ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačů

Bakalářská práce

Proxy pro automatické generování uživatelského rozhraní

Jan Jaroš
Softwarové inženýrství a technologie

Květen 2019
Vedoucí práce: Ing. Jiří Šebek

Poděkování / Prohlášení

Rád bych poděkoval svému vedoucímu práce Ing. Jiřímu Šebkovi za cenné rady a připomínky při vedení mé bakalářské práce. Také bych rád poděkoval své rodině za jejich obrovskou podporu během studia a také svému dobrému kamarádovi Jiřímu Košatovi, který mi vždy dokázal dobře poradit během studia na vysoké škole.

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 18. 5. 2019

.....

Abstrakt / Abstract

Práce se zabývá úpravou již hotového frameworku na generování uživatelského rozhraní pomocí metadat na novější a modernější přístup, kdy se rozšiřuje XML konfigurace o JSON tak, aby fungovalo stávající řešení frameworku. Je provedena rešerše frameworku nazvaného AspectFaces, který umožňuje generovat uživatelské rozhraní z datového modelu aplikace. Tyto grafické elementy jsou pak naplněné daty od uživatele/databáze nebo jsou využívány ke sběru uživatelských dat pomocí formulářů. Dále práce vyhodnocuje reálné použití na několika vývojářích a v neposlední řadě je provedeno několik testů, které ověřují správnou funkčnost. Na závěr práce je úprava frameworku změřena a její výsledky jsou náležitě okomentovány.

Klíčová slova: aspectfaces, framework, generování UI, uživatelské rozhraní, json, xml, metadata.

The work deals with the modification of already finished framework for generating user interface using metadata to a newer and more modern approach when the XML configuration is extended by JSON so that the existing framework solution works. The framework called AspectFaces, which allows generating a user interface from the application data model, was researched. These graphic elements are then filled with data from the user/database or used to collect user data using forms. Furthermore, the work evaluates the real use on several developers and last but not least there are several tests that verify correct functionality. At the end of the work, the framework is measured and its results are properly commented.

Keywords: aspectfaces, framework, generating UI, user interface, json, xml, metadata.

Obsah /

1 Úvod ... 1
1.1 Motivace ... 1
1.2 Programování uživatelského rozhraní ... 1
1.3 Cíl práce ... 2
2 Rešerše ... 3
2.1 Metamodeling ... 3
2.2 Model Driven Development ... 3
2.3 Proxy ... 3
2.4 Automatické generování uživatelského rozhraní ... 4
2.5 Aspect Oriented Programming ... 4
2.6 XML ... 4
2.7 JSON ... 4
2.8 REST ... 4
2.9 JSF ... 5
2.10 Gson ... 5
2.11 Jackson ... 5
3 Analýza ... 6
3.1 AFSwinx ... 6
3.2 AFRest ... 7
3.3 AspectFaces ... 8
3.4 Způsob parsování konfiguračních souborů ... 9
4 Specifikace požadavků ... 11
4.1 Funkční požadavky ... 11
4.2 Nefunkční požadavky ... 11
5 Implementace ... 12
5.1 Použité nástroje k vývoji ... 12
5.2 Rozvětvení ... 12
5.3 Měření rychlostí ... 13
5.4 Způsob identifikace konfigurace ... 15
5.5 Není proxy jako proxy ... 15
5.6 Případy užití ... 16
5.6.1 Aktéři ... 16
5.6.2 Developer ... 16
5.7 Architektura ... 18
5.8 Nutné úpravy ve frameworku ... 19
6 Testování ... 20
6.1 Unit testy ... 20
6.1.1 Unit test case ... 20
6.2 Integrační testy ... 21
6.2.1 Test case integračních testů ... 21
6.3 Testování rychlosti tvorby konfiguračních souborů ... 21
7 Závěr ... 23
7.1 Budoucí vylepšení ... 23
Literatura ... 24
A Zadání práce ... 27
B Zkratky a symboly ... 29
B.1 Zkratky ... 29
C Obsah přiloženého CD ... 30

Tabulky / Obrázky

5.1. Naměřené časy s org.json 14
5.2. Naměřené časy s Gson 15
5.3. Atributy konfigurace 19
6.1. Naměřené časy XML a JSON konfigurací 22

3.1. Definice zdrojů 6
3.2. Vytvoření tabulky v AFSwinx ... 7
3.3. Příklad structure.config.xml 7
3.4. Příklad definované komponenty 8
3.5. Struktura AspectFaces 8
3.6. Struktura Configuration Holders 10
3.7. Struktura Configuration Parsers 10
5.1. Nová struktura Configuration Parser 12
5.2. Maven závislost org.json 13
5.3. Způsob měření rychlosti v prohlížeči 13
5.4. Registrace konfiguračních souborů 14
5.5. Maven závislost Gson 14
5.6. Příklady užití frameworku 16
5.7. Konfigurační část aspectfaces-config.xml souboru 18
5.8. Nový návrh JSON konfigurace . 18

Kapitola 1
Úvod

Počítačové technologie jdou každým dnem kupředu. Není se proto čemu divit, když vývojář používá nové technologie a občas musí sáhnout po starších technologiích. Tyto technologické anomálie ovšem nejsou nic příjemného. Kolikrát se dělají kompromisy, ať už ve výkonu, rychlosti nebo třeba v tempu vývoje. Jedná se tedy o modifikaci nebo o rozšíření stávajících frameworků pomocí nových přístupů a technologií. Nejdříve je potřeba se seznámit s problémem a potom na rozšíření pracovat. Pomocí analýzy zjistit místa vylepšení a z ní pak přejít do implementace. Z návrhu udělat funkční práci a tu v sekci testování otestovat a vyhodnotit výsledky.

1.1 Motivace

JSON formát je velice oblíbený mezi programátory. Je to především proto, že se lépe čte, snadněji se udržuje a v neposlední řadě je také méně náročný na výkon [1]. Výhod JSON oproti XML je několik. Jelikož je XML značkovací jazyk, tak je potřeba být obeznámený s používáním XML formátu. Jedna z věcí, kterou je dobré mít na paměti je ta, abychom si uvědomili pro jaké účely naší aplikaci píšeme. XML je specifický v několika charakteristických znacích. Jedním z nich je například ten, že je potřeba zprávy validovat, jestli tam nejsou špatné znaky a chyby. Další věcí, kterou je potřeba brát v potaz je ta, že obsahuje spoustu znaků, které vlastně ani nejsou data, ale jen XML jako takové. To také souvisí s tím, že se taková data musí přenášet po síti. Lze tedy mluvit o tom, že se přenáší data, která vůbec nepotřebujeme a při využívání rychlých technologií může být použití XML značně pomalé, což není žádoucí. Je neustálý zájem o rychlost naší aplikace a poté použití technologie, která takovou aplikaci akorát zpomaluje je nadmíru kontraproduktivní. Když už se k nám data v XML dostanou je potřeba tento soubor rozparsovat a nějakým způsobem opět přenést důležitá data, což v případě XML je náročné na procesor [1]. XML má velký přínos v oblasti excel dokumentů a při práci s XSLT soubory. JSON oproti tomu je lépe čitelný jelikož nemáme tolik znaků použito pro tagy. Jedná se totiž o jazyk, který je založen na key-value párech. JSON bývá zpravidla menší na disku a není tak náročný na procesor. Nespornou výhodou pro vývojáře je, že JSON formát je velice snadno čitelný člověkem, to usnadňuje a urychluje vývoj aplikace [2]. Tato dobrá čitelnost zůstává i při vnořování více objektů do sebe a při vytváření struktur v JSON. Lze tedy snadno a čitelně tvořit větší objekty, struktury a pole.

1.2 Programování uživatelského rozhraní

Programování uživatelského rozhraní zabírá mnoho času vývojářům. Tento čas lze ovšem použít na rozšíření funkcionalit produktu než na vypisování a zobrazování jednoduchých informací jako je třeba dynamické zobrazení dat do tabulky nebo poslání informací z formuláře na server, kde jsou poté informace dále zpracovány. K tomuto účelu slouží rodina frameworků z AspectFaces [3]. Konkrétně jde o AFRest a AFSwinx, které jsou postaveny nad AspectFaces. Všechny tyto frameworky využívají ke generování definic widgetů a dat soubory ve formátů .xml.

1.3 Cíl práce

Cílem práce je úprava stávajícího frameworku, který generuje uživatelské rozhraní na základě metadat. Nejdříve je potřeba celou problematiku zanalyzovat. Z analýzy poté vzniknou návrhy řešení, která budou patřičně odiskutovaná a zvolí se nejlepší možné podložené testováním pro každou možnou implementaci. Na závěr jsou provedeny testy rychlostí a zhodnocení celé práce.

Kapitola 2
Rešerše

V této části práce jsou představeny technologie používané frameworkem nebo jeho novou úpravou. Jedná se o ty nejzákladnější technologie, které framework používá. Jsou zde představeny i různé programovací přístupy a knihovny z kterých se skládá tato práce.

2.1 Metamodeling

Předpona meta- je v dnešní době spojována s různými IT pojmy. Např. metamodeling, metaprogramování, metamodel, metadata a další. Toto slovo pochází z řečtiny a v IT světě znamená o jeden stupeň abstrakce výše. Pokud tedy mluvíme o metamodelu, myslíme tím model popisující model, metadata jsou data popisující data [4]. Metamodeling se využívá při tvorbě nových metodologií. Každá metodologie je popsána metamodelem, který je popsán buď formálně nebo neformálně. Metamodeling také obsahuje prostředky k nalezení společných vlastností různých metodologií a hledá způsoby jejich vzájemných kooperací.

2.2 Model Driven Development

Již od počátku softwarového inženýrství se lidé snažili o co největší abstrakci problémů. Pomocí objektově orientovaného programování se programátorům daří uchopit problémy dnešního světa v takové míře, že se dokáží vypořádat s komplexitou světa kolem nás, což se dříve bez takto abstraktních přístupů dělalo velmi složitě až nemožně. V tomto duchu pokračuje i Model Driven Development (MDD) [5], který tuto abstrakci ještě více prohlubuje. Místo nutnosti specifikování každého detailu vývojářem pomocí programovacího jazyka umožňuje vymodelovat jaká funkcionalita je vyžadována a jakou architekturu by systém měl mít. V současnosti překladače umí řešit takové problémy jako alokace objektů, vyhledávání metod a zpracování výjimek, které byly jen před pár lety programovány ručně. Díky neustálému zvyšování úrovně abstrakce MDD cílí na automatizaci mnoha programovacích úkonů.

2.3 Proxy

Proxy je v podstatě prostředník mezi dvěma stranami. V mém případě se bude proxy starat o konverzi Java objektů na JSON, se kterým pak bude framework pracovat místo současně používaného XML formátu. Proxy zajistí to, aby se programátor nemusel starat o přepínání způsobu konfigurace frameworku, ale aby proxy za něj samo rozhodlo jakým způsobem bude s konfigurací zacházeno [6].

2.4 Automatické generování uživatelského rozhraní

Uživatelské rozhraní tvoří nedílnou součást moderních systémů, které nějakým způsobem interagují s uživatelem. Dobré uživatelské rozhraní provádí uživatele aplikací. Jednoduše řečeno lze hovořit o posloupnosti obrazovek, rozhraní, tlačítek, ikonek a dalších uživatelských elementů. Není překvapením, že se společnosti snaží pomocí uživatelského rozhraní přilákat nové uživatele a umožnit jim tak jedinečný zážitek z používání jejich aplikace. Vytvoření dobrého uživatelského rozhraní však není rychlá záležitost na implementaci, a proto tvorba uživatelského rozhraní tvoří zhruba 50 % [7] času vývoje konkrétního softwaru. Tato hodnota se samozřejmě může měnit v závislosti na velikost a na účelu softwaru. K urychlení vývoje uživatelského rozhraní lze použít nějaký z několika frameworků, který tento proces umožňují. Jeden z nich je i framework AspectFaces, který na základě meta-model inspekce z data model struktury vytvoří statickou nebo dynamickou integraci několika aspektů do jednoho výstupu.

2.5 Aspect Oriented Programming

Jedná se o programovací paradigma, které doplňuje objektově orientované programování oddělením zodpovědností od kódu za účelem zlepšení modularizace softwaru. Oddělení zodpovědnosti (separation of concerns – SoC) má za cíl usnadnit správu softwaru tím, že seskupí prvky a chování do lehce spravovatelných částí, které mají konkrétní účel a logiku o kterou se starají [8].

2.6 XML

Extensible Markup Language, zkráceně XML je obecný značkovací jazyk, který se používá k výměně dat a informací mezi aplikacemi. XML 1.0 byl představen koncem roku 2008. Syntaxe XML je striktně určená a tím jsou tagy („<“, „/>“), ve kterých jsou názvy atributů a ohraničují hodnotu atributu. Dále musí mít pouze jeden kořenový (root) element [9]. XML používá ke konfiguraci frameworků, databází, serverů, apod.

2.7 JSON

JSON, neboli JavaScript Object Notation je formát pro výměnu dat, textový a nezávislý na programovacím jazyce [10]. Je velice vhodný pro výměnu dat mezi aplikacemi, serverem a klientem a to především díky jeho snadné čitelnosti a oblibě mezi vývojáři různých programovacích jazyků. Jeho obliba je dána především díky zápisu dat, která jsou zapisována v polích nebo agregována v objektech.

2.8 REST

Zkratka REST znamenající Representational State Transfer [11] je způsob, jak pomocí HTTP požadavků vytvořit, smazat, editovat nebo číst informace ze serveru. Pomocí tohoto konceptu lze lehce přistupovat ke zdrojům na server stejně jako měnit stavy aplikace. Všechny zdroje mají vlastní identifikátor v podobě URI. Jelikož se jedná o HTTP požadavky, tak se používají metody GET, PUT, POST a DELETE. Každá z nich má své opodstatnění a jednotlivé definice REST API jsou popsány v dokumentaci. Tento koncept se používá ke komunikaci mezi klientem a serverem. Jedná se o bezestavovou komunikaci, kdy při každém volání REST služby jsou definovány všechny parametry při volání. Každý požadavek může být explicitně uložený v cache paměti.

2.9 JSF

JSF neboli JavaServer Faces je framework, který odděluje aplikační logiku od uživatelského rozhraní. Cílem tohoto řešení je čistější vývoj webových aplikací. Uživatelský interface je popsán pomocí již definovaných XML tagů, do kterých se naplňují data z Java bean nebo se z těchto tagů data čtou. Vývojáři tento framework umožňuje vytváření menších UI souborů s například jen jednou komponentou a tyto komponenty se pak dají přepoužívat a vkládat do větších JSF souborů, které lze nazývat layout nebo kontejner. Celý tento JSF soubor je generovaný na straně serveru. Klient tak nemusí mít nainstalované nic kromě webového prohlížeče. Při přecházení z jedné stránky na jinou se musí celá webová stránka sestavit na serveru a ta se pak pošle jako nová odpověď a celá stránka se překreslí [12].

2.10 Gson

Gson nebo také Google Gson je open-source knihovna určená k serializaci a deserializaci Java objektů z nebo do JSON. Gson dále podporuje serializaci a deserializaci struktur a jiných objektů [13].

2.11 Jackson

Jackson od společnosti FasterXML je nástroj, který zpracovává JSON. Podporuje parsování dokumentů a serializaci a deserializaci objektů z nebo do JSON [14].

Kapitola 3
Analýza

Pomocí frameworků AFSwinx, AFRest [15] a AspectFaces lze generovat dynamické uživatelské rozhraní. Framework AFSwinx se používá pro generování JavaSE aplikací, AFRest je určen pro serverovou část a AspectFaces se používá na samotné generování uživatelského rozhraní. Poslední zmiňovaný je klíčovým frameworkem pro tvorbu, ostatní jsou určeny převážně pro komunikaci a pro vykreslení vygenerovaných komponent klientovi.

3.1 AFSwinx

Klientská část frameworku, která generuje uživateli tabulky nebo formuláře pomocí definic a dat. Definice a data jsou získávána ze serverové části frameworku – AFRest. AFSwinx umožňuje generování tabulek a formulářů na platformě JavaSE a k vizualizaci se používá technologie Swing.

Generují se panely, které je možné přidat do dalších panelů a ty pak dále přidat do již námi napsané aplikace. Nemusí být tedy celá aplikace napsaná dynamicky, ale je možné si statickou část napsat podle uvážení vývojáře a dynamickou část vygenerovat pomocí frameworku.

K dynamickému vytvoření tabulek nebo formulářů je potřeba nejdříve specifikovat zdroje. Ty se definují pomocí XML souboru. Na Obrázku 3.1 je ukázka specifikace zdrojů.

```
<connection id="loginForm">
    <metaModel>
        <endPoint>10.0.2.2</endPoint>
        <endPointParameters>/AFServer/rest/users/loginForm</endPointParameters>
        <protocol>http</protocol>
        <port>8080</port>
        <header-param>
            <param>content-type</param>
            <value>Application/Json</value>
        </header-param>
    </metaModel>
    <send>
        <endPoint>10.0.2.2</endPoint>
        <endPointParameters>/AFServer/rest/users/login</endPointParameters>
        <protocol>http</protocol>
        <port>8080</port>
        <header-param>
            <param>content-type</param>
            <value>Application/Json</value>
        </header-param>
    </send>
</connection>
```

Obrázek 3.1. Definice zdrojů

V tomto XML jsou definovány REST cesty, přístupové porty, adresy a parametry potřebné k získání komponenty kdekoli v aplikaci. Soubor s definicemi zdrojů je načten jako inputstream a poté poslán s identifikátorem zdrojů na server, odkud se vrátí požadovaná komponenta. Na Obrázku 3.2 je znázorněn příklad pro vytvoření tabulky.

```
InputStream connectionResrouce = getClass().getClassLoader().getResourceAsStream("connection.xml");
AFSwinxForm form = AFSwinx.getInstance().getFormBuilder()
        .initBuilder(loginFormName, connectionResrouce, "loginForm")
        .buildComponent();
```

Obrázek 3.2. Vytvoření tabulky v AFSwinx

Framework dále umožňuje získávání dat z komponent. To je výhodné například při tvorbě tabulky nebo při tvorbě formuláře, kdy získáme aktuální data ve formuláři. Datový objekt nesoucí data pro tabulku je *AFDataPack* a pro formulář je to *AFDataHolder*.

Formulář se získává pomocí klíče, pod kterým byl vytvořen, který nám vrátí konkrétní instanci formuláře. Následně je nutné získaná data deserializovat.

Tabulku získáme buď celou nebo můžeme specifikovat jaká data budou vybrána. Nejprve je opět nutné získat konkrétní instanci tabulky a poté pomocí metody *getSelectedData()* získat požadovaná data.

3.2 AFRest

Tento framework se stará o serverevou stranu. Přijímá a zpracovává požadavky od klienta a posílá nové požadavky na AspectFaces framework, který pak z těchto požadavků generuje tabulky a formuláře. Vytváří definice dat a dodává data klientské části aplikace. K přijmutí požadavku se využívá REST, který přijme požadavek. Klient je zodpovědný za to, co posílá serverové části v tom smyslu, zda si žádá formulář nebo tabulku. V případě generování formuláře musí zdroj vracet právě jeden objekt podle kterého bude formulář sestaven. Serverová strana také musí specifikovat mapování datových typů na komponenty. O tento případ se stará soubor *structure.config.xml*, který je umístěn ve *WEB-INF/af*. Příklad takového mapovacího souboru je možné vidět na Obrázku 3.3. Tento mapovací soubor projde všechny datové typy, které přijdou v požadavku na server a namapuje je tak, jak je uvedeno v XML do jednotlivých XML pro konkrétní datové typy definované ve *WEB-INF/af/profile/structure*. Příklad mapování datových typů na komponenty je uveden na Obrázku 3.4.

```
<mapping>
    <type>boolean</type>
    <default tag="structure/dropDownMenu.xml" />
    <condition expression="${type == 'option'}" tag="structure/optionField.xml" />
    <condition expression="${type == 'confidentianlAgreement'}"
        tag="structure/checkBox.xml" />
</mapping>
```

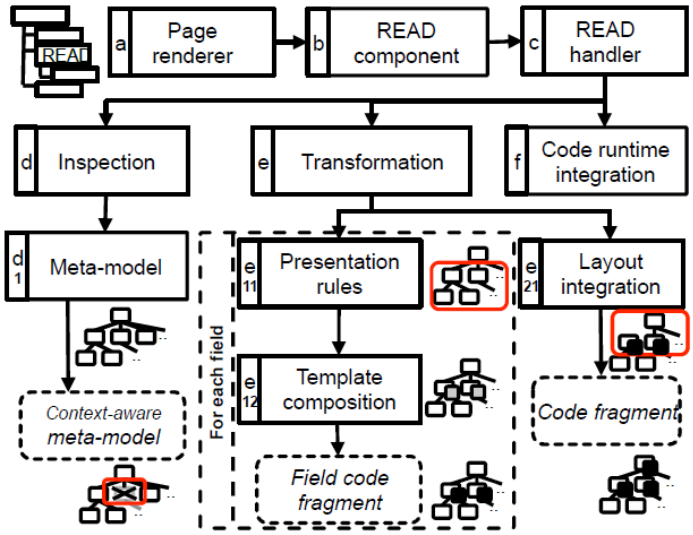
Obrázek 3.3. Příklad structure.config.xml

```
<widget>
  <widgetType>textField</widgetType>
  <fieldName>$field$</fieldName>
  <label>$label$</label>
  <readonly>$readonly$</readonly>
  <validations>
     <required>$required$</required>
     <minLength>$minLength$</minLength>
     <maxLength>$maxLength$</maxLength>
  </validations>
  <fieldLayout>
     <layoutOrientation>$layoutOrientation$</layoutOrientation>
     <labelPosition>$labelPosition$</labelPosition>
     <layout>$layout$</layout>
  </fieldLayout>
</widget>
```

Obrázek 3.4. Příklad definované komponenty

3.3 AspectFaces

Jedná se o transformační nástroj napsaný v Javě, který na základě meta-modelu získaného z data model inspekce vytvoří statické nebo dynamické začlenění několika aspektů do jednoho výstupu. Aspekty se myslí například vlastnosti třídy, jejich jména, typy proměnných, lokalizace, rozvržení, validační pravidla a další. Za pomocí meta-modelu propojí AspectFaces tyto aspekty do textového výstupu XML, zdrojových souborů Java a podobně. Právě díky tomu, že se framework orientuje na komponenty, umožňuje významně zredukovat manuálního úsilí na tvorbu komponent, protože stačí popsat každou komponentu pouze jednou. Detailněji je princip frameworku zobrazena na Obrázku 3.5.



Obrázek 3.5. Struktura AspectFaces [16]

Reálně vývojář začíná vytvořením Java entit tak, jako při tvorbě jakékoli jiné Java aplikace. Následně při tvorbě UI vytvoří konfigurační mapovací soubor ve formátu XML, která je zobrazena v podobě částečné ukázky na Obrázku 3.3. Tyto mapovací soubory se ve formě cesty musí poznamenat do *aspectfaces-config.xml* souboru jako nový záznam

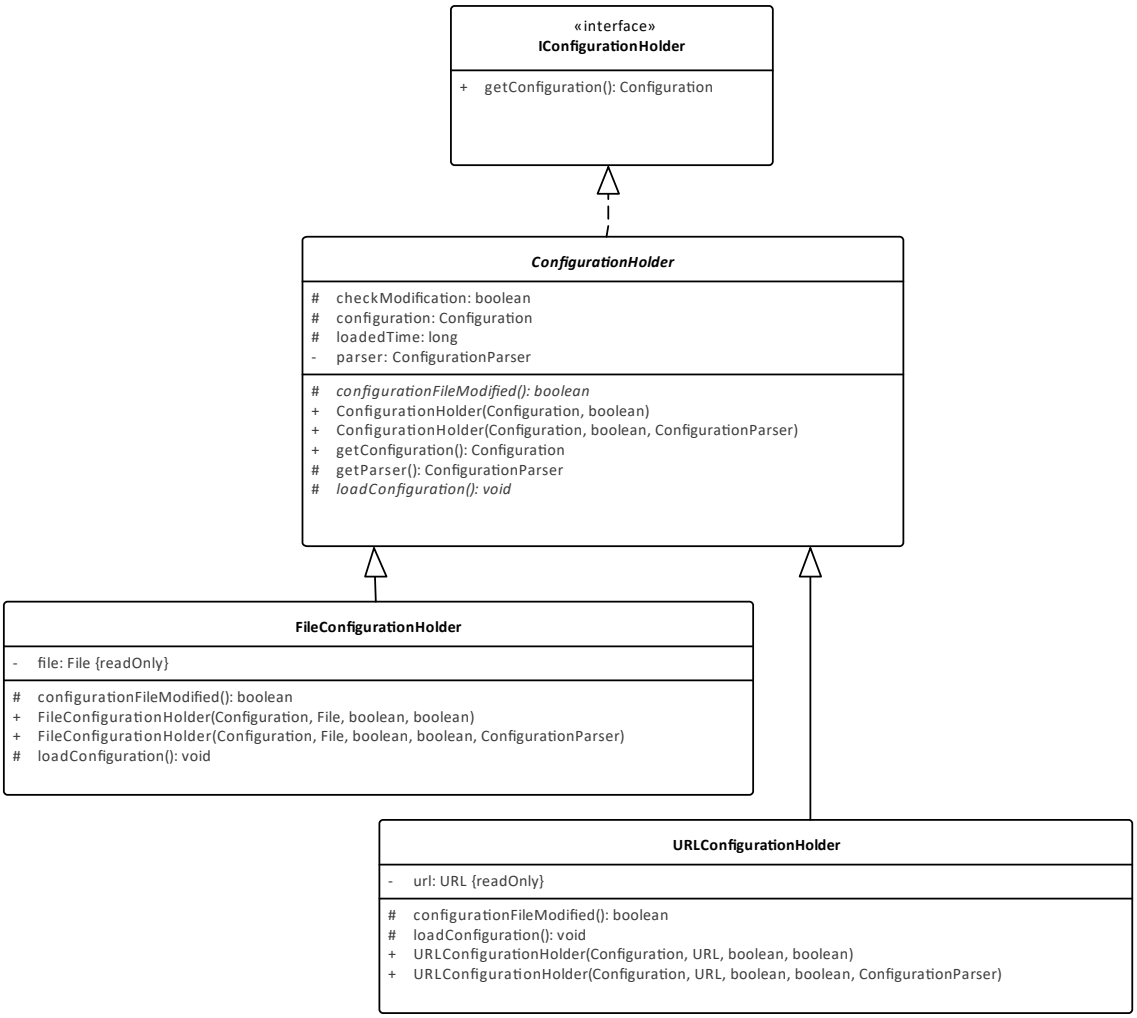
konfigurace. Detailnější podoba *aspectfaces-config.xml* je znázorněna na Obrázku 5.4. Jednotlivé mapovací segmenty představují mapování konkrétní datových typů na UI komponenty.

Těmto komponentám lze přiřadit další atributy, které jsou známy z kódování v HTML. Jedná se například o *required* atribut, který se stará o to, aby daný element byl vyžadovaný po uživateli. Dále framework umí zakrýt znaky při vkládání hesla pomocí atributu *password*. Pro kontrolu délky vstupního řetězce se používá atribut *maxLength* a mezi nejdůležitější atributy, které framework podporuje je *tag*. Ten se stará o to, jak výsledná komponenta bude reálně vypadat, protože má v sobě uloženou cestu k JSF souboru. V tomto JSF souboru jsou vytažené hodnoty z nadefinovaných Java entit. Tyto hodnoty se následně namapují na již zmíněný JSF a naplní ho tak daty. Takto naplněný JSF soubor se vloží do rodičovského JSF souboru společně s jinými JSF soubory a výsledné rodičovské JSF se vykreslí uživateli jako UI.

3.4 Způsob parsování konfiguračních souborů

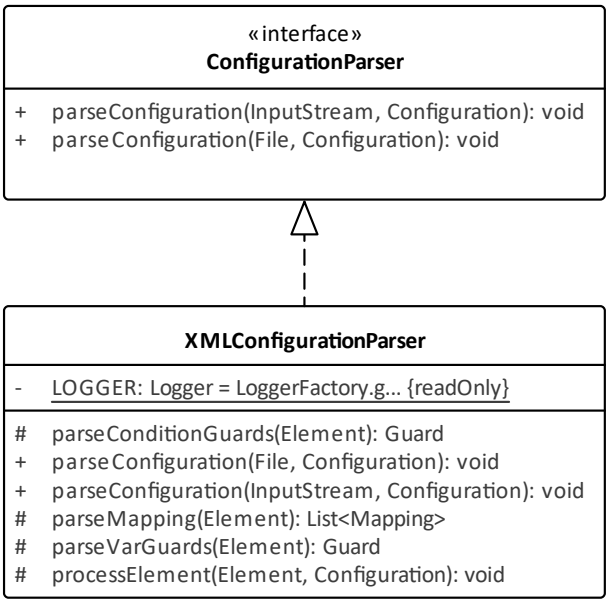
AspectFaces nejdříve zjišťuje zda vůbec existuje *aspectfaces-config.xml* soubor popisující jednotlivé konfigurační mapovací soubory. Pokud neexistuje, tak framework vyhazuje výjimku a končí chybou. Je tedy nezbytně nutné, aby tento soubor byl přítomen. To, že v něm není uvedený žádný konfigurační soubor v <*configuration*/> tagu, to už mu nevadí. V případě absence tohoto tagu, nebude namapován žádný datový typ na komponentu. Pokud ovšem *configuration* tag uveden bude a bude v sobě mít validní cestu k souboru, tak framework provede rozparsování těchto konfiguračních souborů. O toto rozparsování se vývojář nemusí vůbec starat. Vše za něj provede AspectFaces framework.

Rozparsování probíhá v *ConfigurationHolder.java* a to konkrétně v *getParser()* metodě. *GetParser()* má za úkol vybrat správný Parser na základě vnitřních atributů z frameworku, ten pak do configuration uloží a nastaví potřebné parametry na vygenerování UI. V době přebírání kódu frameworku byl ovšem jen jeden Parser, a tak framework vždy sáhl po *XMLConfigurationParser.java*. Tento parser převede jednotlivé atributy na hodnoty v *configuration*. Více detailů znázorňuje přiložený class diagram na Obrázku 3.6, který ukazuje strukturu *ConfigurationHolder.java* a v něm zmiňované metody.



Obrázek 3.6. Struktura Configuration Holders

Na dalším obrázku je vidět architektura Parserů.



Obrázek 3.7. Struktura Configuration Parsers

Kapitola 4
Specifikace požadavků

V této části si popíšeme systémové požadavky, které vyplývají z analýzy. Tvorba požadavků vychází z nedostatků, které framework má a z nároků na framework. Požadavky se dělí na dvě základní části a to na funkční a nefunkční požadavky.

4.1 Funkční požadavky

Tyto požadavky popisují jednotlivé funkcionality modifikace frameworku. Podle nich lze snadno ověřit jejich splnění a nesplnění funkcionalit. Tyto funkční požadavky vychází převážně z cílů práce.

F1 Zpětná kompatibilita
Aplikace vyvinutá na AF frameworku bude stále fungovat i po úpravě frameworku.
F2 XML konfigurace
Úprava frameworku bude nadále podporovat XML konfigurace.
F3 JSON konfigurace
Úprava frameworku bude nově podporovat JSON konfiguraci.
F4 kombinace XML a JSON konfigurací
Úprava frameworku bude umožňovat použití kombinace XML a JSON konfigurací.

4.2 Nefunkční požadavky

Nefunkční požadavky jsou více abstraktní a řeší nároky na implementaci frameworku. Tyto nároky se dají měřit a porovnávat pomocí specifických metrik.

NF1 Rychlost JSON konfigurace
Framework bude rychlejší za použití JSON konfigurace na měřitelné úrovni.
NF2 Menší soubory v případě použití JSON konfigurace
Konfigurační soubory budou na disku zabírat méně místa než XML soubory.
NF3 Rychlejší psaní konfigurací v případě využití JSON formátu
Vývojář bude schopný napsat rychleji konfiguraci v JSON nežli v XML.

Kapitola 5
Implementace

Důležitým aspektem této práce je nepochybně implementace. Ta se skládá z úpravy již existujícího řešení a rozšíření o mapování JSON konfigurace. Mezi hlavní cíle patří možnost používat framework s již existujícím XML řešením, aniž by se stávající implementace nefungovala nebo by bylo nutné provést nějaké úpravy v implementaci. Tento cíl je brán jako ten hlavní a nejdůležitější. Dále bude zajímavé sledovat jaké vlastnosti jednotlivé typy řešení budou mít. Dokáži si představit, že jedno z nich bude výkonnostně náročnější a druhé například nebude tolik náročné na přenos dat.

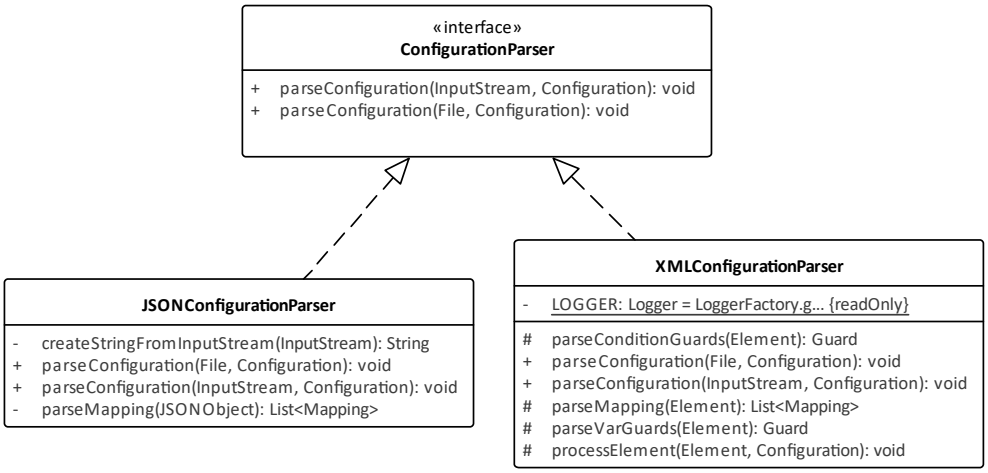
5.1 Použité nástroje k vývoji

Celá práce byla vyvíjena za pomocí IntelliJ IDEA ve verzi 2018.2.5. Na tvorbu a modifikaci JSON souborů bylo použito několik nástrojů. Ten hlavní byl přímo v IntelliJ IDEA, dalším použitým bylo Visual Studio Code a dále Sublime Text 3. Testování probíhalo v prohlížeči Chrome a to konkrétně ve vývojářské konzoli. Vývojovým jazykem je Java 8.

5.2 Rozvětvení

Spoustu času zabralo rozmyšlení, kde framework rozdvojit a kde začít implementovat vlastní řešení. Nakonec nezbývalo nic jiného, než změnit konfigurační soubory a debugovat každý kousek frameworku řádek po řádku.

Nové řešení našlo místo v původní projektové struktuře AspectFaces frameworku. Bylo nutné přidat nový způsob parsování konfiguračních souborů a k tomu byl využit *JSONConfigurationParser.java* tak, jak je uvedeno na class diagramu na Obrázku 5.1. *JSONConfigurationParser.java* je implementací *ConfigurationParser.java* rozhraní. Toto rozhraní se stará o definování parseru konfiguračních souborů.



Obrázek 5.1. Nová struktura Configuration Parser

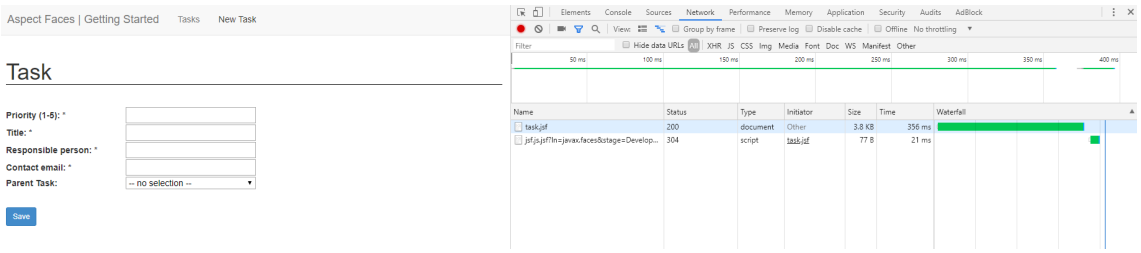
Veškerá logika parsování je v metodě *parseMapping()*. Tato metoda se dále štěpí na menší metody, které se starají o parsování jednotlivých mapping atributů. Ke čtení JSON konfigurací byla použita knihovna *org.json* (jejíž maven závislost je možné vidět na Obrázku 5.2), která umožnila čtení jednotlivých částí po částech. Při výběru nějakého serializátoru připadalo použít například i Gson od Google nebo Jackson od FasterXML. Tyto nástroje by bylo vhodnější použít k deserializaci dat z JSON přímo na nějaké Java entity. Ovšem tento případ nenastal, a proto bylo potřeba kontrolovat každý atribut z JSON samostatně. Jednotlivé atributy pak byly uloženy do tříd frameworku tak, aby nebyla porušena jeho funkčnost. Je tedy navázáno na stejnou logiku frameworku, která je použitá u XML konfigurací.

```
<dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20180813</version>
</dependency>
```

Obrázek 5.2. Maven závislost org.json

5.3 Měření rychlostí

Měření probíhalo pomocí prohlížeče Google Chrome a jeho vývojářského nástroje. V kartě Network bylo sledováno, jak dlouho trvá než se vrátí odpověď. V čase jsou tedy zahrnuty časy na vykreslení a sestavení stránky a ne jen parsování konfigurací. Tyto hodnoty však ničemu nevadí, protože se jedná o stejnou aplikaci jak v případě XML konfigurace tak i JSON konfigurace, takže časy na sestavení stránek můžeme opomenout. Toto měření tedy představuje implementaci pomocí org.json. K tomuto testu bylo potřeba sestavit větší/objemnější JSON a XML konfigurace. Ty se nyní skládají zhruba z 60 datových typů, které jsou potřeba namapovat. Datové typy jsou uměle vytvořené a vymyšlené. Například String01 - String60 a všechny mají podrobnější atributy, aby bylo možné otestovat modifikaci frameworku více do hloubky. Žádný z atributů se však nepoužíval reálně aplikací. To však nevadí, protože AspectFaces nevyžadují, aby každé mapování bylo použito. Nicméně se všechny mapování drží na straně frameworku. Testování probíhalo v Chrome prohlížeči ve vývojářské konzoli v kartě Network a měřil se čas potřebný k vykreslení stránky. Příklad testování je uveden na Obrázku 5.3.



Obrázek 5.3. Způsob měření rychlosti v prohlížeči

Každý konfigurační soubor se měří 20x a z těchto hodnot jsou pak vypočítány průměry. Pro úplné otestování se prováděly testy s vypnutým i zapnutým lazy-load. Registrace konfiguračních souborů je v *aspectfaces-config.xml* přidáno pomocí tagu *configuration* a znázorněno na Obrázku 5.4.

```
<configurations-registration>
    <configuration name="default" path="/WEB-INF/af/html.config.json" check-modification="true" lazy-load="false"/>
</configurations-registration>
```

Obrázek 5.4. Registrace konfiguračních souborů

Testovací soubor JSON zabírá na disku 32 KB a XML soubor zabírá na disku 21 KB. V tabulce lze vidět naměřené hodnoty. Všechny hodnoty jsou v ms a jedná se o čas odpovědi frameworku.

Trochu se tady nabízí otázka, proč je vlastně XML soubor menší než JSON, když by měl mít JSON méně znaků. Opak je ale v tomto případě pravdou. XML má opravdu méně znaků a to především proto, že AspectFaces mají konfigurace psané především v tagu a ne mezi nimi. Nicméně JSON se dá zmenšit a to tak, že se napíše na jeden řádek. Existuje mnoho online nástrojů, které toto umí udělat, a tak to v tabulce je také zmíněno a to především pro zajímavost.

Menší soubory se hodí především pro klient – server architekturu, kdy se posílají jednotlivé soubory od klienta na server, tam se vyhodnotí a posílají se zpět klientovi. Tento proces je časově závislý na přenosu dat mezi klientem a serverem. Přitom stránka se nevykreslí dříve než jsou potřebné konfigurační soubory přeneseny. V tabulce 5.1 jsou uvedeny průměry měření. Všech 20 měření je obsahem přiloženého CD v .xlsx formátu.

	průměr (ms)
XML a lazy-load = true	380
XML a lazy-load = false	338
JSON a lazy-load = true	593
JSON a lazy-load = false	314
inline JSON a lazy-load = true	362
inline JSON a lazy-load = false	333

Tabulka 5.1. Naměřené časy z testování rychlostí konfiguračních souborů.

Zde se ukázalo, že JSON splnil rychlostní očekávání, ale rozdíl rychlostí není uspokující, a tak bylo nanejvýš vhodné zamyslet se nad jiným možným způsobem implementace. Jiný způsob implementace znamenalo použít jinou knihovnu a to konkrétně Gson (Obrázek 5.5), který je velice vhodný na deserializaci dat. Jak už bylo napovězeno v předešlé kapitole, tak dalším přístupem, jak JSON konfiguraci rozparsovat je pomocí Gson knihovny namapovat jednotlivé atributy na konkrétní Java třídy. Vytvořil jsem tedy třídy, které popisují vnitřní struktury v JSON konfiguraci. Celý JSON konfigurační soubor jsem pomocí Gson knihovny deserializoval na vytvořené třídy. Tyto třídy sloužily pouze jako úložiště hodnot z JSON. Ve třídě *JSONConfigurationParser.java* je vytvořena nová logika. Ta ze zmíněných tříd přebrala data na konfigurační hodnoty AspectFaces frameworku.

```
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.5</version>
</dependency>
```

Obrázek 5.5. Maven závislost Gson

	průměr (ms)
JSON a lazy-load = true	486
JSON a lazy-load = false	325

Tabulka 5.2. Naměřené časy z testování rychlostí konfiguračních souborů při použití Gson knihovny.

Na řadu přišlo ověření funkčnosti pomocí testů a měření reálného parsování JSON souboru. Opět bylo provedeno 20 měření pokaždé s vypnutým nebo zapnutým *lazy-load*. V tabulce 5.2 jsou zobrazeny výsledky z měření.

Z výsledků je patrné, že nejlepšího výsledku lze dosáhnout s JSON konfigurací s vypnutým *lazy-load* a s org.json knihovnou. Toto zjištění vede k naimplementování finálních úprav právě s pomocí této knihovny.

5.4 Způsob identifikace konfigurace

Nejdříve se zdálo jako nemožné, aby framework používal kombinaci typů konfigurací. To se však podařilo díky zvolenému přístupu identifikace konfigurace. Framework vždy jako parametr dostává cestu ke konfiguračnímu souboru a ten si pak otevře a jednotlivé atributy přečte. Přečtené atributy namapuje na Java objekty a s těmi pak framework vytváří konkrétní UI komponenty, které vkládá do JSF souborů a ty pak zobrazuje uživateli v HTML prohlížeči. Právě zmíněná cesta byla klíč k přístupu, jak identifikaci naimplementovat.

Modifikace frameworku se vždy podívá na cestu k souboru a podle přípony konfigurace se přiřadí k parseru správná konfigurace. Nespornou výhodou této modifikace je fakt, že je možné pro každou strukturu definovat vlastní konfigurační soubor a každý z nich může mít jinou příponu. Tím je myšleno to, že například *table.config.xml* bude parsovaný pomocí *XMLConfigurationParser.java* a naopak *form.config.json* bude parsovaný *JSONConfigurationParser.java*.

Tato úprava a zjišťování je implementováno v *ConfigurationHolder.java* v metodě *getParser()*, která inicializuje konkrétní parser pro každý konfigurační soubor zvlášť.

5.5 Není proxy jako proxy

Tento přístup ovšem naráží na jeden docela podstatný problém. V názvu této práce je řečeno, aby bylo vytvořené proxy, které pak určí jaký parser by se měl použít. Ovšem tato práce proxy neimplementuje v pravém slova smyslu a to hned z několika důvodů.

Ten nejzásadnější je, že žádné proxy není potřeba. Ano, je možné vytvořit nějaký nový modul, v něm vytvořit nějaké třídy a framework upravit tak, aby procházelo vše přes nově vytvořený modul. Fungovalo by to naprosto stejně a název této práce by dával větší smysl. Nutno ale podotknout, že by takové řešení bylo naprosto zbytečně složité, když stačilo rozšířit stávající řešení o pár rozšíření, které jsou pak mnohem jasnější a přehlednější někomu, kdo framework bude studovat více do hloubky.

Další důvod, který vedl k tomuto rozhodnutí byl ten, že by značná část testování frameworku nebyla funkční a musela by se většina vytvářet znovu.

Proto je tento způsob asi tím nejelegantnějším řešením, které bylo možné. Dbá na přehlednost kódu, jednoduché rozšíření o další možné parsery a využívá potenciálu již hotového frameworku.

5.6 Případy užití

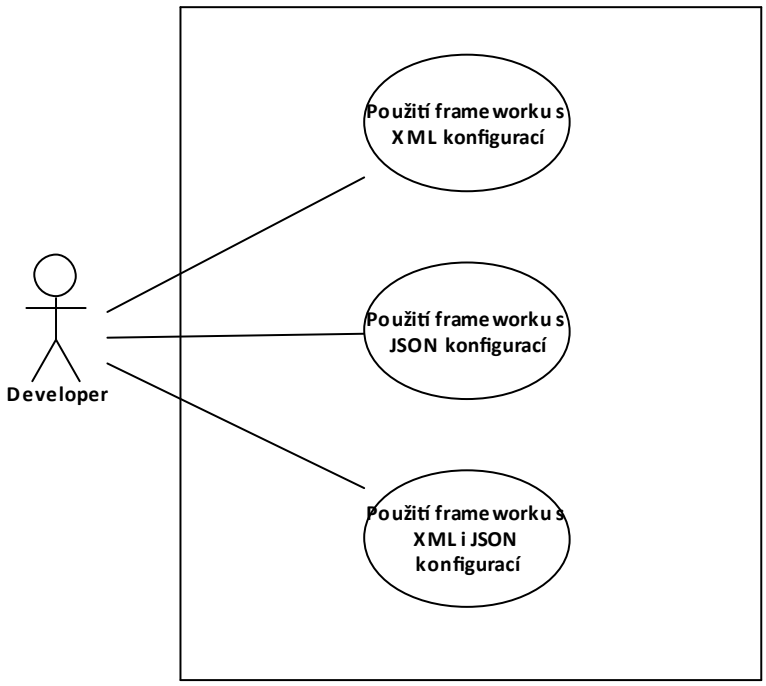
Tato sekce se zaměřuje na způsob, jakým bude možné framework použít. Úprava frameworku se dělí na základní 3 způsoby případů užití.

5.6.1 Aktéři

V této sekci se nachází představení všech aktérů, kteří přijdou do styku s frameworkem, anebo jen s její úpravou.

5.6.2 Developer

Developer je osoba, která se s frameworkem setká nejvíce. To on bude používat úpravy a nová řešení frameworku. Na jeho osobu se bude cílit a bude snaha vytvořit co největší možnou úsporu času při tvorbě aplikací.



Obrázek 5.6. Příklady užití frameworku

Použití frameworku s XML konfigurací

ID: 1

Popis: Programátor využívá framework starým způsobem, kdy generuje formulář nebo tabulku z XML konfigurace. Tento případ může nastat pokud se zvyšuje verze frameworku za účelem získání nové funkcionality, ale bez nutnosti překonfigurování stávající aplikace.

Aktér: Programátor

Vstupní podmínky:

1. Konfigurační soubor musí být správně sestaven podle dokumentace tak, aby s ním framework uměl pracovat. Dále musí být konfigurační soubor zaregistrovaný v aspectfaces-config.xml.

Scénář:

1. Scénář začíná rozparsováním konfiguračních souborů uvedených v aspectfaces-config.xml na konfigurační třídy s příslušnými hodnotami jednotlivých atributů.

2. Po úspěšném rozparsování framework namapuje jednotlivé datové typy na příslušné šablony.
3. Výsledek tohoto mapování se zobrazuje uživateli jako UI element.

Výstupní podmínky:

1. Uživateli se zobrazil UI element s příslušnými daty nebo se data uloží na pozdější práci s nimi.

Použití frameworku s JSON konfigurací

ID: 2

Popis: Programátor vyvíjí novou aplikaci s využitím AF frameworku a chce používat JSON konfiguraci a to především pro její čitelnost a přehlednost. Pomocí tohoto přístupu si programátor zajistí modernější přístup k vývoji aplikace a dlouhodobější udržitelnost kódu.

Aktér: Programátor

Vstupní podmínky:

1. Správně sestavený konfigurační soubor podle podmínek popsaných v dokumentaci tak, aby framework mohl provést inspekci nad jednotlivým mapování popsaným v konfiguračním souboru.

Scénář:

1. V tomto scénáři vývojář sestaví konfigurační soubor a zapíše jeho cestu a název do aspectfaces-config.xml.
2. Framework po úspěšném rozparsování provede mapování na jednotlivé mapovací šablony.
3. Toto mapování je poté využito ke konstrukci UI elementů, které se následně zobrazí uživateli.

Výstupní podmínky:

1. Uživateli se zobrazil UI element, který obsahuje příslušná data nebo určitý počet polí ve formuláři.

Použití frameworku s XML i JSON konfigurací

ID: 3

Popis: Programátor rozšiřuje stávající řešení o nové komponenty a potřebuje nakonfigurovat mapování pro framework. Stávající řešení funguje na XML konfiguracích. Programátor se může rozhodnout, zda bude pokračovat v XML konfiguracích nebo napsat nové řešení na JSON konfiguraci.

Aktér: Programátor

Vstupní podmínky:

1. Aplikace používající AF framework.
2. Funkční stávající řešení s XML konfigurací.

Scénář:

1. Vývojář má v úmyslu rozšířit stávající řešení o novou funkcionalitu a potřebuje vytvořit nový mapovací konfigurační soubor.
2. Rozhodne se pro napsání nové konfigurace v JSON souboru. Ten pak uvede v aspectfaces-config.xml spolu s XML konfiguracemi.
3. Vytvoří validní JSON konfiguraci pro AF framework pro následné mapování.

4. Framework provede mapování příslušných datových typů z JSON a XML konfiguračních souborů na mapovací šablony.
5. Toto mapování je pak použito ke konstrukci UI elementů.

Výstupní podmínky:

1. Uživateli se zobrazí UI element s vyplněnými daty nebo UI element připravený k zadávání dat.

5.7 Architektura

Framework AspectFaces podporuje pouze XML mapping souborů, který je znázorněn na Obrázku 3.3. Tento přístup je ve větším měřítku značně nepřehledný a vývojáři může zabrat více času než nově navržený JSON přístup. Příkladem JSON mapování je Obrázek 5.8. Tyto mapovací soubory se registrují v konfiguračním souboru *aspectfaces-config.xml* v *configurations-registration* bloku:

```
<configurations-registration>
    <configuration name="default" path="/WEB-INF/af/html.config.json" check-modification="true" lazy-load="true"/>
    <configuration name="table" path="/WEB-INF/af/table.config.json" check-modification="true" lazy-load="true"/>
</configurations-registration>
```

Obrázek 5.7. Konfigurační část aspectfaces-config.xml souboru

Nic dalšího není potřeba ve frameworku nastavovat. Úprava je naimplementovaná tak, že přijímá jak XML soubory tak i JSON. To tedy znamená, že programátor může pro konfigurační mapování tabulky (*table.config.**) použít starší – původní řešení frameworku a to jen tím, že dá souboru příponu *.xml* a pro html.config soubor si může například zvolit *.json* příponu a tam použít nový a přehlednější přístup, který je uveden na Obrázku 5.8.

```
{
  "configuration": {
    "mapping": [
      {
        "type": [
          "String"
        ],
        "default": {
          "tag": "html/text.xhtml",
          "maxLength": "255",
          "size": "30",
          "required": "false"
        },
        "condition": [
          {
            "expression": "${not empty email and email == true}",
            "tag": "html/email.xhtml"
          },
          {
            "expression": "${not empty password and password == true}",
            "tag": "html/password.xhtml"
          }
        ]
      }
    ]
  }
}
```

Obrázek 5.8. Nový návrh JSON konfigurace

Toto nové řešení rozšiřuje framework o použití JSON konfiguračních souborů. JSON se skládá z key: value hodnot. V tomto konfigurační souboru je klíčem *configuration* a hodnotou je *mapping*. Dále se struktura rozpadá na pole jednotlivých typů a k nim příslušných hodnot. Na Obrázku 5.8 je jako příklad uveden datový typ a tím je String. Každý jednotlivý datový typ má v default klíči hodnoty pro *tag*, který je povinný a další volitelné atributy. Klíč *tag* udává, na co se má daný datový typ namapovat. V tomto případě datový typ String se namapuje na *text.xhtml*, který je možné nalézt ve složce *html/*. V Tabulce 5.3 jsou popsané atributy s jednotlivými významy pro konkrétní atribut.

configuration	název použitého JSON souboru
mapping	pole jednotlivých mapování
type	výčet datových typů, pro které je mapování určeno
default	výčet vlastností, které má náš datový typ
tag	cesta k souboru, na který se má datový typ namapovat
maxLength	maximální délka vstupu
required	true/false hodnoty, zda je tento atribut požadovaný

Tabulka 5.3. Popis použitých atributů a jejich vlastnosti.

5.8 Nutné úpravy ve frameworku

Nejdříve bylo nutné vyřešit problém s rozpoznáváním typu konfigurace. To se nyní děje tak, že framework poznává konfiguraci na základě přípony souboru. To v praxi znamená, že se konfigurační soubor může jmenovat jakkoli, ale musí končit na *.xml* nebo na *.json*. Pro správnou funkčnost bylo nutné upravit způsob parsování konfiguračních souborů. Tyto konfigurační soubory se analyzují ve třídě *JSONConfigurationParser.java* využívající interface *ConfigurationParser.java*. Bylo tedy nutné naimplementovat třídu tak, aby framework měl validní informace o konfiguračním souboru.

Tento proces rozparsování JSON souboru bylo nezbytné udělat tak, aby jednotlivé typy a atributy byly kompatibilní s již hotovým řešením frameworku. Jsou tedy přepoužité třídy z frameworku tak, aby byla zaručena kompatibilita nového řešení na to původní. Díky tomuto lze také vycházet z napsaných testů pro framework a jen je rozšířit o nový způsob.

Dále při přidávání nového konfiguračního souboru nesmíme zapomenout přidat tuto informaci do *aspectfaces-config.xml* jako nový záznam.

Kapitola 6
Testování

Testování aplikace je nedílnou součástí dodání kvalitního software. Pomocí testování se zjišťuje správná funkčnost a stabilita software. Testování nám udává kvalitu software, kdy se zjišťují slabá místa software. Čím méně jich software má, tím více kvalitní je. Dále nám umožňuje ověřit, zda byl software navrhnut přesně podle zadání a zda bylo zadání dodrženo. Je proto důležité znát cíle projektu a podle těchto cílů vytvořit testovací scénáře [17].

Testování se dělí na několik kategorií. To nejzákladnější dělení je s uživatelem a bez uživatele. V této práci bude primárně využito testování bez uživatele. Tento typ testování se mimo jiných typů dále dělí na integrační a unit testy.

Framework je otestovaný z pohledu vytvořeného dema. To znamená, že testy budou probíhat z pohledu koncového vývojáře, kdy jsou očekávány výstupy a chybové hlášky frameworku tak, aby vývojář věděl, co je za problém a co by měl změnit, aby jeho řešení fungovalo.

6.1 Unit testy

Unit, neboli jednotkové testy testují jednotlivou komponentu software. To ve výsledku znamená, že se aplikace testuje jen na určité kroky jedné komponenty. Pokud bychom chtěli testovat více komponent najednou, tak bychom museli sáhnout po integračních testech. Unit testy nám tedy testují základní funkcionalitu jedné komponenty [18].

6.1.1 Unit test case

Pro provedení unit testů je zvykem psát testovací scénáře abychom udrželi kvalitu a přehlednost testů. Tyto scénáře nám ukáží v jasné podobě čeho chceme dosáhnout a při jakých konkrétních podmínkách. Každý testovací scénář má v sobě ID, který daný testovací scénář identifikuje. Popis, který říká, co daný testovací scénář dělá. Vstupní podmínky, které popisují v jakém stavu se nachází v době testování testovaná aplikace. Předpoklad, který říká, jak se s testovacím scénářem zachází. Kroky scénáře popisují jednotlivé kroky, které jsou potřeba udělat k vykonání testovacího scénáře a samozřejmě očekávaný výsledek scénáře. Díky očekávanému výsledku lze zjistit, zda se testovací scénář vykonal úspěšně nebo neúspěšně.

TC1: Chybějící konfigurační soubor

Vyhozena chyba při pokusu zobrazení stránky, která využívá JSON konfigurační soubor/y.

Vstupní podmínky: Není uveden žádný konfigurační soubor v aspectfaces-config.xml.

Předpoklad: Je správně vytvořený aspectfaces-config.xml soubor a validní projekt.

Kroky scénáře:

1. Kontrola zda je v aspectfaces-config.xml uvedený nějaký konfigurační soubor.

2. V případě, že nějaký konfigurační soubor je uveden, tak smazat jeho záznam.
3. Sestavit projekt.
4. Přistoupit na webovou stránku projektu.

Očekávaný výsledek: Vyhozená chyba při pokusu načtení stránky a záznam v logu aplikace, že nebyl nalezen konfigurační soubor.

TC2: Chybějící type klíč v JSON mapping konfiguraci

Vyhozena chyba s popisem při zobrazení stránky, která využívá konfigurační JSON soubor.

Vstupní podmínky: Chybějící type klíč v JSON konfiguraci v mapping poli.

Předpoklad: Je správně vytvořený aspectfaces-config.xml soubor, validní projekt a konfigurační soubor je uveden v aspectfaces-config.xml souboru.

Kroky scénáře:

1. Kontrola zda je v aspectfaces-config.xml uvedený konfigurační soubor.
2. Sestavit projekt a nasadit aplikaci.
3. Přistoupit na webovou stránku.

Očekávaný výsledek: Vyhozená chyba při pokusu načtení stránky a záznam v logu aplikace, že chybí klíč type v konfiguračním souboru.

6.2 Integrační testy

Integrační testy [19] jsou používány k ověření funkčnosti více softwarových komponent dohromady, kdy dochází k otestování, že nově přidané funkcionality mezi sebou nekolidují. Nejdříve se začínají testovat dvě komponenty mezi sebou a postupně se přidávají další a další komponenty.

6.2.1 Test case integračních testů

Pro udržení přehlednosti testování a zdokumentování jednotlivých kroků jsou opět použity testovací scénáře. Použití je velice podobné jako u unit testů. Každý testovací scénář má vlastní ID, pod kterým se identifikuje. Cíl, kterého chceme dosáhnout pomocí testovacího scénáře. Popis, který vysvětluje, co daný testovací scénář dělá a v neposlední řadě očekávaný výstup.

TC3: Provedení mapování konfiguračního souboru

Cíl: Převézt modelovou entitu pomocí konfiguračního souboru na UI element.

Popis: Pomocí konfiguračního mapovacího souboru převede framework entitu na konkrétně zvolený UI element (tabulka/formulář) a zobrazení UI elementu uživateli.

Očekávaný výstup: Zobrazená UI komponenta uživateli s uloženými daty nebo s vytvořeným formulářem pro uložení uživatelských dat.

6.3 Testování rychlosti tvorby konfiguračních souborů

V této části se podíváme na to, jak to ve skutečnosti vypadá s rychlostí tvorby konfiguračních souborů uživatelem. Tři vývojáři měli za úkol v libovolném editoru přepsat z poskytnutého PNG obrázku nadefinovanou konfiguraci do XML a do JSON formátu.

	XML (min)	JSON (min)
Vývojář 1	7:55	7:01
Vývojář 2	7:06	4:38
Vývojář 3	7:25	5:47

Tabulka 6.1. Naměřené časy z testování rychlosti tvorby XML a JSON konfiguračních souborů.

Výsledky jsou uvedené pro každý formát a vývojáře v tabulce 6.1. Všechny jednotky jsou udávány v minutách.

Z výsledku je více než patrné, že JSON je opravdu pro vývojáře rychlejší a tím pádem i pohodlnější na psaní. Nutno podotknout, že psaní těchto souborů probíhalo bez jakýchkoli schémat. Pro ještě větší urychlení práce je do frameworku zahrnuto schéma pro JSON tak, aby bylo psaní ještě o něco rychlejší a aby editory takzvaně „našeptávaly“ další atributy a hlídaly povinné atributy.

Schéma je umístěné v *core/configuration/schema/json/* jako *config.schema.json* a použití je velice snadné. Stačí si jen schema zkopírovat do oblíbeného textového editoru a ve Vašem konfiguračním souboru toto schéma začít používat.

Kapitola 7
Závěr

Během této práce byla provedena analýza frameworku na dynamické generování uživatelského rozhraní. Následovalo nalezení možných úprav a poté návrh implementace a implementace samotná. Následně byly navrhnuty a provedeny testy a jejich shrnutí.

Během výsledného měření bylo provedeno porovnávání výsledků pro určité úkony vyžadované od vývojáře. Považuji za velký úspěch použitelnost modifikace úpravy se starým řešením, možností rozšíření staršího kódu o novou funkcionalitu nebo jen jejich kombinacemi a celkově výsledky splnily očekávání. Zároveň ale musím jedním dechem dodat, že mě zarazila velikost nového JSON souboru. Tento fakt je způsoben tím, že XML konfigurace obsahuje atributy přímo v XML tagu. To pak dovoluje nepoužívat „zavírací“ tag (<nazevTagu/>).

Jistým rozšířením zadání bylo vytvoření JSON schema pro jednodušší tvorbu JSON konfigurací, kdy každý sofistikovanější JSON editor umí schema použít a pomocí něj programátora vést při psaní JSON konfigurace.

7.1 Budoucí vylepšení

Mezi hlavní nová vylepšení bych nepochybně zařadil uložení JSON schema pro manipulaci s JSON konfigurací na internet tak, aby si vývojář nemusel kopírovat schema k sobě na lokální disk.

Tyto úpravy, které byly udělány v této práci lze rozšířit o další konfigurační soubory. Mezi hlavní bych zdůraznil přepsání XML aplikačních konfiguračních souborů například do YAML, kromě aplikačních souborů lze vylepšit i jiné než mapovací soubory frameworku. Hlavním kandidátem může být například *aspectfaces-config.xml*.

Krom těchto modifikací lze uvažovat o dalším rozšíření frameworku. V dnešní době, kdy je velice populární Javascript na tvorbu dynamických webových stránek si dokáži představit použít podobný princip generování nebo alespoň obohacování UI komponent i v AspectFaces frameworku. Dalším možným vylepšením může být spuštění frameworku v Dockeru, kdy by vývojář mohl nastartovat server pomocí jednoho příkazu.

Literatura

[1] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds a Clemente Izurieta. *Comparison of JSON and XML Data Interchange Formats: A Case Study.* In: Montana State University – Bozeman: 3-6.

[2] *JSON vs XML.*
https://www.educba.com/json-vs-xml/.

[3] *AspectFaces.*
http://www.aspectfaces.com/overview.

[4] Marek Pícka. *METAMODELOVÁNÍ V PRAXI.* In: 1-7.

[5] Colin Atkinson a Thomas Kühne. Model-Driven Development: A Metamodeling Foundation. 1-6.

[6] *Proxy Design Pattern.*
https://sourcemaking.com/design_patterns/proxy.

[7] Martin Tomášek. *ASPEKTOVĚ ORIENTOVANÝ VÝVOJ UŽIVATELSKÝCH ROZHRANÍ PRO JAVA SE APLIKACE.* 2015.

[8] *Aspect-Oriented Programming.*
https://flowframework.readthedocs.io/en/stable/TheDefinitiveGuide/PartIII/AspectOrientedProgramming.html.

[9] *What Is an XML File and How Can You Open and Use It?* 2019.
https://www.makeuseof.com/tag/xml-file-case-wondering/.

[10] *Introducing JSON.*
https://www.json.org/.

[11] *REST API Tutorial.*
https://www.restapitutorial.com/.

[12] *JSF.*
http://www.javaserverfaces.org/.

[13] *Gson User Guide.*
https://github.com/google/gson/blob/master/UserGuide.md.

[14] *Jackson Project Home.*
https://github.com/FasterXML/jackson.

[15] *AFSwinx.*
https://github.com/tomasma5/AFSwinx.

[16] Tomáš Černý a Michael Donahoo. *Towards effective adaptive user interfaces design.* 2013.
https://www.researchgate.net/publication/259705301_Towards_effective_adaptive_user_interface

[17] *What is Software Testing? Introduction, Definition, Basics & Types.*
https://www.guru99.com/software-testing-introduction-importance.html.

[18] *Unit Testing.*
http://softwaretestingfundamentals.com/unit-testing/.

[19] *Testování softwaru*. 2011.
http://testovanisoftwaru.cz/tag/integracni-testovani/.

Příloha A
Zadání práce

ČVUT
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Jaroš** Jméno: **Jan** Osobní číslo: **457904**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Proxy pro automatické generování uživatelského rozhraní

Název bakalářské práce anglicky:

Pokyny pro vypracování:

AspectFaces je framework, pomocí kterého snížíme čas vývoje uživatelského rozhraní o 10-25% [2]. Pomocí něj definujeme prezentační vrstvu pouze jednou a to je distribuované do uživatelského rozhraní, jak chceme, a navíc se vždy přizpůsobuje potřebám backendu bez usílí navíc.
Cíle práce jsou následující:
1) Analyzujte danou problematiku adaptivního generování uživatelského rozhraní [1,3]
2) Prozkoumejte možnosti frameworku aspectfaces [2] vytvořte rešerši na metodiky vývoje softwaru, které framework používá [5]
3) Prostudujte možnosti jak odstranit závislost na xml struktuře z které se generují uživatelská rouzhraní [4]. Nový přístup navrhněte tak, aby podporoval OOP a REST/JSON
4) Rozšiřte adaptivní framework [3] na základě bodu 3)
5) Využijte framework a vytvořte pomocí něj testovací aplikaci a tu otestujte a vyhodnoťte testy

Seznam doporučené literatury:

1. Tomasek, Martin, and Tomas Cerny. "On web services ui in user interface generation in standalone applications." Proceedings of the 2015 Conference on research in adaptive and convergent systems. ACM, 2015.
2. http://www.aspectfaces.com
3. TOMÁŠEK, M. and T. ČERNÝ. On Web Services UI In User Interface Generation in Standalone Applications. In: ČERNÝ, T. and E.S. NADIMI, eds. Proceeding of the 2015 Research in Adaptive and Convergent Systems (RACS 2015). Research in Adaptive and Convergent Systems, Prague, 2015-10-09/2015-10-12. New York: ACM, 2015. p. 363-368. ISBN 978-1-4503-3738-0. DOI 10.1145/2811411.2811537.
4. PAVEL, Matyáš. Automated context-aware user interface generation using UI classification. Praha, 2018. Diplomová práce. ČVUT FEL. Vedoucí práce Martin Tomášek.
5. Atkinson, Colin, and Thomas Kuhne. "Model-driven development: a metamodeling foundation." IEEE software 20.5 (2003): 36-41.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jiří Šebek, kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **07.02.2019** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **20.09.2020**

Ing. Jiří Šebek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

Příloha B
Zkratky a symboly

B.1 Zkratky

MDD	Model Driven Development
ID	Identifier
JSON	JavaScript Object Notation
XML	Extensible Markup Language
OOP	Objektově orientované programování
AOP	Aspektově orientované programování
SoC	Separation of Concerns (oddělení odpovědností)
REST	Representational State Transfer
YAML	YAML Ain't Markup Language
JavaEE	Java Platform, Enterprise Edition
JSF	JavaServer Faces
UI	User Interface
HTML	Hypertext Markup Language
PNG	Portable Network Graphics
XSLT	eXtensible Stylesheet Language Transformations
ms	milisekundy
min	minuty

Příloha C
Obsah přiloženého CD

Obsah přiloženého CD vypadá takto:

- bachelors-work-AF.zip – zdrojové soubory s AspectFaces frameworkem
- bachelors-work-demo.zip – zdrojové soubory k demu aplikace
- bakalarskaPrace – soubor, který obsahuje bakalářskou práci v pdf formátu a také zdrojové tex soubory k textu
- NamereneHodnoty.xlsx – Excel soubor s hodnotami měření a jejich výsledky