosti

Programatické přidání vlastnosti probíhá téměř analogicky jako úprava vlastnosti existující z kapitoly 6.7.1. Podstatný rozdíl nastává v případě, kdy má mít přidávaná vlastnost schopnost částečných modifikací grafu. V tomto případě je nutné objektu typu `com.martinlinha.c3faces.script.Property` přiřadit objekt třídy `com.martinlinha.c3faces.listener.Modifier`. Pro úplnost lze zmínit, že definuje-li vývojář vlastnost ve Facelet stránce, o přiřazení modifikátoru se postará instance třídy `com.martinlinha.c3faces.component.C3Property`.

Je-li například přidávána vizuální vlastnost definující atributy legendy grafu, je postupně vytvořen objekt dané třídy, přiřazen ke komponentě, přidány modifikátory a poté případně nastaven stav objektu reprezentující legendu. Pokud by byl nejdříve objekt měněn a teprve poté přidán modifikátor, nebyly by zachyceny změny a tudíž by nebyly vykresleny modifikace na grafu. Důvodem je, že modifikátor v roli pozorovatele v době úprav ještě neodposlouchával změny provedené nad vlastností. Příklad správně definované vlastnosti

společně s modifikátorem popisuje Výpis 21, jehož výsledkem je skrytá legenda pod grafem.

Výpis 21: Programatické přidání vlastnosti společně s odpovídajícím modifikátorem (Zdroj: autor)

```
Legend legend = new Legend();
chart.getComponentProperties().addProperty(legend);
legend.addListener(new LegendHide().addModifier(new LegendShow()));
legend.setHide(true);
```

7 Dokumentace API

Tato kapitola zahrnuje dokumentaci pro práci s API frameworku C3Faces. Zprvu obecně API popisuje, pokračuje principy jeho používání založené na příkladech a na závěr obsahuje referenční příručku.

7.1 Přehled

JavaScriptová knihovna C3.js je stále ve vývoji a postupem času jsou vytvářeny nové typy grafů, vizuálních vlastností, či jsou rozšiřovány možnosti C3.js API o nové funkce. Zároveň v základní distribuci frameworku C3Faces nejsou pokryty kompletní vizuální možnosti grafů aktuální verze C3.js. Z těchto důvodů toto API slouží především pro účely snadného rozšiřování knihovny C3Faces. Jejím prostřednictvím lze tedy vytvářet nové vlastnosti, modifikátory, ale i komponenty pro vykreslování grafů a komponenty pro deklaraci vlastností ve Facelet stránkách.

Úvodní část je věnována způsobu práce s API, v závěru lze nalézt referenční příručku.

Jedinou nezbytnou podmínkou pro používání tohoto API je přidání závislosti na knihovně C3Faces dostupné z *http://www.github.com/martin-linha/c3faces*.

7.2 Příkladné implementace pomocí API

Podkapitola se zabývá komentovanými příklady používání API. Součástí příkladů jsou ukázky zdrojového kódu.

7.2.1 Implementace komponenty pro graf

Tvorba komponenty vykreslující graf se skládá ze dvou částí. V prvé řadě je nutné vytvořit komponentovou třídu. Tu je následně třeba zaregistrovat do kontextu JSF a knihovny tagů. První zmíněná část je podporována frameworkem C3Faces.

Prvním krokem je vytvoření třídy dědící z abstraktní třídy `com.martinlinha.c3faces.component.C3Chart`. Ta obstarává implementaci celého životního cyklu zpracování požadavku, jenž je pro komponentu klíčový. Zároveň obsahuje všechny patřičné závislosti na externí zdrojové soubory, které jsou potřeba k vykreslování C3.js grafů. Konkrétně se jedná o závislosti na CSS styly, JavaScript knihovny pro D3.js, C3.js a c3faces.js.

Jedinou abstraktní a tedy vyžadovanou metodou je metoda `getCustomProperties()`. Jedná se o metodu, jejímž prostřednictvím lze přidat výchozí vizuální vlastnosti grafu. Očekává tedy seznam objektů typu `com.martinlinha.c3faces.script.Property`.

Pro definici typu grafu stačí překrýt metodu `getChartType()` a v ní vrátit název typu grafu dle požadavků C3.js. Pro srozumitelnější výklad je uveden příklad, v jehož rámci je implementován graf typu *pie*. Zdrojový kód je obsažen ve Výpisu 22. V něm si lze současně povšimnout, že třída je anotována anotací `@FacesComponent`. Jejím prostřednictvím je třída registrována do JSF kontextu a lze na ní později odkazovat při definici knihovny tagů[7].

Výpis 22: Komponentová třída vykreslující graf typu pie (Zdroj: autor)

```
@FacesComponent("com.martinlinha.c3faces.component.Pie")
public class Pie extends C3Chart {
    private final List<Property> customProperties = new ArrayList<>();

    @Override
    public List<Property> getCustomProperties() {
        return customProperties;
    }

    @Override
    public String getChartType() {
        return "pie";
    }
}
```

Je-li žádoucí, aby graf měl ve svém výchozím nastavení například šířku 250 px a výšku také 250 px, stačí přidat do odpovídajícího seznamu objekt reprezentující tuto vlastnost. Kód celé překryté metody následně odpovídá Výpisu 23 níže.

Výpis 23: Překrytá metoda sloužící k definici výchozích vlastností grafu (Zdroj: autor)

```
@Override
public List<Property> getCustomProperties() {
    Size size = new Size();
    size.setWidth(250);
    size.setHeight(250);
    customProperties.add(size);

    return customProperties;
    }
```

[7] Registrovat komponentu lze také prostřednictvím XML konfigurace v souboru faces-config.xml

Aby bylo možné komponentu použít ve Facelet stránce, je ještě nezbytné komponentu deklarovat v knihovně tagů. Pro uvedený příklad výše by záznam v odpovídající knihovně vypadal obdobně jako kód ve Výpisu 24.

Výpis 24: Definice grafu v knihovně tagů (Zdroj: autor)

```
<tag>
    <tag-name>pie</tag-name>
    <component>
       <component-type>com.martinlinha.c3faces.component.Pie</component-type>
    </component>
</tag>
```

Kompletní specifikaci API třídy `com.martinlinha.c3faces.component.C3Chart` lze nalézt v referenční příručce v kapitole 7.3.1.

7.2.2 Implementace vlastnosti grafu

Všechny třídy reprezentující vlastnosti grafu dědí z abstraktní třídy `com.martinlinha.c3faces.script.property.Property` nebo z některého jeho potomků. Hlavním produktem těchto tříd je vygenerovaný JavaScriptový kód, jenž je následně odesílaný v HTTP odpovědích. Framework obsahuje implementaci celkem pěti tříd sloužící jako šablony bloků odpovídající JSON notaci pro různé druhy produkovaného skriptu. Jejich výčet je obsažen v Tabulce 22 společně s příklady vytvořených skriptů.

Tabulka 22: *Šablony pro generování rozdílných typů skriptů společně s příklady (Zdroj: autor)*

Název třídy	**Příklad produkované skriptu**
com.martinlinha.c3faces.script.ArrayBlock	columns: [22]
com.martinlinha.c3faces.script.CommaValueBlock	label: 'Data 1'
com.martinlinha.c3faces.script.MethodBlock	onzoom: function (param) { ... }

com.martinlinha.c3faces.script.ObjectBlock	point: {
	...
	}

com.martinlinha.c3faces.script.ValueBlock	show: false

K objasnění těchto šablon slouží nadcházející příklad. Cílem je implementace třídy reprezentující vizuální atributy bodů na grafu. Dle C3.js dokumentace by měl mít výsledný skript strukturu dle příkladu ve Výpisu 25.

Výpis 25: Příklad C3.js kódu definující vizuální vlastnosti bodů na grafu (Zdroj: autor)

```
point: {
    r: 5,
    focus: {
        expand: {
            enabled: true
        }
    }
}
```

Objekt na nejvyšší úrovni má název `point` a struktura tohoto bloku odpovídá šabloně `com.martinlinha.c3faces.script.ObjectBlock`. Konstruovaná třída tedy bude dědit tuto třídu. Pomocí překrytí metody `getName()` je nutné vrátit název objektu nejvyšší úrovně. Jelikož každá komponenta může evidovat vlastnost s daným názvem nanejvýš jednou, je tento název současně unikátní identifikátor. Ten bude později sloužit k vyhledání objektu této třídy v mapě komponentových vlastností.

Na základě skutečnosti, že třída `com.martinlinha.c3faces.property.Property` implementuje návrhový vzor *Composite*, každá jeho instance vlastní seznam členů stejného datového typu. Tento seznam bude naplněn v překryté metodě `preScriptBuild()` sloužící jako *template* metoda volaná při začátku metody generující výsledný skript. Ta pak pomocí rekurze projde všechny uzle kompozitního stromu a postupně sestaví skript v požadované struktuře. Přidání člena obstarává zděděná metoda `addChild(Property property)`. Java kód sloužící k vytvoření skriptu z Výpisu 25 je uveden ve Výpisu 26.

Výpis 26: Java kód generující JavaScript z Výpisu 25 (Zdroj: autor)

```
@Override
public void preScriptBuild() {
        addChild(new ValueBlock("r", r));
        addChild(new ObjectBlock("focus", new ObjectBlock("expand",
                    new ValueBlock("enabled", expand))));
}
```

Nyní je možné instanci vytvořené třídy přidat do kolekce vlastností grafu.

7.2.3 Implementace komponenty vizuální vlastnosti grafu

Protože vytvořenou vlastnost z předchozího příkladu zatím nelze přidat prostřednictvím Facelet stránky, je nutné vytvořit odpovídající komponentovou třídu. V tomto případě je nezbytné, aby konstruovaná třída dědila ze třídy `com.martinlinha.c3faces.component.property.C3Property`. Ta vyžaduje implementaci jedné metody, a sice `getAssociatedProperty()` s návratovým typem `Property`. Zde tedy pouze stačí vrátit objekt třídy vytvořený v předchozím příkladu společně s hodnotami

získaných z Facelet stránky. Řešení společně s užitím pomocných statických metod ze třídy `com.martinlinha.c3faces.util.ComponentUtil`, jež jsou popsány v kapitole 7.3.7, obsahuje Výpis 27 níže.

Výpis 27: Vrácení instance vytvořené vlastnosti pro Facelet komponentu (Zdroj: autor)

```
@Override
public Property getAssociatedProperty() {
  return new Point(ComponentUtil.parseBoolean(getAttributes().get("expand")),
                   ComponentUtil.parseDouble(getAttributes().get("r")));
```

Zbývá jen komponentu zaregistrovat v knihovně tagů a JSF kontextu obdobně jako v kapitole 7.2.1, a povolit atributy tagu s názvy `expand` a `r`.

7.2.4 Implementace modifikátoru

Implementace modifikátoru bude příkladně provedena pro vlastnost určující velikost grafu. V první řadě vytvářená třída musí zdědit třídu `com.martinlinha.c3faces.listener.Modifier`. Dle jejího předmětu byla vytvářená třída nazvána `Resize`. Následně je nutné implementovat abstraktní metodu `getModificationProperty()` vracející objekt typu `Property`. Jelikož se jedná o změnu velikosti, odpovídající třída typu `Property` v tomto případě je třída `com.martinlinha.c3faces.property.Size`.

Je nutné vrátit novou instanci, která bude mít nastavené své atributy podle změn, jenž se za dobu vyřizování HTTP požadavku na serveru udály. Aby bylo možné tyto změny identifikovat, je nutné zaregistrovat nového posluchače změn u třídy `com.martinlinha.c3faces.property.Size` a určit místa, kde budou změny posluchačům hlášeny.

U demonstrovaného příkladu by upravené metody zmíněné třídy vypadaly dle Výpisu 28 níže.

Výpis 28: Metody třídy com.martinlinha.c3faces.property.Size upozorňující pozorovatele na změny (Zdroj: autor)

```
    public void setHeight(Integer height) {
        fire("sizeHeight", height);
        this.height = height;
    }
    public void setWidth(Integer width) {
        fire("sizeWidth", width);
        this.width = width;
    }
```

Nyní již jen zbývá při inicializaci objektu typu `com.martinlinha.c3faces.property.Size` přidat třídu `Resize` jako posluchače změn prostřednictvím metody `addListener(ChangeListener listener)`.

Tím byly zachyceny případy, kdy jsou upraveny atributy existující instanci této třídy. V momentě, kdy dojde k úpravě výšky nebo šířky grafu, je zavolána metoda `fire(String key, Object value)`, kterou třída zdědila od abstraktní třídy `com.martinlinha.c3faces.property.Property`. V prvním parametru této metody je zmíněn název změny, který se ukládá jako klíč do mapy objektů a tedy musí být unikátní. Druhým předávaným parametrem je pak objekt samotné změny.

Nyní si už vytvářená třída `Resize` bude moci vyzvednout změny pod specifikovanými klíči a nastavit je v předávaném objektu typu `com.martinlinha.c3faces.property.Size`. K naleznutí nastalých změn lze použít zděděnou metodu `getPropertyLastChange()` ze třídy `com.martinlinha.c3faces.listener.change.Changes`. Stejná třída totiž obsahuje onu mapu, do které se vytvořené změny ukládají.

Poslední maličkostí, nezbytnou pro úspěšnou kompilaci třídy `Resize`, je implementace metody `getMethodName()` vracející řetězec. Zde je vrácen název metody odpovídající názvu požadované JavaScript metody v C3.js API, tedy `resize`.

Kompletní kód třídy, jenž úspěšně poslouchá změny a případně generuje odpovídající skripty pro modifikaci grafu, obsahuje Výpis 29.

Výpis 29: Třída generující skript pro dynamickou změnu velikosti grafu (Zdroj: autor)

```
public class Resize extends Modifier {

    @Override
    protected Property getModificationProperty() {
        return new Size((Integer) getPropertyLastChange("sizeWidth"),
                        (Integer) getPropertyLastChange("sizeHeight"));
    }

    @Override
    protected String getMethodName() {
        return "resize";
    }
}
```

7.3 Referenční příručka

7.3.1 Balíček com.martinlinha.c3faces.component

Abstraktní třída com.martinlinha.c3faces.component.C3Chart

Výchozí třída pro tvorbu komponent vykreslující C3.js grafy. Třída překrývá metody třídy `javax.faces.component.UIInput` související s fázemi životního cyklu zpracování požadavku. Aby byl přístup do jednotlivých fází umožněn i v rámci API, je dovoleno překrytí těchto metod i na nižší úrovni. V tom případě je však nezbytné znovu zavolat původní metodu z předka, aby bylo zajištěno provedení nezbytného kódu. Konkrétně se jedná o metody

- `decode()`,
- `encodeBegin()`,
- `encodeChildren()`,
- `encodeEnd()`,
- `encodeAll()`.

Tabulka 23 poskytuje seznam veřejně poskytovaných metod této třídy a jejich popis.

Tabulka 23: *Názvy a popis metod třídy com.martinlinha.c3faces.component.C3Chart (Zdroj: autor)*

Název metody	Popis
String getAjaxAction()	vrací řetězec obsahující JavaScript kód pro jsf.js
ComponentProperties getComponentProperties()	vrací odkaz na třídu `ComponentProperties` s kolekcí vlastností grafu, jenž má rozsah platnosti napříč AJAX požadavky
abstract List<Property> getCustomProperties()	abstraktní metoda, jejímž prostřednictvím lze přidat vlastnosti, jež budou připojeny ke komponentě při každé inicializaci.
String getCharType()	název typu grafu dle C3 specifikace
String getCssClass()	CSS třída připojená k div elementu vlastnící graf
String getStyle()	řetězec obsahující `style` atribut připojený k div elementu vlastnící graf
void setStyle(String)	nastaví atribut `style` připojený k div elementu
String getFixedJsVar()	vrací řetězec obsahující název JavaScript proměnné pod kterou je uložen vykreslený graf

Název metody	Popis
void resetListeners()	zavolá metodu `reset()` na posluchačích všech vlastností grafu

Třída com.martinlinha.c3faces.component.ComponentProperties

Třída obsahující mapu objektů reprezentujících vizuální vlastnosti dané komponenty. Jejím prostřednictvím lze modifikovat vlastnosti grafu. Tabulka 24 vystihuje seznam metod této třídy a jejich popis.

Tabulka 24: *Názvy a popis metod třídy com.martinlinha.c3faces.component.ComponentProperties (Zdroj: autor)*

Název metody	Popis
Data getComponentData()	vrací instanci třídy `Data`
boolean addProperty(Property)	přidá požadovaný objekt pouze neexistuje-li v mapě objekt vracející ekvivalentní řetězec z metody Property.getName(). Vrací true, je-li objekt přidán
boolean addProperty(Data)	přidá požadovaný objekt typu `Data` do kolekce vlastností a vrátí true, jestliže předávaný objekt je nová nebo jiná instance než posledně přidaná stejného typu
void addProperties(Collection <Property>)	přidá všechny požadované objekty u kterých platí, že neexistuje v mapě jiný objekt vracející ekvivalentní řetězec z metody `Property.getName()`
Property getProperty(String)	vrací instanci třídy `Property` dle předaného klíče
List<Property> getProperties()	vrací seznam všech objektů typu `Property` dané komponenty
List<PropertyModifier> getPropertyModifier	vrací seznam všech objektů typu `Property`, jež jsou zároveň instancí třídy `PropertyModifier`

7.3.2 Balíček com.martinlinha.c3faces.jsf

Třídy com.martinlinha.c3faces.jsf.C3PartialViewContextFactory a com.martinlinha.c3faces.jsf.C3PartialViewContext

Implementace JSF nazvaná *Mojarra*, kterou používá tento framework, má problémy s přidáváním skriptů do *eval* části z komponent ve fázi *RENDER RESPONSE*. Jsou-li přidávány skripty prostřednictvím metod `startEval()` a `endEval()` v `encode()` metodách komponent, generování AJAX odpovědi skončí neúspěšně.

Prostřednictvím těchto tříd lze přidávat skripty do eval části AJAX odpovědi, jež jsou vypsány až ve správný moment. Odpověď je tedy odeslána úspěšně.

Třída `com.martinlinha.c3faces.jsf.C3PartialViewContextFactory` je registrována v kontextu JSF a slouží k inicializaci třídy `com.martinlinha.c3faces.jsf.C3PartialViewContext`. Soupis jejích metod včetně popisu obsahuje Tabulka 25.

Tabulka 25: *Názvy a popis metod třídy com.martinlinha.c3faces.jsf.C3PartialViewContext (Zdroj: autor)*

Název metody	Popis
static C3PartialViewContext getCurrentInstance()	vrací aktuální instanci třídy `C3PartialViewContext`
void addCallbackScript(String)	přidá skript do kolekce skriptů, jež budou součástí AJAX odpovědi
List<String> getCallbackScripts()	vrací seznam všech skriptů pro aktuálně generovanou AJAX odpověď
void setCallbackScripts(List<String>)	nastaví nový seznam skriptů

7.3.3 Balíček com.martinlinha.c3faces.component.property

Abstraktní třída com.martinlinha.c3faces.component.property.C3Property

Třída pro tvorbu komponent reprezentující vizuální vlastnosti grafu. Tyto komponenty jsou vkládány do těla tagů vykreslující grafy. Jedinou metodu poskytovanou touto třídou zachycuje Tabulka 26 níže.

Tabulka 26: *Názvy a popis metod abstraktní třídy com.martinlinha.c3faces.component.property.C3Property (Zdroj: autor)*

Název metody	Popis
abstract Property getAssociatedProperty()	metoda vracející objekt typu `Property`, jenž bude následně vložen do mapy vlastností komponenty

7.3.4 Balíček com.martinlinha.c3faces.listener

Rozhraní com.martinlinha.c3faces.listener.ChangeListener

Rozhraní implementované třídami, které poslouchají změny určitého vydavatele. Tabulka 27 obsahuje seznammetod a popis tohoto rozhraní.

Tabulka 27: *Názvy a popis metod rozhraní com.martinlinha.c3faces.listener.ChangeListener (Zdroj: autor)*

Název metody	Popis
void onChange(Change<?>)	metoda volaná při notifikaci pozorovatelů vydavatelem předávající objekt typu `Change`

Rozhraní com.martinlinha.c3faces.listener.PropertyModifier

Objekty implementující toto rozhraní jsou ve vhodný okamžik požádány o vygenerování kódu pro C3.js API. Tabulka 28 obsahuje kompletní přehled metod a jejich popis.

Tabulka 28: *Názvy a popis metod rozhraní com.martinlinha.c3faces.listener.PropertyModifier (Zdroj: autor)*

Název metody	Popis
String getScript(String, int)	metoda vracející skript pro C3.js API. Vyžaduje řetězec s názvem JavaScript proměnné a číslo určující dobu trvání animace
int totalDuration()	implementace této metody musí vrátit celkovou dobu trvání všech animací. Používaná v momentě, kdy implementující třída je kompozitní
void reset()	metoda volaná na konci vykreslované komponenty, sloužící typicky k resetování vytvořených objektů

Třídy ObservableSet a C3ViewDataObservableSet

Pro účely zaznamenávání změn, jež se udály nad kolekcí určitého typu objektů, slouží třída `com.martinlinha.c3faces.listener.ObservableSet`. Obaluje kolekci typu `java.util.LinkedHashSet`, aby bylo dodrženo pořadí v jakém byly prvky přidány a implementuje návrhový vzor *Observer*. V momentě, kdy je tato kolekce jakkoliv modifikována, upozorní všechny své posluchače.

Třída `com.martinlinha.c3faces.listener.C3ViewDataObserverableSet` je potomek výše zmíněné třídy a slouží ke sledování změn kolekce objektů typu `com.martinlinha.c3faces.listener.C3ViewDataSet`.

7.3.5 Balíček com.martinlinha.c3faces.listener.change

Třída com.martinlinha.c3faces.component.property.change.Change

Třída reprezentující událou změnu na pozorovateli. Je-li třída kumulativní, může obsahovat kolekci relevantních objektů změn (používáno především k ukládání historie změn provedené na daném objektu). Pro objekty umožňující kumulativní změny je používána inicializace potomka třídy `com.martinlinha.c3faces.listener.change.CumulatibleChange`. Jsou-li změny prováděny nad objekty typu `com.martinlinha.c3faces.model.C3ViewDataSet`, je použita třída `com.martinlinha.c3faces.listener.change.ViewDataSetChange` a `com.martinlinha.c3faces.listener.change.ViewDataSetCumlulatibleChange`.

Tabulka 29 níže obsahuje názvy metod a popis.

Tabulka 29: *Názvy a popis metod třídy com.martinlinha.c3faces.listener.change.Change (Zdroj: autor)*

Název metody	Popis
void addValue(T)	nastaví poslední změnu třídy na hodnotu předanou v parametru a přidá ji do seznamu změn (má především smysl, je-li objekt kumulativní)

Abstraktní třída com.martinlinha.c3faces.component.property.change.Changes

Tato třída slouží pro sběr objektů změn. Jednotlivé změny pak ukládá do mapy. Tabulka 30 obsahuje seznam metod a jejich popis.

Tabulka 30: *Názvy a popis metod abstraktní třídy com.martinlinha.c3faces.component.property.change.Changes (Zdroj: autor)*

Název metody	Popis
Object getPropertyLastChange(String)	metoda prohledá mapu změn podle předaného klíče a pokud se zde objekt nachází, vrátí objekt poslední změny
Set<ViewDataSetChange> getViewDataSetChanges()	vrací podmnožinu všech změn nacházejících se v mapě změn, jenž jsou instancí třídy `ViewDataSetChange`
getPropertyChangeSet(String)	prohledá mapu změn a najde-li objekt s předaným klíčem, vrátí historii jeho změn
onChange(Change<?>)	změna obdržená od vydavatele je nejdříve hledána v mapě změn, je-li nalezena a zároveň je tato změna kumulativní, je přidána do historie změn daného objektu, jinak je vložena jako nová změna

7.3.6 Balíček com.martinlinha.c3faces.script

Abstraktní třída com.martinlinha.c3faces.script.Property

Výchozí třída pro tvorbu vizuálních vlastností grafu. Implementuje návrhový vzor *Composite*, obsahuje tedy kolekci objektů stejného typu. Současně implementuje návrhový vzor *Observer*, na jehož základě vlastní kolekci pozorovatelů typu `com.martinlinha.c3faces.listener.ChangeListener` a vystupuje v roli vydavatele změn.

Vygenerovaný kód touto třídou se skládá z názvu a těla vlastnosti, jenž je odděleno separátorem. Tělo je obaleno prefixem a sufixem. Pokud by prefix a sufix byl vyjádřen znakem lomítka a separátor znakem dvojtečky, vyprodukovaný skript by mohl mít tvar jako kód na Výpisu 30.

Výpis 30: Příklad vygenerovaného kódu s prefixem, sufixem a separátorem (Zdroj: autor)

```
name: /body/
```

Tabulka 31 níže popisuje metody této třídy.

Tabulka 31: *Názvy a popis metod abstraktní třídy com.martinlinha.c3faces.script.Property (Zdroj: autor)*

Název metody	Popis
final String getScript()	metoda vrací dle nastavených atributů vygenerovaný skript, součástí je i název JavaScript objektu dle specifikace C3.js, metoda pomocí rekurze postupně volá stejnou metodu na všech objektech stejného typu z kompozitní kolekce.
final String getScript(boolean)	pracuje stejně jako metoda `final String getScript()`, pomocí předávaného parametru však lze rozhodnout, zda vygenerovaný skript má obsahovat i název JavaScript objektu.
abstract String getPrefix()	prefix, jímž začíná tělo skriptu
abstract String getSuffix()	sufix, jímž končí tělo skriptu
void addChild(Property)	pokud předávaná `Property` nemá prázdné tělo, přidá objekt do kompozitní kolekce
void addChild(Property,boolean)	pokud předávaná `Property` nemá prázdné tělo, přidá objekt kompozitní kolekce a případně smaže všechny `Property` s prázdným názvem
void addChildren(List<Property>)	všechny záznamy z předávaného seznamu přidá prostřednictvím metody `addProperty(Property)` do kolekce kompozitních objektů

Název metody	Popis
void preScriptBuild()	template metoda, která je volána na začátku metody `final String getScript()`
String getSeparator()	separátor mezi názvem a tělem vygenerovaného skriptu
void fire(String, Object)	upozorní všechny zaregistrované pozorovatele a předá jim objekt typu `Change` s předaným řetězcem jako názvem a objektem jako předmětem změny
void fireCumulatible(String, Object)	upozorní všechny zaregistrované pozorovatele a předá jim objekt typu `CumulatibleChange` s předaným řetězcem jako názvem a objektem jako předmětem změny
String getName()	vrací řetězec obsahující název generovaného bloku
void setName(String)	nastaví řetězec s názvem pro generovaného bloku
String getBody()	vrací tělo generovaného bloku
void setBody(String)	nastaví řetězec s tělem pro generovaný blok
void addListener(ChangeListener)	přidá nového pozorovatele typu `ChangeListener` do seznamu pozorovatelů
void addListeners(Collection <ChangeListener>)	přidá všechny pozorovatele typu `ChangeListener` z předané kolekce do seznamu pozorovatelů

Třída com.martinlinha.c3faces.script.ArrayBlock

Třída dědící z `com.martinlinha.c3faces.script.Property`. Pouze překrývá metody `getPrefix()` a `getSuffix()` znaky [a]. Výpis 31 obsahuje příklad vygenerovaného skriptu.

Výpis 31: Příklad skriptu generovaného třídou com.martinlinha.c3faces.script.ArrayBlock (Zdroj: autor)

```
name: [body]
```

Třída com.martinlinha.c3faces.script.CommaValueBlock

Třída dědí z `com.martinlinha.c3faces.script.ValueBlock`. Pomocí překrytí odpovídajících metod generuje kód syntakticky podobný Výpisu 32.

Výpis 32: Příklady skriptu generovaného třídou com.martinlinha.c3faces.script.CommaValueBlock (Zdroj: autor)

```
'name', 'body'
```

Třída com.martinlinha.c3faces.script.MethodBlock

Třída dědící z `com.martinlinha.c3faces.script.Property`. Slouží k výpisu JavaScript funkcí o libovolném počtu parametrů, jež lze předat konstruktoru této třídy. Pokud by parametry měly názvy `param1` a `param2`, kód by vypadal dle Výpisu 33.

Výpis 33: Příklady skriptu generovaného třídou com.martinlinha.c3faces.script.MethodBlock (Zdroj: autor)

```
name: function(param1, param2) { body }
```

Třída com.martinlinha.c3faces.script.ObjectBlock

Třída sloužící ke generování skriptu ve struktuře JSON objektu. Příklad obsahuje Výpis 34 níže.

Výpis 34: Příklady skriptu generovaného třídou com.martinlinha.c3faces.script.ObjectBlock (Zdroj: autor)

```
name: { body }
```

Třída com.martinlinha.c3faces.script.ValueBlock

Objekty tohoto typu dědí z `com.martinlinha.c3faces.script.Property`. Metody třídy nad rámec zděděných shrnuje Tabulka 32. Příklady produkovaného skriptu jsou obsaženy ve Výpisu 35.

Výpis 35: Příklady skriptu generovaného třídou com.martinlinha.c3faces.script.ValueBlock (Zdroj: autor)

```
name: body
name: 'body'
'name': 'body'
```

Tabulka 32: *Názvy a popis metod třídy com.martinlinha.c3faces.script.ValueBlock (Zdroj: autor)*

Název metody	Popis
boolean isNameQuoted()	definuje, zda má být část skriptu s názvem obalena v jednoduchých závorkách
boolean isBodyQuoted()	definuje, zda má být část skriptu s tělem obalena v jednoduchých závorkách

7.3.7 Balíček com.martinlinha.c3faces.util

Třída com.martinlinha.c3faces.util.ComponentUtil

Třída poskytující pomocné statické metody typické pro tvorbu komponent. Jejich výčet společně s popisem obsahuje Tabulka 33.

Tabulka 33: *Názvy a popis metod třídy com.martinlinha.c3faces.util.ComponentUtil (Zdroj: autor)*

Název metody	Popis
static Boolean parseBoolean(Object)	je-li objekt `null`, vrátí `null` jinak přetypuje objekt na `String` a vrátí převedený objekt na `Boolean`
static Integer parseInteger(Object)	je-li objekt `null`, vrátí `null`, jinak přetypuje objekt na `String` a vrátí převedený objekt na `Integer`
static Double parseDouble(Object)	je-li objekt `null`, vrátí `null`, jinak přetypuje objekt na `String` a vrátí převedený objekt na `Double`
static <E extends Enum<E>> E findEnum(Class<E>, String)	v zadané třídě typu `Enum` se pokusí nalézt hodnotu enumerace odpovídající řetězci z parametru

Třída com.martinlinha.c3faces.util.JSBuilder

Tato pomocná třída pomáhá vytvářet různé druhy JavaScript kódu. Jelikož každá metoda vrací referenci stejného objektu na kterém byla volána, lze volání řetězit. Seznam metod společně s jejich popisem je obsažen v Tabulce 34.

Tabulka 34: *Názvy a popis metod třídy com.martinlinha.c3faces.util.JSBuilder (Zdroj: autor)*

Název metody	Popis
static JSBuilder build()	vrací novou instanci třídy
JSBuilder c3()	připojí řetězec „c3“
JSBuilder var(String)	připojí řetězec odpovídající deklaraci proměnné `var` v JavaScriptu, jejíž jméno je řetězec předaný v parametru této metody
JSBuilder endLine()	připojí středník a zalomí řádek
JSBuilder getResult()	vrátí sestavený řetězec
JSBuilder generate(Collection<Property>)	vytvoří řetězec odpovídající volání metody `generate()`. Do jejího těla pak přidá skript vygenerovaný jednotlivou instancí třídy Property z předané kolekce.

Název metody	Popis
JSBuilder setElement(String)	vloží předaný řetězec jako parametr JavaScript funkce `setElement()` a připojí výsledný řetězec
JSBuilder(String, String)	vloží předané řetězce jako parametry JavaScript funkce `append()` a připojí výsledný řetězec

Třída com.martinlinha.c3faces.util.JSTools

Pomocná třída, jenž obsahuje řadu užitečných statických metod pro sestavování řetězců typických pro JavaScript. Jejich výčet a popis obsahuje Tabulka 35.

Tabulka 35: *Názvy a popis metod třídy com.martinlinha.c3faces.util.JSTools (Zdroj: autor)*

Název metody	Popis
static String colonAid(String)	projde řetězec předaný v parametru a před všechny nalezené dvojtečky přidá dva znaky zpětného lomítka
static String jsName(String)	projde řetězec předaný v parametru a všechny nalezené dvojtečky nahradí znakem $
static String commaSeparatedInts(Collection<Integer>)	vytvoří řetězec čísel oddělených čárkou
static String commaSeparatedStrings(Collection<Strings>)	řetězce z předané kolekce oddělí čárkou a vrátí jej jako jeden řetězec
static String commaSeparatedStringsQuoted(Collection <String>)	řetězce z předané kolekce obalí jednoduchými závorkami, oddělí čárkou a vrátí jej jako jeden řetězec
static String semicolonSeparatedStatements(List<String> values)	řetězce z předané kolekce oddělí středníkem, zalomí řádek a vrátí výsledek jako jeden řetězec
static String semicolonSeparatedStatements(String[])	řetězce z předaného pole oddělí středníkem, zalomí řádek a vrátí výsledek jako jeden řetězec
static String semicolonSeparatedModifierScript(List <PropertyModifier>, String)	vygeneruje sekvenci skriptů dle specifikace C3.js, každý skript navíc obalí do metody `setTimeout()` s odpovídající délkou trvání, zároveň připojí název JavaScript proměnné předané jako druhý parametr metody

Název metody	Popis
static String columns(Set<C3ViewDataSet>)	z předané kolekce objektů typu `C3ViewDataSet` vytvoří skript obsahující data dle specifikace C3.js

Třída com.martinlinha.c3faces.util.Faces

Třída obsahující pomocné statické metody pro práci s JSF a tříd z balíčku `com.martinlinha.c3faces.jsf`. Podrobnosti obsahuje Tabulka 36.

Tabulka 36: *Názvy a popis metod třídy com.martinlinha.c3faces.util.Faces (Zdroj: autor)*

Název metody	Popis
static <T> T getContextAttribute (FacesContext, String)	dle předaného názvu vrací odpovídající objekt z kontextových atributů objektu `FacesContext`
static void setContextAttribute (FacesContext, String, Object)	vloží objekt s předaným názvem do kontextových atributů objektu `FacesContext`
static void addCallbackScript (FacesContext, String)	přidá řetězec představující JavaScript volání, jenž bude vyhodnoceno v AJAX odpovědi
static C3PartialViewContext getPartialViewContext(FacesContext)	vrací aktuální instanci typu `C3PartialViewContext`
static Map<String, Object> getSessionMap(FacesContext)	vrací instanci mapy objektů pro aktuální sezení
static Map<String, Object> getViewMap(FacesContext)	vrací instanci mapy objektů pro aktuální *view*
static Map<String, Object> getRequestParameterMap (FacesContext)	vrací instanci mapy objektů pro aktuální požadavek
static boolean isAjaxRequest(FacesContext)	vrací `true`, pokud je aktuální požadavek typu AJAX

7.3.8 JavaScript API

Je-li komponenta z knihovny C3Faces obsažena v dané Facelet stránce, je ve vygenerovaném HTML kódu automaticky přidán odkaz na JavaScript soubor *c3faces.js*. Ten obsahuje dvě JavaScript funkce popsané v Tabulce 37.

Tabulka 37: *Názvy a popis funkcí souboru c3faces.js (Zdroj: autor)*

Název metody	Popis
function setElement(jsVar)	na předané JavaScript proměnné se pokusí nalézt parametr `element`, jenž dle specifikace C3.js obsahuje odkaz na vygenerovaný `svg` element reprezentující graf
function append(divId, jsVar)	JavaScriptová funkce, která připojí k předanému `div` elementu graf, jehož odkaz je obsahem JavaScriptové proměnné z druhého parametru této funkce
function getElement(jsVar)	vrací odkaz na DOM objekt obsahující `svg` element s grafem

8 Závěr

V této diplomové práci se zaměřuji na vývoj komponent pro vizualizaci dat v prostředí JavaServer Faces. Její motivace vznikla v době, kdy jsem v mé profesní praxi obdržel zadání na implementaci komplexních grafů v projektu postaveném na JavaServer Faces. Bohužel jsem nenalezl knihovnu komponent uspokojující kladené požadavky. V souvislosti s řešením tohoto problému jsem se seznámil s JavaScript knihovnou C3.js, jenž takto komplexní grafy umožňuje vytvářet.

Tyto důvody vedly k definici cílů této práce. První cíl spočívá ve vývoji frameworku postaveným nad JSF pro tvorbu komplexních grafů z knihovny C3.js. Jeho prostřednictvím by pak bylo možné efektivně přenášet tyto JavaScriptem generované grafy do komponent JavaServer Faces. Druhým cílem je implementace základní sady JSF komponent realizována ve vytvořeném frameworku. Vývojář by tedy měl k dispozici okamžitě využitelnou JSF knihovnu přepoužitelných komplexních grafů.

Prvním krokem k dosažení těchto cílů je krátká rešerše. Její snahou je ověřit, zda podobné řešení již nebylo vyvinuto. Následně jsem stanovil požadavky, jež by měly být splněny jak vznikajícím frameworkem, tak knihovnou komponent. Jelikož produktem všech těchto JSF komponent je C3.js kód, analyzována byla především jeho syntaxe a struktura. Pokračoval jsem návrhem architektury programu a návrhem API frameworku. Poté jsem implementoval API pro tvorbu vizuálních komponent. Za pomocí vzniklého API jsem vytvořil základní sadu komponent včetně objektů definujících vizuální vlastnosti grafů a objektů pro jejich částečné modifikace. Součástí práce je samozřejmě uživatelská příručka sloužící pro uživatele knihovny komponent a současně dokumentace k API.

Výsledkem této diplomové práce je tedy vyvinuté API, jehož prostřednictvím lze vytvářet JSF komponenty založené na JavaScript knihovně C3.js. Současně jsem v tomto API implementoval komponenty pokrývající všechny typy C3.js grafů a komponenty sloužící k deklaraci jejich vizuálních atributů. JSF vývojář tedy může ihned implementovat C3.js grafy do svého projektu. Celý produkt je zabalen v jednom archivu, jenž lze nalézt na *http://www.github.com/martin-linha/c3faces*.

Mohu konstatovat, že se mi podařilo úspěšně naplnit cíle stanovené touto prací. V kapitole věnující se analýze je specifikováno celkem 16 požadavků. V textu práce na ně postupně odkazuji a lze tedy snadno nalézt text relevantní k jejich realizaci. Ukázky vyvinutých komponent společně s demonstrací komplexních vlastností a složitých animací jsou k naleznutí na prezentačním webu na adrese *http://c3faces.martinlinha.com*.

Přínosy práce jsou reprezentovány vzniklým frameworkem. Podařilo se mi zpřístupnit komplexní grafy obsažené v JavaScript knihovně C3.js pro vývojáře v JavaServer Faces. Ti, bez nutnosti znát C3.js, mohou začít do svých webových projektů přidávat tyto grafy

deklarativním způsobem prostřednictvím JSF komponent. Zároveň mohou použít vzniklé API a vytvářet tak nové komponenty z C3.js či jejich vizuální atributy.

Terminologický slovník

Termín	Zkratka	Význam [zdroj]
Anotace		představují metadata k různým částem kódu pro specifikaci jejich vlastností
Application Programming Interface	API	rozhraní pro programování aplikací
Asynchronous JavaScript and XML	AJAX	technologie pro tvorbu interaktivních webových aplikací za pomoci asynchronního zpracování
Cascade Style Sheet	CSS	popisují způsob zobrazení elementů v HTML stránkách
Controller		zajišťuje změny ve view nebo datovém modelu
Expression Language	EL	jazyk pro odkazování na Java objekty z prezentační vrstvy
Facelets		open source framework představený společně s JSF 2.0 pro tvorbu prezentační vrstvy (nahrazuje JSP)
Framework		soubor technologií poskytující nadstavbu nad daným systémem a zjednodušující jeho funkcionalitu
Hypertext Markup Language	HTML	značkovací jazyk pro tvorbu webových stránek
Hypertext Transfer Protocol	HTTP	protokol pro výměnu dokumentů ve formátu HTML
Java Community Process	JCP	proces, jehož prostřednictvím se vyvíjí technické standardy a specifikace pro Java technologie
Java Enterprise Edition	J2EE	součást platformy Java pro vývoj podnikových aplikací
Java Specification Request	JSR	je součástí JCP a jedná se o strukturovaný požadavek na novou funkcionalitu, či změnu již implementované funkcionality Java technologií
JavaScript		skriptovací jazyk pro webové prohlížeče
JavaServer Pages	JSP	technologie používaná pro tvorbu prezentační vrstvy Java webových aplikací
Jmenný prostor		angl. Namespace, určuje unikátní identifikátor knihovny tagů
Model-View-Controller	MVC	návrhový vzor oddělující datový model, prezentační vrstvu a řídicí logiku

Termín	Zkratka	Význam [zdroj]
Observer		návrhový vzor, na jehož základě může objekt odposlouchávat změny provedené na objektu jiném
Rozsah platnosti		angl. Scope, definuje časový úsek, po který je objekt k dispozici
Scalable Vector Graphics	SVG	následovník jazyka VML, jedná se o značkovací jazyk pro 2D vektorovou grafiku
Servlet		serverová komponenta obhospaduřící příchozí HTTP požadavky
Sezení		angl. Session, udržuje stav objektu napříč HTTP požadavky pro jednoho uživatele
Tag		základní komponenta značkovacích jazyků
Template metoda		metoda z návrhového vzoru Template, kdy objekt vyžaduje implementaci určité abstraktní metody, kterou následně volá z jiné, privátní metody
Vector Markup Language	VML	značkovací jazyk založený na XML pro definici vektorové grafiky na webové stránce
View		reprezentuje jednu webovou stránku aplikace
Závislost		angl. Dependency, stav, kdy je komponenta závislá na jiné

Seznam literatury

[1] DEWAR, Mike a With Neil GRIFFIN. Getting started with D3. 1. ed. Sebastopol, CA: O'Reilly Media, 2012, xiii, 255 p. ISBN 978-144-9328-795.

[2] KOŘISTKOVÁ, Barbora. *Framework RichFaces*. Praha, 2014. Diplomová práce. Vysoká škola ekonomická v Praze.

[3] MAČOUN, Jakub. *Tvorba webových aplikací v JSF PrimeFaces*. Praha, 2012. Bakalářská práce. Vysoká škola ekonomická v Praze.

[4] ANGHEL, Leonard. *Mastering JavaServer Faces 2.2*. Birmingham (Velká Británie): Packt Publishing Ltd., 2014. ISBN 978-1-78217-646-6.

[5] ED BURNS, Chris Schalk a With Neil GRIFFIN. JavaServer faces 2.0 the complete reference. New York: McGraw-Hill, 2010. ISBN 978-007-1625-104.

[6] RYCHTECKÝ, Lukáš. Framework pro návrh responzivního uživatelského rozhraní nové generace. Praha, 2011. Diplomová práce. České vysoké učení technické v Praze.

[7] ORACLE CORPORATION. *The Java EE Tutorial: The Web Tier* [online]. 2013 [cit. 2015-02-26]. Dostupné z: https://docs.oracle.com/javaee/7/tutorial/partwebtier.htm

[8] GEARY, David. Templating and composite components. *IBM developerWorks* [online]. 2010, 5 [cit. 2015-03-11]. Dostupné z: http://www.ibm.com/developerworks/java/library/j-jsf2fu2/j-jsf2fu2-pdf.pdf

[9] GEARY, David. After-the-fact Ajax composite components. *IBM developerWorks* [online]. 2010, č. 4 [cit. 2015-03-21]. Dostupné z: http://www.ibm.com/developerworks/library/j-jsf2fu-0610/j-jsf2fu-0610-pdf.pdf

[10] GEARY, David. Ajax components. *IBM developerWorks* [online]. 2010, č. 3 [cit. 2015-03-13]. Dostupné z:http://www.ibm.com/developerworks/java/library/j-jsf2fu-0410/j-jsf2fu-0410-pdf.pdf

[11] *C3.js | D3-based reusable chart library - Reference* [online]. 2013 [cit. 2015-02-25]. Dostupné z:http://c3js.org/reference.html

[12] Composite component with multiple input fields. In: SCHOLTZ. *The BalusC Code* [online]. 2013 [cit. 2015-04-01]. Dostupné z: http://balusc.blogspot.cz/2013/01/composite-component-with-multiple-input.html

[13] IFFLAND, David. C3.js Brings Charting Power Without the Learning Curve. In: *InfoQ: Software development news* [online]. 2014 [cit. 2015-02-24]. Dostupné z: http://www.infoq.com/news/2014/09/c3js-d3-charting

[14] MURRAY, Scott. Interactive data visualization for the web. 1st ed. Sebastopol, CA: O'Reilly Media, 2013, xiii, 255 p. ISBN 1449361080.

[15] GEARY, David M a Cay S HORSTMANN. Core JavaServer faces. 3rd ed. Upper Saddle River, NJ: Prentice Hall, c2010, xx, 636 p. ISBN 0137012896.

[16] ÇALIŞKAN, Mert a Oleg VARAKSIN. PrimeFaces cookbook: over 90 practical recipes to learn PrimeFaces - the rapidly evolving, leading JSF component suite. Birmingham: Packt Pub., 2013, iv, 312 p. Community experience distilled.

[17] How to Write Your Own JSF Components. *Exadel, Inc.* [online]. 2009, č. 1 [cit. 2015-03-13]. Dostupné z:http://www.exadel.com/tutorial/jsf/HowToWriteYourOwnJSFComponents.pdf

[18] *OmniFaces VDL Documentation* [online]. 2012 [cit. 2015-03-30]. Dostupné z:http://omnifaces.org/docs/vdldoc/2.0/

[19] *Flot: Attractive JavaScript plotting for jQuery* [online]. 2012 [cit. 2015-03-07]. Dostupné z:http://www.flotcharts.org/

[20] RichFaces Documentation. *JBoss Developer* [online]. 2010 [cit. 2015-03-27]. Dostupné z:http://richfaces.jboss.org/docs

[21] ÇIVICI, Çağatay. PrimeFaces - User Guide 5.2. *PrimeTek Informatics* [online]. 2015, First edition [cit. 2015-03-27]. Dostupné z: http://www.primefaces.org/docs/guide/primefaces_user_guide_5_2.pdf

[22] SALEH, Hazem, Allan Lykke CHRISTENSEN a Zubin WADIA. Pro JSF and HTML5: building rich internet components. Second edition. xvii, 391 pages. Expert's voice in Java. ISBN 9781430250104.

[23] Custom components. In: DE BUSSCHER, Rudy. *JSF Corner* [online]. 2011 [cit. 2015-03-15]. Dostupné z:http://jsfcorner.blogspot.cz/2011/01/custom-components.html

[24] GEARY, David. Event handling, JavaScript, and Ajax. *IBM developerWorks* [online]. 2009, Part three [cit. 2015-04-11]. Dostupné z: http://www.ibm.com/developerworks/java/library/j-jsf2fu3/j-jsf2fu3-pdf.pdf

[25] Creating charts with C3.js. In: VAN DEN LINDEN, Roberto. *TRIFORK* [online]. 2014 [cit. 2015-03-23]. Dostupné z: https://blog.trifork.com/2014/07/29/creating-charts-with-c3-js/

[26] *AJAX JSF Matrix* [online]. 2006 [cit. 2015-02-23]. Dostupné z: http://www.jsfmatrix.net/

[27] Component libraries. *JSFCentral.com* [online]. 2002 [cit. 2015-02-21]. Dostupné z:http://www.jsfcentral.com/products/components/

[28] JqPlot Usage. *JqPlot Charts and Graphs in jQuery* [online]. 2009 [cit. 2015-02-21]. Dostupné z:http://www.jqplot.com/docs/

[29] ICEfaces Documentation, Release notes and Guides. *ICEsoft Technologies* [online]. 2011 [cit. 2015-03-18]. Dostupné z: http://www.icesoft.org/java/projects/ICEfaces/documentation.jsf

[30] *Apache MyFaces* [online]. 2006 [cit. 2015-03-11]. Dostupné z: http://myfaces.apache.org/

[31] Fiji Framework. *Exadel* [online]. 2010 [cit. 2015-03-22]. Dostupné z: http://livedemo.exadel.com/fiji-demo/

[32] *JSflot - A JavaServer Faces JSF chart library* [online]. 2009 [cit. 2015-03-27]. Dostupné z: https://code.google.com/p/jsflot/

[33] *Flot4JSF2 - Charting components for JSF 2.0* [online]. 2009 [cit. 2015-03-27]. Dostupné z: https://code.google.com/p/flot4jsf2/

[34] HALL, Marty, Larry BROWN a Yaakov CHAIKIN. Core servlets and JavaServer pages. 2nd ed. Upper Saddle River, NJ: Prentice Hall PTR, c2004-c2008, 2 v. ISBN 97801314826092.

Seznam obrázků a tabulek

Seznam obrázků

Obrázek 1 : Architektura JSF (Zdroj: [6]) 9
Obrázek 2 : Komponentový model JavaServer Faces (Zdroj: [5]) 10
Obrázek 3 : Životní cyklus zpracování požadavku v JSF (Zdroj: [7]) 11
Obrázek 4 : Vygenerovaný graf za pomoci kódu z Výpisu 1 (Zdroj: autor) 14
Obrázek 5 : Upravený zdrojový kód stránky odpovídající grafu na Obrázku 4 (Zdroj: autor) 15
Obrázek 6 : Upravený graf z Obrázku 4 za pomoci C3.js API (Zdroj: autor) 18
Obrázek 7 : Architektura a komunikace mezi komponentami vyvíjeného programu (Zdroj: autor) 22
Obrázek 8 : Diagram aktivit pro generování skriptu konstrukce a modifikací grafu (Zdroj: autor) 24
Obrázek 9 : API pro tvorbu JSF komponent (Zdroj: autor) 30
Obrázek 10 : API pro tvorbu objektů vizuálních vlastností a modifikací (Zdroj: autor) 33
Obrázek 11 : Vygenerovaný graf za pomoci kódu z Výpisu 15 (Zdroj: autor) 40
Obrázek 12 : Integrované C3Faces grafy v komponentách PrimeFaces (Zdroj: autor) 82

Seznam tabulek

Tabulka 1: Seznam komplexních JSF knihoven, poskytujících komponenty pro vykreslování grafů, a názvy JavaScript knihoven vykreslujících grafy (Zdroj: autor) 4
Tabulka 2: Seznam JSF knihoven zaměřených na vizualizaci dat a podpůrných JavaScript knihoven (Zdroj: autor) 5
Tabulka 3: Seznam požadavků na framework (Zdroj: autor) 18
Tabulka 4: Seznam požadavků na komponentovou sadu (Zdroj: autor) 19
Tabulka 5: Názvy tříd pro šablony generovaných skriptů a příklady užití (Zdroj: autor) 31
Tabulka 6: Názvy implementovaných grafů a odpovídající tagy pro Facelet stránky (Zdroj: autor) 35
Tabulka 7: Názvy implementovaných vizuálních vlastností a odpovídajících tagů (Zdroj: autor) 36
Tabulka 8: Popis atributů pro tag barProperties (Zdroj: autor) 42
Tabulka 9: Popis atributů pro tag donutProperties (Zdroj: autor) 42
Tabulka 10: Popis atributů pro tag gaugeProperties (Zdroj: autor) 43
Tabulka 11: Popis atributů pro tagy gridX a gridY (Zdroj: autor) 43
Tabulka 12: Popis atributů pro tag legend (Zdroj: autor) 44
Tabulka 13: Popis atributů pro tag padding (Zdroj: autor) 45
Tabulka 14: Popis atributů pro tag pieProperties (Zdroj: autor) 45
Tabulka 15: Popis atributů pro tag point (Zdroj: autor) 45

Tabulka 16: Popis atributů pro tag region (Zdroj: autor).....46
Tabulka 17: Popis atributů pro tag size (Zdroj: autor).....46
Tabulka 18: Popis atributů pro tag subchart (Zdroj: autor).....47
Tabulka 19: Popis atributů pro tag tooltip (Zdroj: autor).....47
Tabulka 20: Popis atributů pro tag zoom (Zdroj: autor) 48
Tabulka 21: Seznam tříd vlastností a modifikátorů v C3Faces (Zdroj: autor).....51
Tabulka 22: Šablony pro generování rozdílných typů skriptů společně s příklady (Zdroj: autor)55
Tabulka 23: Názvy a popis metod třídy com.martinlinha.c3faces.component.C3Chart (Zdroj: autor) 59
Tabulka 24: Názvy a popis metod třídy com.martinlinha.c3faces.component.ComponentProperties (Zdroj: autor).....60
Tabulka 25: Názvy a popis metod třídy com.martinlinha.c3faces.jsf.C3PartialViewContext (Zdroj: autor) 61
Tabulka 26: Názvy a popis metod abstraktní třídy com.martinlinha.c3faces.component.property.C3Property (Zdroj: autor).....61
Tabulka 27: Názvy a popis metod rozhraní com.martinlinha.c3faces.listener.ChangeListener (Zdroj: autor) 62
Tabulka 28: Názvy a popis metod rozhraní com.martinlinha.c3faces.listener.PropertyModifier (Zdroj: autor).....62
Tabulka 29: Názvy a popis metod třídy com.martinlinha.c3faces.listener.change.Change (Zdroj: autor) 63
Tabulka 30: Názvy a popis metod abstraktní třídy com.martinlinha.c3faces.component.property.change.Changes (Zdroj: autor).....63
Tabulka 31: Názvy a popis metod abstraktní třídy com.martinlinha.c3faces.script.Property (Zdroj: autor).....64
Tabulka 32: Názvy a popis metod třídy com.martinlinha.c3faces.script.ValueBlock (Zdroj: autor)66
Tabulka 33: Názvy a popis metod třídy com.martinlinha.c3faces.util.ComponentUtil (Zdroj: autor) 67
Tabulka 34: Názvy a popis metod třídy com.martinlinha.c3faces.util.JSBuilder (Zdroj: autor)67
Tabulka 35: Názvy a popis metod třídy com.martinlinha.c3faces.util.JSTools (Zdroj: autor)68
Tabulka 36: Názvy a popis metod třídy com.martinlinha.c3faces.util.Faces (Zdroj: autor) 69
Tabulka 37: Názvy a popis funkcí souboru c3faces.js (Zdroj: autor) 70

Příloha A: Systémové požadavky

Implementace frameworku je založena na následujících technologiích

Název	Verze
Java	1.7
JSF specifikace	2.2.8
Mojarra (JSF implementace)	2.2.8
C3.js	0.4.10
D3.js	3.5.5

Příloha B: Integrace s frameworkem PrimeFaces

Komponenty z knihovny C3Faces jsou integrovatelné s komponentami v současné době nejpopulárnějšího JSF frameworku *PrimeFaces*. Následující Výpis 36, z kterého je patrná kombinace tagů z jmenných prostorů `xmlns:c3="http://c3faces.martinlinha.com/ns"` a `xmlns:p="http://primefaces.org/ui"`, vykreslí komponenty z Obrázku 12 níže.

Výpis 36: Facelet zahrnující integraci C3Faces komponent s PrimeFaces

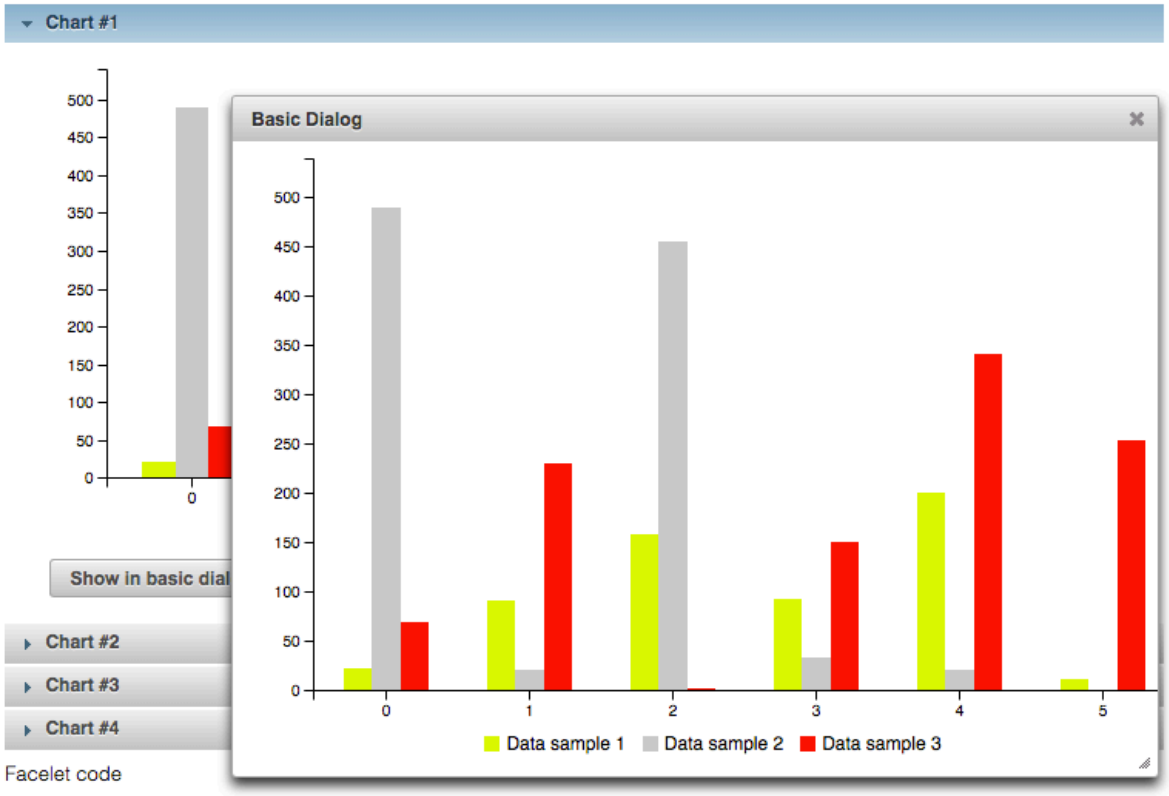
```
<h:form>
    <p:accordionPanel >
       <p:tab title="Chart #1">
          <c3:bar id="bar" data="#{primefacesIntegrationBean.data1}"
value="#{primefacesIntegrationBean.selectedDataSet}">
                            <f:valueChangeListener
type="com.martinlinha.showcase.listener.DataChangeValueListener" />
                            <p:ajax update="msg"
listener="#{primefacesIntegrationBean.showMessageDialog}"/>
          </c3:bar>
          <h:panelGrid columns="3">
             <p:commandButton value="Show in basic dialog" type="button"
onclick="PF('dlg1').show();" />
             <p:commandButton value="Show in dialog with effects" type="button"
onclick="PF('dlg2').show();" />
          </h:panelGrid>
          <p:message id="msg" for="bar" />

          <p:dialog header="Basic Dialog" widgetVar="dlg1" height="400"
width="600">
             <c3:bar id="bar1" data="#{primefacesIntegrationBean.data1}"
style="width: 600px; height: 400px;"/>
          </p:dialog>

          <p:dialog header="Effects" widgetVar="dlg2" showEffect="explode"
hideEffect="bounce"  height="400" width="600">
             <c3:bar id="bar3" data="#{primefacesIntegrationBean.data1}"
style="width: 600px; height: 400px;"/>
          </p:dialog>

       </p:tab>
       <p:tab title="Chart #2">
          <c3:pie data="#{primefacesIntegrationBean.data2}" />
       </p:tab>
       <p:tab title="Chart #3">
          <c3:gauge data="#{primefacesIntegrationBean.data3}" />
```

```
        </p:tab>
        <p:tab title="Chart #4">
           <c3:spline data="#{primefacesIntegrationBean.data4}" />
        </p:tab>
     </p:accordionPanel>
</h:form>
```



Obrázek 12 : *Integrované C3Faces grafy v komponentách PrimeFaces (Zdroj: autor)*

Tento příklad je současně k dispozici na adrese *http://c3faces.martinlinha.com/faces/primefaces_integration.xhtml*.

Příloha C: Zdrojové kódy

Zdrojové kódy implementované knihovny jsou k dispozici na přiloženém CD.